Université Virtuelle Africaine

INFORMATIQUE APPLIQUÉE: CSI 4201

PROGRAMMATION DE SYSTEME

Dr.Cherif Diallo

Avant propos

L'Université virtuelle africaine (UVA) est fier de participer à l'amélioration de l'accès à l'éducation dans les pays africains à travers la production de matériel didactique de qualité. Nous sommes également fiers de contribuer aux connaissances mondiales comme nos ressources pédagogiques sont pour la plupart accessibles de l'extérieur du continent africain.

Ce module a été développé dans le cadre d'un programme à un diplôme en informatique appliquée, en collaboration avec 18 institutions partenaires africaines de 16 pays. Un total de 156 modules ont été élaborés pour assurer la disponibilité ou traduit en anglais, français et portugais. Ces modules ont également été mis à disposition en tant que ressources éducatives libres (REL) sur oer.avu.org.

Au nom de l'Université virtuelle africaine et notre patron, nos institutions partenaires, la Banque africaine de développement, je vous invite à utiliser ce module dans votre établissement, pour votre propre formation, de partager le plus largement possible et à participer activement à l'avu les communautés de pratique de votre intérêt. Nous nous engageons à être en première ligne de l'élaboration et le partage de ressources éducatives libres.

L'Université virtuelle africaine (UVA) est une organisation intergouvernementale panafricaine créée par la location avec le mandat d'accroître sensiblement l'accès à un enseignement supérieur de qualité et de formation à l'aide de l'information technologies de la communication. Une Charte, l'établissement de l'avu en tant qu'organisation intergouvernementale, a été signé ce jour par dix-neuf (19) Les gouvernements africains - le Kenya, le Sénégal, la Mauritanie, le Mali, la Côte d'Ivoire, Tanzanie, Mozambique, République démocratique du Congo, Bénin, Ghana, République de Guinée, Burkina Faso, Niger, Soudan du Sud, Soudan, l'Éthiopie, la Gambie, la Guinée-Bissau et le Cap-Vert.

Les institutions suivantes ont participé au programme d'informatique appliquée : (1) Université d'Abomey Calavi au Bénin ; (2) Université de Ougagadougou au Burkina Faso ; (3) l'Université Lumière de Bujumbura au Burundi ; (4) l'Université de Douala au Cameroun ; (5) Université de Nouakchott en Mauritanie ; (6) l'Université Gaston Berger au Sénégal ; (7) Université des Sciences, des Techniques et technologies de Bamako au Mali (8) Ghana Institute of Management and Public Administration ; (9) Université des Sciences et Technologies de Kwame Nkrumah au Ghana ; (10) l'Université Kenyatta au Kenya ; (11) l'Université d'Egerton au Kenya ; (12) l'Université d'Addis Abeba en Ethiopie (13) Université du Rwanda (14) ; Université de Dar es Salaam en Tanzanie ; (15) l'Université Abdou Moumouni de Niamey au Niger ; (16) l'Université Cheikh Anta Diop de Sénégal ; (17) Universidade Pedagógica au Mozambique ; et (18) l'Université de la Gambie en Gambie.

Bakary Diallo,

Recteur de l'

Université virtuelle africaine

Crédits de production

Auteur

Dr. Cherif Diallo

Pair Réviseur

Pelagie Houngue

UVA – Coordination Académique

Dr. Marilena Cabral

Coordinateur global Sciences Informatiques Apliquées

Prof Tim Mwololo Waema

Coordinateur du module

Jules Degila

Concepteurs pédagogiques

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipe Média

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Droits d'auteur

Ce document est publié dans les conditions de la Creative Commons

Http://fr.wikipedia.org/wiki/Creative_Commons

Attribution http://creativecommons.org/licenses/by/2.5/

Le gabarit est copyright African Virtual University sous licence Creative Commons Attribution-ShareAlike 4.0 International License. CC-BY, SA

Supporté par

Projet Multinational II de l'UVA financé par la Banque africaine de développement.

Table des matières

Avant propos 2

Crédits de production 3

Droits d'auteur 4

Supporté par 4

Aperçu du cours 9

Bienvenue à la programmation de système 9

Prérequis. 9

Matériaux 9

Objectifs du cours 10

Unités 10

Évaluation 11

Plan 11

Lectures et autres ressources 12

Unité 0. Évaluation diagnostique 14

Introduction à l'unité. 14

Objectifs de l'unité 14

Activités d'apprentissage 15

Activité 1: Ouvrir un Fichier Source C ou C++ 15

Termes clés 15

Activité 2: Compiler avec GCC. 16

Compiler un Fichier Source Isolé 17

Lier les Fichiers Objet 18

Activité 3: Automatiser le Processus avec GNU Make 19

Activité 4: Déboguer avec le Débogueur GNU (GDB) 21

Évaluation 26

Résumé de l'unité 26

Lectures et autres ressources 27

Évaluation de l'unité 27

Unité 1. Librairie C et appels systèmes d'E/S 28

Introduction à l'unité. 28

Objectifs de l'unité 28

Activité 1 – Aperçu de la librairie GNU C 29

Introduction 29

Termes clés 29

Exercice – fiches de lecture 30

Exercice – fiches de lecture 30

Activité 2 : E/S de bas niveau 31

Introduction 31

Évaluation 31

Ouvrir un Fichier 32

Fermer un fichier 34

Écrire des données 34

Lecture de Données 37

Se déplacer dans un Fichier 39

Activité 3: Structure stat 41

Structure stat 41

Écriture et Lecture Vectorielles 43

Lien avec les Functions d'E/S Standards du C 44

Évaluation 45

Résumé de l'unité 47

Évaluation de l'unité 48

Instructions 48

Lectures et autres ressources 49

Unité 2. Programmation shell et Code assembleur en ligne 50

Introduction à l'unité. 50

Objectifs de l'unité 53

Activité 2: Les bases du shell 54

Premier script shell 55

Fichiers d'initialisation et variables 60

Les expressions conditionnelles 64

Activité 2: Code assembleur en ligne 72

Introduction 72

Quand Utiliser du Code Assembleur 72

Assembleur en Ligne Simple 73

Convertir un asm en Instructions Assembleur 73

Syntaxe Assembleur Avancée 74

Instructions Assembleur 74

Sorties 75

Entrées 76

Déclaration des Modifications 76

Problèmes d'Optimisation 79

Problèmes de Maintenance et de Portabilité 79

Exercices divers 80

Évaluation de l'unité 84

Instructions 84

Exercices divers 84

Lectures et autres ressources 86

Unité 3. Processus, Threads et Gestion de la Mémoire 87

Introduction à l'unité. 87

Termes clés 88

Objectifs de l'unité 88

Laboratoire #1 : Manipulation de processus et interpréteur de commandes . . . 88

Activité 1: Les Processus 89

Aperçu des Processus 89

Identifiants de Processus 89

Voir les Processus Actifs 90

Tuer un Processus 91

Créer des Processus 91

Utiliser la fonction system 91
Utiliser la Famille de exec 93
Utiliser fork et exec 94
Ordonnancement de Processus 96
Signaux 97
Fin de Processus 99
Attendre la Fin d'un Processus 101
Les Appels Système wait 101
Processus Zombies 103
Libérer les Ressources des Fils de Façon Asynchrone 104
Évaluation 106
Activité 2: Threads 107
Introduction 107
Création de Threads 107
Transmettre des Données à un Thread 109
Synchroniser des Threads 111
Valeurs de Retour des Threads 112
Plus d'Informations sur les Identifiants de Thread 114
Attributs de Thread 115
Évaluation 116
Activité 3: Gestion de la mémoire 117
Introduction 117
Mémoire Partagée 117
Communication Locale Rapide 117
Le Modèle Mémoire 117
Allocation 118
Attachement et Détachement 119
Contrôler et Libérer la Mémoire Partagée 120
Programme Exemple 120
Evaluation 122

Résumé de l'unité **123**

Évaluation de l'unité **123**

Instructions : 123

Lectures et autres ressources **125**

Laboratoire #1 : Manipulation de processus et interpréteur de commandes . . **125**

Objectif du laboratoire : 125

Première partie : Exécution d'une commande 125

Unité 4. Communication Interprocessus **129**

Introduction à l'unité. **129**

Termes clés **130**

Objectifs de l'unité **130**

Laboratoire #2 : Conception et implémentation d'un ordonnanceur de processus avec multi-niveau de priorités **130**

Laboratoire #3 : Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation **130**

Activité 1: Tubes ou pipes **131**

Tubes 131

Créer des Tubes 131

Communication entre Processus Père et Fils 132

Rediriger les Flux d'Entrée, de Sortie et d'Erreur Standards 135

Popen et pclose 137

Évaluation **138**

Activité 2: FIFO **138**

Créer une FIFO 139

Accéder à une FIFO 139

Évaluation **140**

Différences avec les Canaux Nommés Windows 140

Activité 3: Sockets **140**

Concepts relatifs aux Sockets 141

Appels Système 142

Créer et Détruire des Sockets 142
Appeler connect 143
Envoyer des Informations 143
Serveurs 143
Sockets Locaux 144
Un Exemple Utilisant les Sockets Locaux 144
Sockets Internet 149
Couples de Sockets 153
Évaluation 153
Résumé de l'unité 153
Évaluation de l'unité 154
Instructions 154
Lectures et autres ressources 155
Évaluation du cours 155
Résumé du Module 156
Laboratoire #2 : Conception et implémentation d'un ordonnanceur de processus avec multi-niveau de priorités 156
Objectif du laboratoire 156
Laboratoire #3 : Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation 168
Objectif du laboratoire 168

Aperçu du cours

Bienvenue à la programmation de système

La programmation système est un type de programmation qui vise au développement de programmes qui font partie du système d'exploitation d'un ordinateur ou qui en réalisent les fonctions. Elle se distingue de la programmation des applications en ce qu'elle s'intéresse non pas au traitement des données, mais à la résolution des problèmes pour les humains, aux interfaces, aux protocoles et à la gestion des ressources, telles que le temps et l'espace. Donc, en réalité seuls les programmes d'application sont réellement utilisés par les utilisateurs. Les programmes système le sont implicitement.

Elle inclut, en outre, l'accès aux fichiers, la programmation du clavier, de l'écran, des modems, la programmation réseau, et, en général, la programmation de tous les périphériques qui font entrer ou sortir de l'information d'un ordinateur, de la mémoire vive et des processeurs.

Ce module est composé des unités suivantes :

- Unité 0 : Évaluation diagnostique
- Unité 1 : Librairie C et appels systèmes d'E/S
- Unité 2: Programmation shell et Code assembleur en ligne
- Unité 3: Processus, Threads et Gestion de la mémoire
- Unité 4: Communications interprocessus

Prérequis

- Introduction aux systèmes d'exploitation
- Introduction à la programmation structurée
- Les bases du système d'exploitation Unix/Linux
- Programmation en langage C/C++

Matériaux

Les matériaux nécessaires pour compléter ce cours comprennent les :

- Un PC équipé d'un système d'exploitation Linux, de préférence connecté à internet;

Objectifs du cours

Préparer les étudiants à bien maîtriser le langage de programmation des systèmes et à comprendre l'organisation d'un microprocesseur de base.

À la fin de ce cours, l'étudiant devrait être en mesure de rédiger des programmes de faible niveau, de saisir des commandes de système d'exploitation et d'exécuter des programmes Unix (de bas niveau).

Unités

Unité 0: Évaluation diagnostique

Cette unité fournit une vue d'ensemble du module de programmation de systèmes. Elle passe en revue les bases de la programmation en langage C et précise l'organisation des modules, les contenus, activités et évaluations.

Unité 1: Librairie C et appels systèmes d'E/S

Cette unité fournit un aperçu de la bibliothèque C GNU, y compris les bibliothèques standards et la portabilité, les bases de l'utilisation de la bibliothèque, et les cas d'utilisation des fonctions de la bibliothèque C pour l'allocation d'espace de stockage pour les données du programme. L'unité explore en outre les fonctions de la bibliothèque GNU C pour manipuler les fichierset des répertoires. L'unité traite également le système d'entrée / sortie de fichier appelle àGNU plateforme / LINUX et met en évidence le noyau I/O appels rapport aux fonctions d'E/S de la Bibliothèque C standard.

Unité 2: Programmation shell et Code assembleur en ligne

Les programmes Shell fournis par les systèmes d'exploitation acceptent l'instruction de l'utilisateurou des commandes pour les passer au noyau, de même, les programmes écrits en niveau supérieurtels que le langage C qui fournissent également un moyen de charger lematériel pour effectuer une tâche de l'utilisateur. Cette unité explore à la fois le contexte de programmation, à savoir, la programmation shell et l'assemblage en ligne de code C.

Unité 3: Processus, Threads et Gestion de la mémoire

Les programmes doivent être mis en mémoire et placés à l'intérieur d'un processus pour être exécutés. Une instance en cours d'exécution d'un programme est appelé un processus, et les threads sont des mécanismes permettant à un programme defaire plusieurs choses à la fois. Dans cette unité, nous allons décrire les processus, les threads et les fonctions de manipulation de la mémoire dans les systèmes Linux dont la plupart des sontsimilaires à ceux d'autres systèmes UNIX.

Un lab conclut cette unité :

- Lab#1 : Manipulation de processus et interpréteur de commandes conclut cette unité.

Unité 4: Communications interprocessus

La communication interprocessus est le transfert de données entre les processus. Danscette unité, nous présentons différentes manières pour la communication entre les processus parents et fils, entre les processus «non reliés», et même entre les processus surdifférentes machines. T

rois types de communications interprocessussontdiscutés : Tubes (Pipes), FIFOs, et Sockets

Deux labs concluent cette unité :

- Lab#2 : Conception et implémentation d'un ordonnanceur de processus avec multi-niveau de priorités.
- Lab#3 : Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation

Évaluation

Les évaluations formatives (vérification de progrès) sont incluses dans chaque unité.

Les évaluations sommatives (tests et travaux finaux) sont fournies à la fin de chaque module et traitent des connaissances et compétences du module.

Les évaluations sommatives sont gérées à la discrétion de l'établissement qui offre le cours. Le plan d'évaluation proposé est le suivant:

1	Unité 1 : Evaluation de l'unité Unité 2 : Evaluation de l'unité	20%
2	Unité 3 : Evaluation de l'unité	30%
3	Unité 4 : Evaluation de l'unité	50%

Plan

Unité	Sujets et Activités	Durée estimée
Unité 0 & 1	Travaux individuels théoriques, et pratiques. Tutorats	40h
Unité 2	Travaux individuels théoriques, et pratiques. Tutorats	30h
Unité 3	Travaux individuels théoriques, et pratiques. Tutorats	25h
Unité 4	Travaux individuels théoriques, et pratiques. Tutorats	25h

Lectures et autres ressources

Les lectures et autres ressources dans ce cours sont indiquées ci-dessous.

Unité 0

Lectures et autres ressources obligatoires:

- Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001, pp3-15

Lectures et autres ressources optionnelles:

- Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love
- UNIX Systems Programming: Communication, Concurrency, and Threads. By Kay A. Robbins, Steven Robbins

Unité 1

Lectures et autres ressources obligatoires:

- The GNU C Library Reference Manual, Sandra Loosemore With Richard M. Stallman,Roland McGrath, Andrew Oram, and Ulrich Drepper for version 2.21, Copyright c1993–2014 Free Software Foundation, Inc.(http://www.gnu.org/software/libc/manual/pdf/libc.pdf)
- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel,New Riders Publishing, FIRST EDITION: June, 2001, pp 281300

Lectures et autres ressources optionnelles:

- Linux System Programming: Talking Directly to the Kernel and C Library By RobertLove
- UNIX Systems Programming: Communication, Concurrency, and Threads. By KayA. Robbins, Steven Robbins
- http://www.acm.uiuc.edu/webmonkeys/book/c_guide/: The C Library Reference Guide
- http://www.delorie.com/gnu/docs/glibc/libc_toc.html: The GNU C Library

Unité 2

Lectures et autres ressources obligatoires:

- Learning the bash Shell: Unix Shell Programming By Cameron Newham and Bill Rosenblatt,Copyright 2005, O'Reilly Media, USA.

Lectures et autres ressources optionnelles:

- Learning the bash Shell: Unix Shell Programming (In a Nutshell (O'Reilly)), Kindle Edition, Cameron Newham (Author)
- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001

Unité 3

Lectures et autres ressources obligatoires:

- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001,

Lectures et autres ressources optionnelles:

- Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love
- Linux Kernel Development, Robert Love, Pearson Education, 22 Jun 2010
- UNIX Systems Programming: Communication, Concurrency, and Threads. By Kay A. Robbins, Steven Robbins

Unité 4

Lectures et autres ressources obligatoires:

- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001,

Lectures et autres ressources optionnelles:

- UNIX Systems Programming: Communication, Concurrency, and Threads, By Kay A. Robbins, Steven Robbins, Prentice Hall Professional, 2003
- Interprocess Communications in Linux, By John Shapley Gray, Prentice Hall Professional, 2003
- UNIX Network Programming: Interprocess communications, Volume 2 , W. Richard Stevens Prentice Hall PTR, 1999
- Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love

Unité 0. Évaluation diagnostique

Introduction à l'unité

Cette unité vous permettra de vérifier les connaissances que vous devez avoir avant de commencer le cours. Vous pouvez faire l'évaluation de l'unité avant de faire des activités d'apprentissage pour aider à rafraîchir vos connaissances.

Cette unité permet d'évaluer si les apprenants ont été initiés aux principes fondamentaux de la conception des systèmes d'exploitation, et d'avoir un aperçu sur leur expérience pratique de l'installation et de la maintenance des systèmes d'exploitation.

Les systèmes d'exploitation sont essentiels aux activités informatiques. Un système d'exploitation est un programme qui agit comme intermédiaire entre l'utilisateur d'un ordinateur et le matériel informatique. Deux des principaux buts d'un système d'exploitation consistent à gérer les ressources (temps machine, mémoire, etc.) et de contrôler les utilisateurs et les logiciels.

L'apprenant doit savoir les principaux concepts qui sous-tendent les systèmes d'exploitation, et être capable d'examiner les voies et moyens de réaliser les objectifs en matière de conception.

Cette unité présente les étapes de base nécessaires à la création d'un programme Linux en C ou C++. En particulier, il explique comment créer et modifier un code source C ou C++, compiler ce code et déboguer le résultat.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Créer et modifier des fichiers sources C/C++
- Compiler des programmes C/C++ avec GCC
- Lier les fichiers objet
- Automatiser la compilation et l'édition des liens avec le GNU Make
- Faire le debug de programmes C/C++ avec GNU Debugger.

Termes clés

Compilateur: Traduit le programme écrit en langage humain en un programme en langage machine pouvant être exécuté

Edition de liens (Linke): Un programme qui prend plusieurs fichiers générés par un compilateur pour produire un seul fichier (exécutable, bibliothèque ou un autre fichier objet)

Debugger: Le programme utilisé pour comprendre pourquoi le programme ne fonctionne pas comme prévu.

GCC: GNU Compiler Collection.

GNU: GNU's Not Unix!

Activités d'apprentissage

Activité 1: Ouvrir un Fichier Source C ou C++

Vous pouvez lancer Emacs en saisissant emacs suivi de la touche Entrée dans votre terminal. Lorsque Emacs a démarré, vous pouvez utiliser les menus situés dans la partie supérieure pour créer un nouveau fichier source. Cliquez sur le menu File, sélectionnez Open File puis saisissez le nom du fichier que vous voulez ouvrir dans le minibuffer au bas de l'écran1. Si vous voulez créer un fichier source C, utilisez un nom de fichier se terminant par .c ou .h. Si vous désirez créer un fichier C++, utilisez un nom de fichier se terminant par .cpp, .hpp, .cxx, .hxx, .C ou .H. Lorsque le fichier est ouvert, vous pouvez taper comme vous le feriez dans un programme de traitement de texte. Pour sauvegarder le fichier, sélectionnez l'entrée Save Buffer dans le menu File. Lorsque vous avez terminé d'utiliser Emacs, vous pouvez choisir l'option Exit Emacs dans le menu File.

Si vous n'aimez pas cliquer, vous pouvez utiliser les raccourcis clavier pour ouvrir ou fermer un fichier et sortir d'Emacs. Pour ouvrir un fichier, saisissez C-x C-f (C-x signifie de maintenir la touche Control enfoncée tout en appuyant sur la touche x). Pour sauvegarder un fichier, saisissez C-x C-s. Pour sortir d'Emacs, saisissez simplement C-x C-c. Si vous voulez devenir un peu plus familier avec Emacs, sélectionnez l'entrée Emacs Tutorial dans le menu Help. Le didacticiel vous propose quantité d'astuces sur l'utilisation efficace d'Emacs.

Activité 2: Compiler avec GCC

Un compilateur transforme un code source lisible par un humain en code objet lisible par la machine qui peut être exécuté. Les compilateurs de choix disponibles sur les systèmes Linux font tous partie de la GNU Compiler Collection, plus communément appelée GCC3. GCC inclut également des compilateurs C, C++, Java, Objective-C, Fortran et Chill. Cette unité se concentre plus particulièrement sur la programmation C et C++.

Supposons que vous ayez un projet comme celui du Listing 1.2 avec un fichier source C++ (reciprocal.cpp) et un fichier source C (main.c) comme dans le Listing 1.1. Ces deux fichierssont supposés être compilés puis liés entre eux pour produire un programme appeléreciprocal4.Ce programme calcule l'inverse d'un entier.

Listing 1 – (main.c) – Fichier source C

```
1   # include <stdio .h>
2    # include <stdlib .h>
3    # include " reciprocal . hpp "
4
5   int main ( int argc , char ** argv )
6    {
7          int i;
8
9          i = atoi ( argv [1]) ;
10         printf ("L'inverse de %d est %g\n", i, reciprocal (i));
11         return 0;
12   }
```

Listing 2 – (reciprocal.cpp) – Fichier source C++

```
1    # include <cassert >
2    # include " reciprocal . hpp "
3
4    double reciprocal ( int i) {
5    // i doit être différent de zéro
6    assert (i != 0);
7    return 1.0/ i;
```

```
8    }
```

Il y a également un fichier d'entête appelé reciprocal.hpp (voir Listing 1.3).

Listing 3 – (reciprocal.hpp) – Fichier d'entête

```
1    # ifdef __cplusplus
2    extern "C" {
3    # endif
4
5    extern double reciprocal ( int i);
6
7    # ifdef __cplusplus
8    }
9    # endif
```

La première étape est de traduire le code C et C++ en code objet.

Compiler un Fichier Source Isolé

Le nom du compilateur C est gcc. Pour compiler un fichier source C, utilisez l'option -c.

Donc par exemple, cette commande compile le fichier source main.c :

% gcc -c main.c

Le fichier objet résultant est appelé main.o. Le compilateur C++ s'appelle g++. Son mode opératoire est très similaire à gcc ; la compilation de reciprocal.cpp s'effectue via la commande suivante :

% g++ -c reciprocal.cpp

L'option -c indique à g++ de ne compiler le fichier que sous forme d'un fichier objet ; sans cela,

g++ tenterait de lier le programme afin de produire un exécutable. Une fois cette commande saisie, vous obtenez un fichier objet appelé reciprocal.o.

Vous aurez probablement besoin de quelques autres options pour compiler un programme d'une taille raisonnable. L'option -I est utilisée pour indiquer à GCC où rechercher les fichiers d'entête. Par défaut, GCC cherche dans le répertoire courant et dans les répertoires où les entêtes des bibliothèques standards sont installés. Si vous avez besoin d'inclure des fichiers d'entête situés à un autre endroit, vous aurez besoin de l'option -I. Par exemple, supposons que votre projet ait un répertoire appelé src, pour les fichiers source, et un autre appelé include. Vous compileriez reciprocal.cpp comme ceci pour indiquer à g++ qu'il doit utiliser en plus le répertoire include pour trouver reciprocal.hpp :

% g++ -c -I ../include reciprocal.cpp

Parfois, vous pourriez vouloir définir des macros au niveau de la ligne de commande. Par exemple, dans du code de production, vous ne voudriez pas du surcoût de l'assertion présente dans reciprocal.cpp ; elle n'est là que pour vous aider à déboguer votre programme. Vous désactivez la vérification en définissant la macro NDEBUG. Vous pourriez ajouter un #define explicite dans reciprocal.cpp, mais cela nécessiterait de modifier la source elle-même. Il est plus simple de définir NDEBUG sur la ligne de commande, comme ceci :

% g++ -c -D NDEBUG reciprocal.cpp

Si vous aviez voulu donner une valeur particulière à NDEBUG, vous auriez pu saisir quelque chose de ce genre :

% g++ -c -D NDEBUG=3 reciprocal.cpp

Si vous étiez réellement en train de compiler du code de production, vous voudriez probablement que GCC optimise le code afin qu'il s'exécute aussi rapidement que possible. Vous pouvez le faire en utilisant l'option en ligne de commande -O2 (GCC a plusieurs niveaux d'optimisation ; le second niveau convient pour la plupart des programmes). Par exemple, ce qui suit compile reciprocal.cpp avec les optimisations activées :

% g++ -c -O2 reciprocal.cpp

Notez que le fait de compiler avec les optimisations peut rendre votre programme plus difficile à déboguer avec un débogueur (voyez la Section 1.4, Déboguer avec le Débogueur GNU (GDB)).

De plus, dans certaines circonstances, compiler avec les optimisations peut révéler des bogues qui n'apparaissaient pas auparavant.

Vous pouvez passer un certain nombre d'autres options à gcc et g++. Le meilleur moyen d'en obtenir une liste complète est de consulter la documentation en ligne. Vous pouvez le faire en saisissant ceci à l'invite de commandes :

Lier les Fichiers Objet

Maintenant que vous avez compilé main.c et reciprocal.cpp, vous devez les lier. Vous devriez toujours utiliser g++ pour lier un programme qui contient du code C++, même s'il contient également du code C. Si votre programme ne contient que du code C, vous devriez utiliser gcc à la place. Comme ce programme contient à la fois du code C et du code C++, vous devriez utiliser g++, comme ceci :

% g++ -o reciprocal main.o reciprocal.o

L'option -o donne le nom du fichier à générer à l'issue de l'étape d'édition de liens. Vous pouvezmaintenant lancer reciprocal comme ceci :

% ./reciprocal 7

L'inverse de 7 est 0.142857

Comme vous pouvez le voir, g++ a automatiquement inclus les bibliothèques d'exécutionC standards contenant l'implémentation de printf. Si vous aviez eu besoin de lier une autrebibliothèque (comme un kit de développement d'interfaces utilisateur), vous auriez indiqué labibliothèque avec l'option -l. Sous Linux, les noms des bibliothèques commencent quasiment toujourspar lib. Par exemple, la bibliothèque du Module d'Authentification Enfichable(PluggableAuthentication Module, PAM) est appelée libpam.a. Pour inclure libpam.a lors del'édition deliens, vous utiliserez une commande de ce type :

% g++ -o reciprocal main.o reciprocal.o -lpam

Le compilateur ajoutera automatiquement le préfixe lib et le suffixe .a.

Comme avec les fichiers d'entête, l'éditeur de liens recherche les bibliothèques dans certainsemplacements standards, ce qui inclut les répertoires /lib et /usr/lib qui contiennent lesbibliothèques système standards. Si vous voulez que l'éditeur de liens cherche en plus dansd'autres répertoires, vous devez utiliser l'option -L, qui est l'équivalent de l'option -l dont nousavons parlé plus tôt. Vous pouvez utiliser cette commande pour indiquer à l'éditeur de liensde rechercher les bibliothèques dans le répertoire /usr/local/lib/pam avant de les rechercherdans les emplacements habituels :

% g++ -o reciprocal main.o reciprocal.o -L/usr/local/lib/pam -lpam

Bien que vous n'ayez pas à utiliser l'option -I pour que le préprocesseur effectue ses recherchesdans le répertoire courant, vous devez utiliser l'option -L pour que l'éditeur de liens le fasse.

Par exemple, vous devrez utiliser ce qui suit pour indiquer à l'éditeur de liens de rechercher labibliothèque test dans le répertoire courant :

% gcc -o app app.o -L. -ltest

Activité 3: Automatiser le Processus avec GNU Make

Si vous êtes habitué à la programmation pour le système d'exploitation Windows, vous avez probablement l'habitude de travailler avec un Environnement de Développement Intégré (IDE).

Vous ajoutez les fichiers à votre projet puis l'IDE compile ce projet automatiquement. Bien que des IDE soient disponibles pour Linux, ce livre n'en traite pas. Au lieu de cela, il vous montre comment vous servir de GNU Make pour recompiler votre code automatiquement, comme le font en fait la majorité des programmeurs Linux.

L'idée de base derrière make est simple. Vous indiquez à make quelles cibles vous désirez compiler puis donnez des règles expliquant comment les compiler. Vous pouvez également spécifier des dépendances qui indiquent quand une cible particulière doit être recompilée.

Dans notre projet exemple reciprocal, il y a trois cibles évidentes : reciprocal.o, main.o et reciprocal lui-même. Vous avez déjà à l'esprit les règles nécessaires à la compilation de ces cibles sous forme des lignes de commande données précédemment. Les dépendances nécessitent un minimum de réflexion. Il est clair que reciprocal dépend de reciprocal.

O et main.o car vous ne pouvez pas passer à l'étape d'édition de liens avant d'avoir compilé chacun des fichiers objets. Les fichiers objets doivent être recompilés à chaque fois que le fichier source correspondant est modifié. Il y a encore une subtilité : une modification de reciprocal.hpp doit entraîner la recompilation des deux fichiers objets car les deux fichiers source incluent ce fichier d'entête.

En plus des cibles évidentes, il devrait toujours y avoir une cible clean. Cette cible supprime tous les fichiers objets générés afin de pouvoir recommencer sur des bases saines. La règle pour cette cible utilise la commande rm pour supprimer les fichiers.

Vous pouvez fournir toutes ces informations à make en les plaçant dans un fichier nomméMakefile. Voici ce qu'il contient :

```
reciprocal : main .o reciprocal .o

g++ $( CFLAGS ) -o reciprocal main .o reciprocal .o

main .o: main .c reciprocal . hpp

gcc $( CFLAGS ) -c main .c

reciprocal .o: reciprocal . cpp reciprocal . hpp

g++ $( CFLAGS ) -c reciprocal . cpp

clean :

rm -f *.o reciprocal
```

Vous pouvez voir que les cibles sont listées sur la gauche, suivies de deux-points puis des dépendances. La règle pour la construction d'une cible est placée sur la ligne suivante (ignorez le $(CFLAGS) pour l'instant). La ligne décrivant la règle doit commencer par un caractère de tabulation ou make ne s'y retrouvera pas. Si vous éditez votre Makefile dans Emacs, Emacs vous assistera dans le formatage.

Si vous supprimez les fichiers objets que vous avez déjà créé et que vous tapez simplement :

% make

sur la ligne de commande, vous obtiendrez la sortie suivante :

```
% make

gcc -c main.c

g++ -c reciprocal.cpp

g++ -o reciprocal main.o reciprocal.o
```

Vous constatez que make a automatiquement compilé les fichiers objet puis les a liés. Si vous modifiez maintenant main.c d'une façon quelconque puis saisissez make de nouveau, vous obtiendrez la sortie suivante :

```
% make

gcc -c main.c

g++ -o reciprocal main.o reciprocal.o
```

Vous constatez que make recompile main.o et réédite les liens, il ne recompile pasreciprocal.cppcar aucune des dépendances de reciprocal.o n'a changé.

$(CFLAGS) est une variable de make. Vous pouvez définir cette variable soit dans le Makefile lui-même soit sur la ligne de commande. GNU make substituera la variable par sa valeur lorsqu'il exécutera la règle. Donc, par exemple, pour recompiler avec les optimisations activées, vous procéderiez de la façon suivante :

```
% make clean

rm -f *.o reciprocal

% make CFLAGS=-O2

gcc -O2 -c main.c

g++ -O2 -c reciprocal.cpp

g++ -O2 -o reciprocal main.o reciprocal.o
```

Notez que le drapeau -O2 a été inséré à la place de $(CFLAGS) dans les règles.

Dans cette section, nous n'avons présenté que les capacités les plus basiques de make. Vous pourrez en apprendre plus grâce à la commande suivante :

% info make

Vous trouverez des informations sur la façon de rendre un Makefile plus simple à maintenir, comment réduire le nombre de règles à écrire et comment calculer automatiquement les dépendances.

Activité 4: Déboguer avec le Débogueur GNU (GDB)

Le débogueur est le programme que vous utilisez pour trouver pourquoi votre programme ne se comporte pas comme vous pensez qu'il le devrait. Vous y aurez souvent recours5. Le débogueur GNU (GNU debugger, GDB) est le débogueur utilisé par la plupart des programmeurs Linux.

Vous pouvez utiliser GDB pour exécuter votre code pas à pas, poser des points d'arrêt et examiner les valeurs des variables locales.

Pour utiliser GDB, vous devez compiler en activant les informations de débogage. Pour cela, ajoutez l'option -g sur la ligne de commande de compilation. Si vous utilisez un Makefile comme nous l'avons expliqué plus haut, vous pouvez vous contenter de positionner CFLAGS à –g lors de l'exécution de make, comme ceci :

```
% make CFLAGS=-g

gcc -g -c main.c

g++ -g -c reciprocal.cpp

g++ -g -o reciprocal main.o reciprocal.o
```

Lorsque vous compilez avec -g, le compilateur inclut des informations supplémentaires dans les fichiers objets et les exécutables. Le débogueur utilise ces informations pour savoir à quelle adresse correspond à quelle ligne et dans quel fichier source, afficher les valeurs des variables, etc.

Vous pouvez démarrer gdb en saisissant :

```
% gdb reciprocal

Lorsque GDB démarre, il affiche l'invite :

(gdb)
```

La première étape est de lancer votre programme au sein du débogueur. Entrez simplement la commande run et les arguments du programme. Essayez de lancer le programme sans aucun argument, comme ceci :

```
(gdb) run

Starting program: reciprocal

Program received signal SIGSEGV, Segmentation fault.

 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287

287 strtol.c: No such file or directory.

(gdb)
```

Le problème est qu'il n'y a aucun code de contrôle d'erreur dans main. Le programme attend un argument, mais dans ce cas, il n'en a reçu aucun. Le message SIGSEGV indique un plantage du programme. GDB sait que le plantage a eu lieu dans une fonction appelée __strtol_internal.

Cette fonction fait partie de la bibliothèque standard, et les sources ne sont pas installées ce qui explique le message « No such file or directory » (Fichier ou répertoire inexistant). Vous pouvez observer la pile en utilisant la commande where :

```
(gdb) where

#0 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287

#1 0x40096fb6 in atoi (nptr=0x0) at ../stdlib/stdlib.h:251

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8
```

Vous pouvez voir d'après cet extrait que main a appelé la fonction atoi avec un pointeurNULL ce qui est la source de l'erreur.Vous pouvez remonter de deux niveaux dans la pile jusqu'à atteindre main en utilisant la commande up :

```
(gdb) up 2

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8

8   i = atoi (argv[1]);
```

Notez que GDB est capable de trouver le fichier source main.c et qu'il affiche la ligne contenant l'appel de fonction erroné. Vous pouvez inspecter la valeur des variables en utilisant la commande print :

```
(gdb) print argv[1]

$2 = 0x0
```

Cela confirme que le problème vient d'un pointeur NULL passé à atoi.

Vous pouvez placer un point d'arrêt en utilisant la commande break :

```
(gdb) break main

Breakpoint 1 at 0x804862e: file main.c, line 8.
```

Cette commande place un point d'arrêt sur la première ligne de main6. Essayez maintenant de relancer le programme avec un argument, comme ceci :

```
(gdb) run 7

Starting program: reciprocal 7
```

Breakpoint 1, main (argc=2, argv=0xbffff5e4) at main.c:8

8 i = atoi (argv[1])

Vous remarquez que le débogueur s'est arrêté au niveau du point d'arrêt.Vous pouvez passer à l'instruction se trouvant après l'appel à atoi en utilisant la commande next :

gdb) next

9 printf ("L'inverse de %d est %g\n", i, reciprocal (i));

Si vous voulez voir ce qui se passe à l'intérieur de la fonction reciprocal, utilisez la commande step, comme ceci :

(gdb) step

reciprocal (i=7) at reciprocal.cpp:6

6 assert (i != 0);

Vous êtes maintenant au sein de la fonction reciprocal.

Vous pouvez trouver plus commode d'exécuter gdb au sein d'Emacs plutôt que de le lancer directement depuis la ligne de commande. Utilisez la commande M-x gdb pour démarrer gdb dans une fenêtre Emacs. Si vous stoppez au niveau d'un point d'arrêt, Emacs ouvre automatiquement le fichier source approprié. Il est plus facile de se rendre compte de ce qui se passe en visualisant le fichier dans son ensemble plutôt qu'une seule ligne.

Obtenir Plus d'Informations

Quasiment toutes les distributions Linux disposent d'une masse importante de documentation.

Vous pourriez apprendre la plupart des choses dont nous allons parler dans cette unité simplement en lisant la documentation de votre distribution Linux (bien que cela vous prendraitprobablement plus de temps). La documentation n'est cependant pas toujours très bienorganisée,donc la partie la plus subtile est de trouver ce dont vous avez besoin. La documentationdate quelquefois un peu, aussi, prenez tout ce que vous y trouvez avec un certain recul. Si lesystème ne se comporte pas comme le dit une page de manuel, c'est peut-être parce que celle-ciest obsolète.

Pour vous aider à naviguer, voici les sources d'information les plus utiles sur la programmation avancée sous Linux.

Pages de Manuel

Les distributions Linux incluent des pages de manuel pour les commandes les plus courantes, les appels système et les fonctions de la bibliothèque standard. Les pages de manuel sont divisées en sections numérotées ; pour les programmeurs, les plus importantes sont celles-ci :

(1) Commandes utilisateur

(2) Appels système

(3) Fonctions de la bibliothèque standard

(8) Commandes système/d'administration

Les numéros indiquent les sections des pages de manuel. Les pages de manuel de Linux sont installées sur votre système ; utilisez la commande man pour y accéder. Pour accéder à une page de manuel, invoquez simplement man nom, où nom est un nom de commande ou de fonction. Dans un petit nombre de cas, le même nom apparaît dans plusieurs sections ; vous pouvez indiquer explicitement la section en plaçant son numéro devant le nom. Par exemple, si vous saisissez la commande suivante, vous obtiendrez la page de manuel pour la commande sleep (dans la section 1 des pages de manuel Linux) :

% man sleep

Pour visualiser la page de manuel de la fonction sleep de la bibliothèque standard, utilisez cette commande :

% man 3 sleep

Chaque page de manuel comprend un résumé sur une ligne de la commande ou fonction.

La commande whatis nom liste toutes les pages de manuel (de toutes les sections) pour une commande ou une fonction correspondant à nom. Si vous n'êtes pas sûr de la commande ou fonction à utiliser, vous pouvez effectuer une recherche par mot-clé sur les résumés via man –k mot-clé.

Les pages de manuel contiennent des informations très utiles et devraient être le premier endroit vers lequel vous orientez vos recherches. La page de manuel d'une commande décrit ses options en ligne de commande et leurs arguments, ses entrées et sorties, ses codes d'erreur, sa configuration et ce qu'elle fait. La page de manuel d'un appel système ou d'une fonction de bibliothèque décrit les paramètres et valeurs de retour, liste les codes d'erreur et les effets de bord et spécifie le fichier d'entête à inclure si vous utilisez la fonction.

Info

Le système de documentation Info contient des informations plus détaillées pour beaucoup de composants fondamentaux du système GNU/Linux et quelques autres programmes. Les pages

Info sont des documents hypertextes, similaires aux pages Web. Pour lancer le navigateur texte Info, tapez simplement info à l'invite de commande. Vous obtiendrez un menu avec les documents Info présents sur votre système (appuyez sur Ctrl+H pour afficher les touches permettant de naviguer au sein d'un document Info).

Parmi les documents Info les plus utiles, on trouve :

```
gcc Le compilateur gcc

libc La bibliothèque C GNU, avec beaucoup d'appels système

gdb Le débogueur GNU

emacs L'éditeur de texte Emacs

info Le système Info lui-même
```

Presque tous les outils de programmation standards sous Linux (y compris ld, l'éditeur de liens ; as, l'assembleur et gprof, le profiler) sont accompagnés de pages Info très utiles. Vous pouvez accéder directement à un document Info en particulier en indiquant le nom de la page sur la ligne de commandes :

% info libc

Si vous programmez la plupart du temps sous Emacs, vous pouvez accéder au navigateur Info intégré en appuyant sur M-x info ou C-h i.

Fichiers d'entête

Vous pouvez en apprendre beaucoup sur les fonctions système disponibles et comment les utiliser en observant les fichiers d'entête. Ils sont placés dans /usr/include et /usr/include/sys.

Si vous obtenez des erreurs de compilation lors de l'utilisation d'un appel système, par exemple, regardez le fichier d'entête correspondant pour vérifier que la signature de la fonction est la même que celle présentée sur la page de manuel.

Sur les systèmes Linux, beaucoup de détails obscurs sur le fonctionnement des appels systèmes apparaissent dans les fichiers d'entête situés dans les répertoires /usr/include/bits,

/usr/include/asm et /usr/include/linux. Ainsi, le fichier /usr/include/bits/signum.h définit les valeurs numériques des signaux. Ces fichiers d'entête constituent une bonne lecture pour les esprits curieux. Ne les incluez pas directement dans vos programmes, cependant ; utilisez toujours les fichiers d'entête de /usr/include ou ceux mentionnés dans la page de manuel de la fonction que vous utilisez.

Code Source

Nous sommes dans l'Open Source, non ? Le juge en dernier ressort de la façon dont doit fonctionner le système est le code source, et par chance pour les programmeurs Linux, ce code source est disponible librement. Il y a des chances pour que votre système Linux comprenne tout le code source du système et des programmes fournis ; si ce n'est pas le cas, vous avez le droit, selon les termes de la Licence Publique Générale GNU, de les demander au distributeur (le code source n'est toutefois pas forcément installé. Consultez la documentation de votre distribution pour savoir comment l'installer).

Le code source du noyau Linux lui-même est habituellement stocké sous /usr/src/linux.
Si ce livre vous laisse sur votre faim concernant les détails sur le fonctionnement des processus, de la mémoire partagée et des périphériques système, vous pouvez toujours apprendre directement à partir du code. La plupart des fonctions décrites dans ce livre sont implémentées dans la bibliothèque C GNU; consultez la documentation de votre distribution pour connaîtrel'emplacement du code source de la bibliothèque C.

Conclusion

Dans cette unité nous avons présenté les bases de la programmation en langage C/C++.

Évaluation

1. Pratiquer les exemplesde codes fournis dans cette unité.

Résumé de l'unité

Cette unité a présenté les fondamentaux de la programmation en C/C++ avec la manipulation des fichiers sources, des principes de compilation, et d'éditions de liens ainsi que les techniques de debugging.

Évaluation de l'unité

Vérifiez votre compréhension!

Instructions

Cette évaluation peut être faite en travail de groupe (par 2)

Lisez attentivement le cours et répondez aux questions suivantes

Pour la question 2, choisir deux distributions linux différentes et faire le même travail pour les deux distributions en comparant.

1. Qu'est-ce qu'un Makefile, quel est son rôle ?
2. Inspecter et analyser le code source dans le filesystem Linux /usr/src/linux

Lectures et autres ressources

Mark Mitchell, Jeffrey Oldham, and Alex Samue Advanced Linux Programming Copyright © 2001 by New Riders Publishing FIRST EDITION: June, 2001

Unité 1. Librairie C et appels systèmes d'E/S

Introduction à l'unité

La bibliothèque standard C fournit des macros, des définitions de types et de fonctions pour des tâches comme la manipulation des chaînes de caractères, des calculs mathématiques, du traitement d'entrée / sortie, l'allocation de la mémoire et plusieursautres services du système d'exploitation. Cette unité fera usage de la bibliothèque GNU C, et nous supposons quevous êtes au moins un peu familier avec le langage de programmation C et les concepts de basede la programmation.

La bibliothèque GNU C comprend plusieurs fichiers d'en-tête, chacun fournissant des définitions etdéclarations pour un groupe d'utilisations connexes; ces informations sont utilisées par le compilateur C lorsquele traitement de votre programme l'exige. Par exemple, le fichier d'en-tête `stdio.h'déclare des initialisations pour effectuer des entrées/ sorties, et le fichier d'en-tête `string.h' regroupe un certain nombre d'utilitaires de traitement de chaînes de caractères.

Il y a beaucoup de fonctions de la bibliothèque GNU C et il est irréaliste d'espérer que vous serezcapable de se rappeler exactement comment utiliser chacun d'entre eux. Il est plus important de devenirgénéralement familiers avec les types d'installations que la bibliothèque fournit, de sorte que lorsque vous écrivez vos programmes, vous pouvez reconnaître quand il faut faire usage des fonctions de la bibliothèque. Le but de cette unité est de vous indiquer comment utiliser les routines de la bibliothèque GNU. Plus précisément, l'unité permet de discuter du dossier couramment utilisépour les fonctions d'E/S, les appels systèmes et la manipulation des fichiers, des répertoires et des liens.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Utiliser les facultés offertes par la bibliothèque C dans des codes en langage C
- Utiliser les fonctions d'E/S et les appels Systèmes
- Manipuler les fichiers, répertoires et les liens dans des codes langage C.

Termes clés

ISO: International Standard Organization

GCC: GNU Compiler Collection.

GNU: GNU's Not Unix!

POSIX: Potable Operating System Interface Extension

Bibliothèque C: Collection de routines ou de programme pré compilés disponiblespour une réutilisation dans un programme ordinaire de C

Répertoire: Un dossier du système d'exploitation contenant plusieurs fichiers

Lien (symbolique): Un type particulier de fichier qui contient une référence à un autre fichier ou un répertoire sous la forme d'un chemin absolu ourelatif et qui affecte la résolution des noms de fichiers.

Fichier: Unité logique de stockage de données dans un système d'exploitation

Entrée/Sortie (E/S ou I/O): Se réfère à l'activité de lecture (entrée) et d'écriture (sortie) de données

Appels systèmes: une demande pour le système d'exploitation à faire quelque chose au nom du programme de l'utilisateur

Activités d'apprentissage

Activité 1 – Aperçu de la librairie GNU C

Introduction

Le langage C ne fournit pas de fonctionnalités innés permettant de réaliser des opérations communes d'entrée / sortie, de gestion de la mémoire, de manipulation de chaînes, et autres. Au lieu de cela, ces fonctions sont définies dans une bibliothèque standard, que vous devez compiler et lier avec vos programmes.

La bibliothèque GNU C, décrit dans cette unité, définit toutes les fonctions de la bibliothèque qui sont spécifiées par la norme ISO C, ainsi que des fonctionnalités supplémentaires spécifiques à POSIX et d'autres dérivés du système d'exploitation Unix, et des extensions spécifiques au système GNU.

Exercice – fiches de lecture

Cette activité est une activité de lecture où l'apprenant doit aller lire le lien http://www.delorie.com/gnu/docs/glibc/libc_toc.html et faire une description sommaire des concepts suivants :

1. ISO C - International standard for the C programming
2. POSIX (Portable Operating System Interface)
3. ISO/IEC 9945 (aka IEEE 1003) standards for operating systems
4. Berkeley Unix BSD and SunOS
5. SVID (System V Interface Description)
6. XPG (The X/Open Portability Guide)

Exercice – fiches de lecture

Cette activité est également une activité d'apprentissage par la lecture où l'apprenant doit aller lire le lien http://www.delorie.com/gnu/docs/glibc/libc_toc.html et faire une description sommaire sur l'utilisation de la bibliothèque GNU C quant aux concepts ci-dessous :

1. Fichier d'entêtes
2. Macros de définition de fonctions

    _POSIX_SOURCE

    _POSIX_C_SOURCE

    _BSD_SOURCE

    _SVID_SOURCE

    _XOPEN_SOURCE

    _XOPEN_SOURCE_EXTENDED

    _LARGEFILE_SOURCE

    _LARGEFILE64_SOURCE

    _FILE_OFFSET_BITS

    _ISOC99_SOURCE

    _GNU_SOURCE

    _REENTRANT

    _THREAD_SAFE

3. Noms réservés
4. la bibliothèque pour l'allocation de stockage des données du programme
5. Allocation dynamique de mémoire
6. Libération de la mémoire avec malloc

Évaluation

1. 1.Si plus d'espace n'est disponible, malloc renvoie un pointeur NULL. Vous devriez vérifier la valeur de chaque appel à malloc. Au lieu de cela, il est utile d'écrire un sous-programme qui appelle malloc et signale une erreur si la valeur est un pointeur NULL, retourne uniquement si la valeur est différente de zéro. Il est convenu d'appeler cette fonction xmalloc. Ecrire et étudier ce programme.
2. 2.Démontrer l'utilisation de malloc en écrivant un programme complet de C basée sur la fonction qui permet de copier une séquence de caractères dans une chaîne nouvellement allouée.
3. 3.Écrire un sous-programme, appelé realloc, qui prend le soin de gérer des messages d›erreur à l'instar de xmalloc pour malloc

Activité 2 : E/S de bas niveau

Introduction

Les programmeurs C sous GNU/Linux ont deux jeux de fonctions d'entrées/sortiesà leur disposition. La bibliothèque C standard fournit des fonctions d'E/S : printf, fopen, etc.Le noyau Linux fournit un autre ensemble d'opérations d'E/S qui opèrent à un niveau inférieurà celui des fonctions de la bibliothèque C.

Comme cetteunité est destinée à des gens connaissant déjà le langage C, nous supposerons quevous avez déjà rencontré et savez comment utiliser les fonctions d'E/S de la bibliothèque C.

Il y a souvent de bonnes raisons d'utiliser les fonctions d'E/S de bas niveau de Linux.
Elles sont pour la plupart des appels systèmes au noyau2 et fournissent un accès direct auxpossibilités sous-jacentes offertes par le système aux applications. En fait, les fonctions d'E/S dela bibliothèque C standard sont implantées au-dessus des appels systèmes d'E/S de bas niveaude Linux. L'utilisation de cette dernière est généralement la façon la plus efficace d'effectuer desopérations d'entrée/sortie – et est parfois également plus pratique.

Lire et Écrire des Données

La première fonction d'E/S que vous avez rencontré lorsque vous avez commencé à apprendre le langage c'était certainement printf. Elle formate une chaîne de texte puis l'affiche sur la sortiestandard. La version générique, fprintf, peut afficher le texte sur un autre flux que la sortiestandard. Un flux est représenté par un pointeur FILE*. Vous obtenez un tel pointeur en ouvrantun fichier avec fopen. Lorsque vous en avez fini, vous pouvez le fermer avec fclose. En plus defprintf, vous pouvez utiliser d'autres fonctions comme fputc, fputs ou fwrite pour écrire desdonnées dans un flux, ou fscanf, fgetc, fgets ou fread pour lire des données.

Avec les opérations d'E/S de bas niveau de Linux, vous utilisez descripteur de fichier aulieu d'un pointeur FILE*. Un descripteur de fichier est un entier qui fait référence à une instancedonnée d'un fichier ouvert au sein d'un processus. Il peut être ouvert en lecture, en écriture ou enlecture/écriture. Un descripteur de fichier ne fait pas forcément référence à un fichier ouvert ; ilpeut représenter une connexion vers un composant d'un autre système qui est capable d'envoyerou de recevoir des données.Incluez les fichiers d'en-tête <fcntl.h>, <sys/types.h>, <sys/stat.h> et <unistd.h> si vousutilisez l'une des fonctions d'E/S de bas niveau décrites ici.

Ouvrir un Fichier

Pour ouvrir un fichier et obtenir un descripteur de fichier pouvant y accéder, utilisez la commande d'appelopen. Il prend le chemin du fichier à ouvrir sous forme d'une chaîne de caractères et desindicateurs spécifiant comment il doit l'être. Vous pouvez utiliser open pour créer un nouveaufichier ; pour cela, passez un troisième argument décrivant les droits d'accès à appliquer aunouveau fichier.

Si le second argument est O_RDONLY, le fichier est ouvert en lecture seul ; une erreur serasignalée si vous essayez d'y écrire. De même, O_WRONLY ouvre le descripteur de fichier en écritureseule. Passer O_RDWR crée un descripteur de fichier pouvant être utilisé à la fois en lecture eten écriture. Notez que tous les fichiers ne peuvent pas être ouverts dans les trois modes. Parexemple, les permissions d'un fichier peuvent interdire à un processus de l'ouvrir en lecture ouen écriture ; un fichier sur un périphérique en lecture seule, comme un lecteur CD-ROM ne peutpas être ouvert en lecture.

Vous pouvez passer des options supplémentaires en utilisant un OU binaire de ces valeursavec d'autres indicateurs. Voici les valeurs les plus courantes :

Passez O_TRUNC pour tronquer le fichier ouvert, s'il existait auparavant. Les données écrites remplaceront le contenu du fichier ;

Passez O_APPEND pour ajouter les données au contenu d'un fichier existant. Elles sont écrites à la fin du fichier ;

Passez O_CREAT pour créer un nouveau fichier. Si le nom de fichier que vous passez à open correspond à un fichier inexistant, un nouveau fichier sera créé, si tant est que le répertoire le contenant existe et que le processus a les permissions nécessaires pour créer des fichiers dans ce répertoire. Si le fichier existe déjà, il est ouvert ;

Passez O_EXCL et O_CREATE pour forcer la création d'un nouveau fichier. Si le fichier existe déjà, l'appel à open échouera.

Si vous appelez open en lui passant O_CREATE, fournissez un troisième argument indiquant les permissions applicable au nouveau fichier.

Par exemple, le programme du Listing B.1 crée un nouveau fichier avec le nom de fichierindiqué sur la ligne de commande. Il utilise l'indicateur O_EXCL avec open afin de signaler uneerreur si le fichier existe déjà. Le nouveau fichier dispose des autorisations en lecture/écriturepour l'utilisateur et le groupe propriétaires et est en lecture seule pour les autres utilisateurs (sivotre umask est positionné à une valeur différente de zéro, les permissions effectives pourraientêtre plus restrictives).

Listing B.1 – (create-file.c) – Create a New File

```
1 # include <fcntl .h>
2 # include <stdio .h>
3 # include <sys / stat .h>
4 # include <sys / types .h>
5 # include <unistd .h>
6 int main ( int argc , char * argv [])
7 {
8 /* Chemin vers le nouveau fichier . */
9 char * path = argv [1];
10 /* Permissions du nouveau fichier . */
11 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH ;
12 /* Crée le fichier . */
13 int fd = open (path , O_WRONLY | O_EXCL | O_CREAT , mode );
14 if (fd == -1) {
15 /* Une erreur est survenue , affiche un message et quitte . */
16 perror (« open «);
17 return 1;
18 }
19 return 0;
20 }
```

Voici le programme en action :

```
% ./create-file testfile

% ls -l testfile

-rw-rw-r-- 1 samuel users 0 Feb 1 22:47 testfile

% ./create-file testfile

open: Le fichier existe
```

Notez que le fichier fait zéro octet car le programme n'y a écrit aucune donnée.

Fermer un fichier

Lorsque vous en avez fini avec un descripteur de fichier, fermez-le avec close. Dans certainscas, comme dans le cas du programme du Listing B.1 il n'est pas nécessaire d'appeler closeexplicitement car Linux ferme tous les descripteurs de fichiers lorsqu'un processus se termine(c'est-à-dire à la fin du programme). Bien sûr, une fois que vous avez fermé un descripteur defichier, vous ne pouvez plus l'utiliser.

La fermeture d'un descripteur de fichier peut déclencher des actions spécifiques de la partde Linux, selon la nature du descripteur de fichier. Par exemple, lorsque vous fermez un descripteurcorrespondant à un socket réseau, Linux ferme la connexion entre les deux ordinateurscommunicants via le socket.Linux limite le nombre de descripteurs de fichiers qu'un processus peut maintenir ouvertsen même temps. Les descripteurs de fichiers ouverts utilisent des ressources noyau, il est doncconseillé de fermer les descripteurs de fichiers dès que vous avez terminé de les utiliser. La limiteclassique est de 1024 descripteurs par processus. Vous pouvez l'ajuster avec l'appel systèmesetrlimit.

Écrire des données

Écrire des données dans un fichier se fait par le biais de l'appel write. Passz lui un descripteurde fichier, un pointeur vers un tampon de données et le nombre d'octets à écrire. Les donnéesécrites n'ont pas besoin d'être une chaine de caractères ; write copie des octets quelconquesdepuis le tampon vers le descripteur de fichier.

Le programme du Listing B.2 écrit l'heure courante à la fin du fichier passé sur la ligne decommande. Si le fichier n'existe pas, il est créé. Ce programme utilise également les fonctionstime, localtime et asctime pour obtenir et formater l'heure courante ; consultez leurs pages demanuel respectives pour plus d'informations.

Listing B.2 – (timestamp.c) – Ajoute l'Heure Courante à un Fichier

```
1 # include <fcntl .h>

2 # include <stdio .h>
```

```
3 # include <string .h>
4 # include <sys / stat .h>
5 # include <sys / types .h>
6 # include <time .h>
7 # include <unistd .h>
8 /* Renvoie une chaine représentant l'heure courante . */
9 char * get_timestamp ()
10 {
11 time_t now = time ( NULL );
12 return asctime ( localtime (& now ));
13 }
14 int main ( int argc , char * argv [])
15 {
16 /* Fichier auquel ajouter l'horodatage . */
17 char * filename = argv [1];
18 /* Récupère l'heure courante . */
19 char * timestamp = get_timestamp ();
20 /* Ouvre le fichier en écriture . S'il existe , ajoute les
données
21 à la fin , sinon , un nouveau fichier est créé . */
22 int fd = open ( filename , O_WRONLY | O_CREAT | O_APPEND ,
0666) ;
23 /* Calcule la longueur de la chaîne d'horodatage . */
24 size_t length = strlen ( timestamp );
25 /* L'écrit dans le fichier . */
26 write (fd , timestamp , length );
27 /* Fini . */
28 close (fd);
29 return 0;
30 }
```

Voici comment fonctionne ce programme :

```
% ./timestamp tsfile

% cat tsfile

Thu Feb 1 23:25:20 2001

% ./timestamp tsfile

% cat tsfile

Thu Feb 1 23:25:20 2001

Thu Feb 1 23:25:47 2001
```

Notez que la première fois que nous invoquons timestamp, il crée le fichier tsfile, alors quela seconde fois, les données sont ajoutées à la fin.

L'appel write renvoie le nombre d'octets effectivement écrits ou -1 si une erreur survient. Pour certains types de descripteurs de fichiers, le nombre d'octets écrits peut être inférieur aunombre d'octets demandés. Dans ce cas, c'est à vous d'appeler write encore une fois pour écrire le reste des données.

La fonction du Listing B.3 montre une façon de le faire. Notez que pourcertaines applications, vous pourriez avoir à effectuer des contrôles supplémentaires avant lareprise de l'écriture. Par exemple, si vous utilisez un socket réseau, il faudrait ajouter le codepermettant de détecter si la connexion a été fermée pendant l'écriture et, si c'est le cas, prendreles mesures adéquates.

Listing B.3 – (write-all.c) – Écrit tout le Contenu d'un Tampon

```
1 /* Écrit l' intégralité des COUNT octets de BUFFER vers le
descripteur

2 de fichier FD. Renvoie -1 en cas d'erreur ou le nombre d'octets

3 écrits . */

4 ssize_t write_all ( int fd , const void * buffer , size_t count
)

5 {

6 size_t left_to_write = count ;

7 while ( left_to_write > 0) {

8 size_t written = write (fd , buffer , count );

9 if ( written == -1)

10 /* Une erreur est survenue . Terminé . */

11 return -1;

12 else
```

```
13 /* Mémorise le nombre d'octets restant à écrire . */
14 left_to_write -= written ;
15 }
16 /* Nous ne devons pas avoir écrit plus de COUNT octets ! */
17 assert ( left_to_write == 0);
18 /* Le nombre d'octets écrits est exactement COUNT . */
19 return count ;
20 }
```

Lecture de Données

L'appel permettant de lire des données est read. Tout comme write, il prend en argumentsun descripteur de fichier, un pointeur vers un tampon et un nombre d'octets. Ce dernier indiquele nombre d'octets à lire à partir du descripteur. L'appel à read renvoie -1 en cas d'erreur oule nombre d'octets effectivement lus. Il peut être inférieur au nombre d'octets demandé, parexemple, s'il ne reste pas suffisamment d'octets à lire dans le fichier.

Le Listing B.4 présente un programme utilisant l'appel read. Il affiche une image hexadécimaledu contenu du fichier passé sur la ligne de commande. Chaque ligne affiche le déplacementdans le fichier et les 16 octets suivants.

Listing B.4 – (hexdump.c) – Affiche l'Image Hexadécimale d'un Fichier

```
1 # include <fcntl .h>
2 # include <stdio .h>
3 # include <sys / stat .h>
4 # include <sys / types .h>
5 # include <unistd .h>
6 int main ( int argc , char * argv [])
7 {
8 unsigned char buffer [16];
9 size_t offset = 0;
10 size_t bytes_read ;
11 int i;
12 /* Ouvre le fichier en lecture . */
13 int fd = open ( argv [1] , O_RDONLY );
```

```
14 /* Lit le fichier morceau par morceau , jusqu 'à ce que la
lecture

15 soit « trop courte «, c'est -à- dire que l'on lise moins que
ce que

16 l'on a demandé , ce qui indique que la fin du fichier est
atteinte . */

17 do {

18 /* Lis la « ligne « suivante . */

19 bytes_read = read (fd , buffer , sizeof ( buffer ));

20 /* Affiche le déplacement dans le fichier , suivi des octets
correspondants . */

21 printf («0x %06 x : «, offset );

22 for (i = 0; i < bytes_read ; ++i)

23 printf (« %02 x «, buffer [i]);

24 printf («\n»);

25 /* Conserve notre position dans le fichier . */

26 offset += bytes_read ;

27 }

28 while ( bytes_read == sizeof ( buffer ));

29 /* Terminé . */

30 close (fd);

31 return 0;

32 }
```

Voici hexdump en action. Il affiche l'image de son propre exécutable :

```
% ./hexdump hexdump

0x000000 : 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00

0x000010 : 02 00 03 00 01 00 00 00 c0 83 04 08 34 00 00 00

0x000020 : e8 23 00 00 00 00 00 00 34 00 20 00 06 00 28 00

0x000030 : 1d 00 1a 00 06 00 00 00 34 00 00 00 34 80 04 08

...
```

La sortie peut être différente chez vous, selon le compilateur utilisé pour construire hexdump estles options de compilation.

Se déplacer dans un Fichier

Un descripteur de fichier connait sa position dans le fichier. Lorsque vous y écrivez ou quevous y lisez, sa position est modifiée selon le nombre d'octets lus ou écrits. Parfois, cependant,vous pouvez avoir besoin de vous déplacer dans un fichier sans lire ou écrire de données. Parexemple, vous pouvez vouloir écrire au milieu d'un fichier sans en modifier le début, ou vouspouvez avoir besoin de retourner au début d'un fichier et de le relire sans avoir à le réouvrir.

L'appel lseek vous permet de modifier votre position dans un fichier. Passez-lui le descripteurdeuxfichiers et deux autres arguments indiquant la nouvelle position.

Si le troisième argument est SEEK_SET, lseek interprète le second argument comme une position, en octets, depusi le début du fichier ;

Si le troisième argument est SEEK_CUR, lseek interprète le second argument comme un déplacement, positif ou négatif, depuis la position courante ;

Si le troisième argument est SEEK_END, lseek interprète le second argument comme un déplacement à partir de la fin du fichier. Une valeur positive indique une position au-delà de la fin du fichier.

L'appel à lseek renvoie la nouvelle position, sous forme d'un déplacement par rapport au débutdu fichier.Le type de ce déplacement est off_t. Si une erreur survient, lseek renvoie -1. Vous nepouvez pas utiliser lseek avec certains types de descripteurs de fichiers comme les sockets.

Si vous voulez obtenir votre position dans un fichier sans la modifier, indiquez un déplacementde 0 par rapport à votre position actuelle – par exemple :

```
off_t position = lseek ( file_descriptor , 0, SEEK_CUR );
```

Linux autorise l'utilisation de lseek pour spécifier une position au-delà de la fin du fichier. Normalement, si un descripteur de fichier est positionné à la fin d'un fichier et que vous y écrivez,Linux augmente automatiquement la taille du fichier pour faire de la place pour les nouvellesdonnées. Si vous indiquez une position au-delà de la fin du fichier puis que vous y écrivez, Linuxcommence par agrandir le fichier de la taille du «trou» que vous avez créé en appelant lseekpuis écrit à la fin de celui-ci. Ce trou n'occupe toutefois pas de place sur le disque ; Linux notesimplement sa taille. Si vous essayez de lire le fichier, il apparait comme s'il était rempli d'octetsnuls.

En utilisant cette propriété de lseek, il est possible de créer des fichiers extrêmement grandsqui n'occupent pratiquement aucun espace disque. Le programme lseek-huge du Listing B.5 lefait. Il prend en arguments de ligne de commande un nom de fichier et sa taille, en mégaoctets.Le programme ouvre un nouveau fichier, se place après la fin de celui-ci en utilisant lseek puisécrit un octet à 0 avant de fermer le fichier.

Listing B.5 – (lseek-huge.c) – Créer de Gros Fichiers avec lseek

```
1 # include <fcntl .h>
2 # include <stdlib .h>
3 # include <sys / stat .h>
4 # include <sys / types .h>
5 # include <unistd .h>
6 int main ( int argc , char * argv [])
7 {
8 int zero = 0;
9 const int megabyte = 1024 * 1024;
10 char * filename = argv [1];
11 size_t length = ( size_t ) atoi ( argv [2]) * megabyte ;
12 /* Ouvre un nouveau fichier . */
13 int fd = open ( filename , O_WRONLY | O_CREAT | O_EXCL , 0666)
;
14 /* Se place un octet avant la fin désirée du fichier . */
15 lseek (fd , length - 1, SEEK_SET );
16 /* Écrit un octet nul . */
17 write (fd , &zero , 1);
18 /* Terminé . */
19 close (fd);
20 return 0;
21 }
```

Activité 3: Structure stat

Structure stat

En utilisant open et read, vous pouvez extraire le contenu d'un fichier. Mais qu'en est-il desautres informations ? Par exemple, la commande ls -l affiche des informations comme la taille,la date de dernière modification, les permissions ou le propriétaire pour les fichiers du répertoirecourant.

L'appel stat récupère ces informations pour un fichier. Appelez stat en lui passant le chemindu fichier sur lequel vous voulez des informations et un pointeur sur une variable de type structstat. Si l'appel à stat se déroule correctement, il revoie 0 et renseigne les champs de la structureavec des informations sur le fichier ; sinon, il renvoie -1.

Voici les champs les plus intéressants d'une struct stat :

st_mode contient les permissions du fichier.

En plus des permissions clasiques, le champ st_mode encode le type de fichier dans les bits de poids fort. Consultez les explications ci-dessous pour savoir comment le décoder ;

st_uid et st_gid contiennent les identifiants de l'utilisateur et du groupe auxquels le fichier appartient, respectivement.

st_size contient la taille du fichier, en octets ;

st_atime contient la date de dernier accès au fichier (en lecture ou écriture) ;

st_mtime contient la date de dernière modification du fichier.

Un certain nombre de macros analysent la valeur du champ st_mode pour déterminer le typede fichier sur lequel stat a été invoqué. Une macro vaut vrai si le fichier est de ce type :

S_ISBLK (mode) périphérique bloc

S_ISCHR (mode) périphérique caractère

S_ISDIR (mode) répertoire

S_ISFIFO (mode) fifo (canal nommé)

S_ISLNK (mode) lien symbolique

S_ISREG (mode) fichier classique

S_ISSOCK (mode) socket

Le champ st_dev contient les numéros de périphérique majeur et mineur du dispositif matérielsur lequel le fichier se situe. Lenuméro majeur de périphérique est décalé à gauche de 8 bits, tandis que le numéro mineuroccupe les 8 bits de poids faible. Le champ st_ino contient le numéro d'inode du fichier. Celasitue le fichier sur le système de fichiers.

Si vous appelez stat sur un lien symbolique, stat suit le lien et vous renvoie les informationssur le fichier sur lequel il pointe, pas sur le lien symbolique. Cela signifie que S_ISLNK ne

serajamais vrai pour une valeur renvoyée par stat. Utilisez la fonction lstat si vous ne voulez passuivre les liens symboliques ; cette fonction renvoie les informations sur le lien et non pas sur lefichier cible. Si vous appelez lstat sur un fichier qui n'est pas un lien symbolique, elle a le mêmecomportement que stat. Appeler stat sur un lien invalide (un lien qui pointe vers un fichierinexistant ou inaccessible) provoque une erreur, alors que l'appel de lstat sur le même fichiern'a aucune incidence.

Si vous disposez déjà d'un fichier ouvert en lecture/écriture, appelez fstat au lieu de stat.Il prend un descripteur de fichier en premier argument à la place du chemin vers le fichier.

Le Listing B.6 présente une fonction qui alloue un tampon suffisamment long pour recevoirle contenu du fichier et charge les données à l'intérieur. Elle utilise fstat pour déterminer lataille du tampon à allouer et vérifier que le fichier est bien un fichier classique.

Listing B.6 – (read-file.c) – Charge un Fichier dans un Tampon

```
1 # include <fcntl .h>

2 # include <stdio .h>

3 # include <sys / stat .h>

4 # include <sys / types .h>

5 # include <unistd .h>

6 /* Charge le contenu de FILENAME dans un tampon nouvellement
alloué . La

7 taille du tampon est stockée dans * LENGTH . Renvoie le tampon ,
qui devra

8 être libéré par l'appelant . Renvoie NULL si FILENAME ne
correspond pas

9 à un fichier régulier . */

10 char * read_file ( const char * filename , size_t * length )

11 {

12 int fd;

13 struct stat file_info ;

14 char * buffer ;

15 /* Ouvre le fichier . */

16 fd = open ( filename , O_RDONLY );

17 /* Récupère les informations sur le fichier . */

18 fstat (fd , & file_info );

19 * length = file_info . st_size ;
```

```
20 /* S'assure que le fichier est un fichier ordinaire . */
21 if (! S_ISREG ( file_info . st_mode )) {
22 /* Ce n'en est pas un , abandonne . */
23 close (fd);
24 return NULL ;
25 }
26 /* Alloue un tampon suffisamment grand pour recevoir le contenu
du fichier . */
27 buffer = ( char *) malloc (* length );
28 /* Charge le fichier dans le tampon . */
29 read (fd , buffer , * length );
30 /* Terminé . */
31 close (fd);
32 return buffer ;
33 }
```

Écriture et Lecture Vectorielles

L'appel write prend en arguments un pointeur vers le début d'un tampon de données et lataille de ce tampon. Il écrit le contenu d'une région contigüe de mémoire vers un descripteurde fichier. Cependant, un programme a souvent besoin d'écrire plusieurs éléments de données,chacun se trouvant à un endroit différent. Pour utiliser write, un tel programme devrait soitcopier tous les objets dans une région contigüe, ce qui gaspillerait des cycles CPU et de lamémoire, soit effectuer de multiples appels à write.

Pour certaines applications, appeler write plusieurs fois peut être inefficace ou peu souhaitable.Par exemple, lors de l'écriture vers un socket réseau, deux appels à write peut provoquerl'envoi de deux paquets sur le réseau, alors que les mêmes données auraient pu être envoyées enune fois si un seul appel à write avait été possible.

L'appel writev vous permet d'envoyer le contenu de plusieurs zones mémoire non-contigüesvers un descripteur de fichier en une seul opération. C'est ce que l'on appelle l'écriture vectorielle.La contrepartie de l'utilisation de writev est que vous devez créer une structure dedonnées indiquant le début et la taille de chaque région mémoire. Cette structure est un tableaud'éléments struct iovec. Chaque élément correspond à un emplacement mémoire à écrire ; leschamps iov_base et iov_len correspondent respectivement à l'adresse du début de la région età sa taille. Si vous connaissez à l'avance le nombre de zones mémoire à écrire, vous pouvez vouscontenter de déclarer un tableau de struct iovec ; si le nombre de régions peut varier, vousdevez allouer le tableau dynamiquement.

Appelez writev en lui passant le descripteur de fichier vers lequel envoyer les données, letableau de struct iovec et le nombre d'éléments contenus dans ce tableau. La valeur de retourcorrespond au nombre d'octets écrits.

Lien avec les Functions d'E/S Standards du C

Nous avons évoqué le fait que les fonctions d'E/S standards du C sont implémentées commeune surcouche de ces fonctions d'E/S de bas niveau. Parfois, cependant, il peut être pratiqued'utiliser les fonctions de la bibliothèque standard avec des descripteurs de fichiers ou lesfonctions de bas niveau sur un flux FILE*. GNU/Linux autorise ces deux pratiques.

Si vous avez ouvert un fichier en utilisant fopen, vous pouvez obtenir le descripteur de fichiersous-jacent par le biais de la fonction fileno. Elle prend un paramètre FILE* et renvoie ledescripteur de fichier. Par exemple, pour ouvrir un fichier avec l'appel standard fopen mais yécrire avec writev, vous pourriez utiliser une séquencence telle que :

```
FILE * flux = fopen ( nomfichier , «w»);

int descripteur = fileno ( flux );

writev ( descripteur , tableau , taille_tableau );
```

Notez que flux et descripteur correspondent tous deux au même fichier. Si vous appelez fclosecomme ceci, vous ne pourrez plus écrire dans descripteur :

```
fclose ( flux );

De même, si vous appelez close, vous ne pourrez plus écrire dans
flux :

close ( descripteur );
```

Pour effectuer l'opération inverse, obtenir un flux à partir d'un descripteur, utilisez la fonctionfdopen. Elle produit un pointeur FILE* correspondant à un descripteur de fichier. La fonctionfdopen prend en paramètres un descripteur de fichier et une chaîne correspondant au mode danslequel le flux doit être ouvert. La syntaxe de l'argument de mode est la même que pour le secondargument de fopen et il doit être compatbile avec le descripteur de fichier. Par exemple, passez lachaine de mode r pour un descripteur de fichier en lecture ou w pour un descripteur de fichier enécriture. Comme pour fileno, le flux et le descripteur de fichier font référence au même fichier,ainsi, si vous en fermez l'un des deux, vous ne pouvez plus utiliser l'autre.

Évaluation

1. Considérer le programme "createfile.c" fournit ci-après :

```
//createfile.c Create a New File

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main (int argc, char* argv[])

{

/* The path at which to create the new file. */

char* path = argv[1];

/* The permissions for the new file. */

mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |

S_IROTH;

/* Create the file. */

int fd = open (path, O_WRONLY | O_EXCL | O_CREAT, mode);

if (fd == -1)

{

/* An error occurred. Print an error message and bail. */

perror ("open");

return 1;

}

  return 0;

}
```

Quel est le résultat produit par ce programme dans chacun des cas suivants :

a. Copier et exécuterle program (GNU/Linux platform) pour créer un fichier nommé "demofile"

b. Utiliser la commande Linux ls –l pour afficher les attributs du fichier “demofile”. Quelle est la taille de “demofile”? Pourquoi ?

c. Répéterl’opération (a) ci-dessus

2.Considérerle programme “timestamp.c” ci-dessous :

```
//timestamp.c Append a Timestamp to a File

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <time.h>

#include <unistd.h>

/* Return a character string representing the current
date and time. */

char* get_timestamp ()

{

time_t now = time (NULL);

return asctime (localtime (&now));

}

int main (int argc, char* argv[])

{

/* The file to which to append the timestamp. */

char* filename = argv[1];

/* Get the current timestamp. */

char* timestamp = get_timestamp ();

/* Open the file for writing. If it exists, append to it;

otherwise, create a new file. */

int fd = open (filename, O_WRONLY | O_CREAT | O_APPEND,
0666);

/* Compute the length of the timestamp string. */

size_t length = strlen (timestamp);
```

```
/* Write the timestamp to the file. */
write (fd, timestamp, length);
/* All done. */
close (fd);
return 0;
}
```

Quel est le résultat produit par ce programme dans chacun des cas suivants :

a. Copier et exécuterle program (GNU/Linux platform) pour créer un fichier nommé "demo_tsfile"

b. Utiliser la commande Linux catpour afficher le contenu du fichier "demo_tsfile"

c. Répéter les opérations (a) et (b) ci-dessus. Donner les raisons des résultats observés durant cette étape.

Résumé de l'unité

Dans cette unité, nous avons fourni un aperçu de la bibliothèque GNU C, avec un certain nombre dequestions concernant la bibliothèque C GNU, y compris les normes de la bibliothèque et la portabilité, les bases de l'utilisation de la bibliothèque, et le cas des fonctions de la bibliothèque C pour l'allocation de stockage de données des programme.

En outre, nous avons décrit les fonctions de la bibliothèque GNU C pour manipuler les fichiers etrépertoires. Plus précisément, les fonctions relatives à l'examen ou la modification de répertoires ettravaillant avec plusieurs noms de fichiers ont été présentées.

Enfin, l'unité a présenté les mécanismes d'entrée/sortie (E/S). Les demandes de lecture et d'écriture de données ont été explorées, les appels pour extraire des informations des fichiers, le noyau E/S et les appels relatifs à des fonctions standard C de la Bibliothèque d'E/S.

Évaluation de l'unité

Vérifiez votre compréhension!

Instructions

Cette évaluation doit être fait en individuel

Il s'agit d'un devoir à rendre

Lisez attentivement le cours et répondez aux questions suivantes

Examiner les paragraphes ci-dessous, qui fournissent des informations complémentaires sur umask et les fichiers texte DOS / Windows.

Exercices de programmation de niveau intermédiaire

Ayant travaillé jusqu'ici avec cet appareil, nous sommes positifs que vous avez choisi d'écrire tous vosprogrammes pour GNU / Linux. Cependant, vos programmes peuvent parfois besoin de lire des fichiers textegénéré par les programmes DOS ou Windows. Il est important de prévoir une différence importante danscomment des fichiers texte sont structurés entre ces deux plates-formes.

Dans les fichiers de texte GNU / Linux, chaque ligne est séparée de la suivante par un caractère de nouvelle ligne. Une nouvelle ligneest représenté par la constante caractère '\ n', qui a le code ASCII 10. Sous Windows, cependant,les lignes sont séparées par un caractère deuxcombinaisons : un caractère de retour chariot (le caractère '\ r,'qui a le code ASCII 13), suivi d'un caractère de nouvelle ligne.

Certains éditeurs de texte GNU / Linuxaffichage ^ M à la fin de chaque ligne lors de l'affichage d'un texte de Windows fichier, tel est le retour chariotpersonnalisé. Emacs affiche les fichiers de texte de Windows correctement, mais les indique en montrant (DOS) enla ligne de mode dans le bas du tampon.

Certains éditeurs Windows, tels que le Bloc-notes, afficher tout le texte dans un fichier texte GNU / Linux sur une seulligne parce qu'ils attendent un retour chariot à la fin de chaque ligne. D'autres programmes à la foisGNU / Linux et Windows que les fichiers texte de processus peuvent rapporter des erreurs mystérieuses lorsqu'il est administré enentrée d'un fichier texte dans le mauvais format.

Si votre programme lit les fichiers texte générés par Windowsprogrammes, vous aurez probablement envie de remplacer la séquence «\ r \ n 'avec un seul saut de ligne. De même, sivotre programme écrit des fichiers texte qui doit être lu par les programmes Windows, remplacez nouvelle ligne par un seulcaractère avec des combinaisons '\ r \ n «. Vous devez le faire si vous utilisez le niveau basappels d'E / Sou à la bibliothèque C standard des fonctions d'E/S.

1. Vous devez changer umask de la coquille et la suite du processus, par la suite, à chaque changement, exécuter le programme «CreateFile.c» donné à la question 1 de l'évaluation 1.3.4. A chaque retour, enregistrer la valeur de umask utilisé et analyser les résultats observés à partir de l'exécution du programme.
2. Ecrire un programme qui lit correctement les fichiers texte générés par des programmes DOS / Windows.

Lectures et autres ressources

Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001

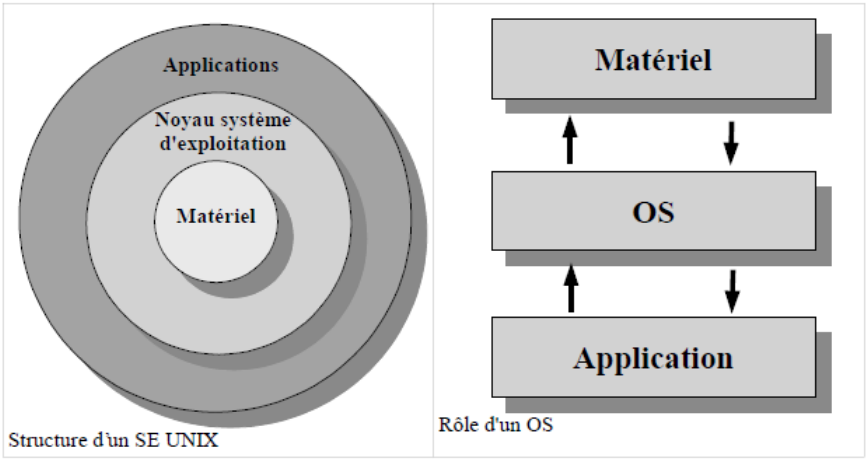
http://www.acm.uiuc.edu/webmonkeys/book/c_guide/: The C Library Reference Guide

http://www.delorie.com/gnu/docs/glibc/libc_toc.html: The GNU C Library

Unité 2. Programmation shell et Code assembleur en ligne

Introduction à l'unité

Un langage de script est un langage de programmation qui permet de manipuler les fonctionnalités d'un système informatique configuré pour fournir à l'interpréteur de ce langage un environnement et une interface qui déterminent les possibilités de celui-ci. Le langage de script peut alors s'affranchir des contraintes de bas niveau — prises en charge par l'intermédiaire de l'interface — et bénéficier d'une syntaxe de haut niveau.



Dans le sens le plus traditionnel, qui est celui des shell scripts, un script sert principalement à lancer et coordonner l'exécution de programmes.

Dans sa version la plus simple, un script ne spécifie qu'une suite de programmes à appeler dans un ordre donné (par exemple éditeur de texte, compilateur, éditeur de liens et exécution du code objet). Cela crée rapidement des inefficiences : à quoi bon en effet tenter de charger et d'exécuter un programme dont la compilation a échoué ?

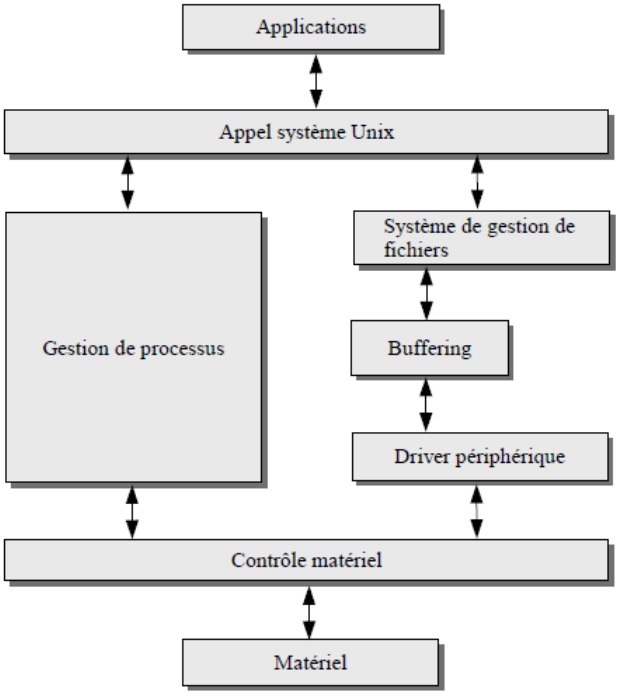
On y ajoute donc assez vite une possibilité d'exécution conditionnelle simple (||, &&) en fonction du résultat de l'étape immédiatement précédente (il ne sert à rien d'exécuter si la compilation est mauvaise).

La possibilité d'y employer des variables, des paramètres, des structures de contrôle (répétition, exécution conditionnelle), etc. fait des langages de scripts de véritables langages de programmation.

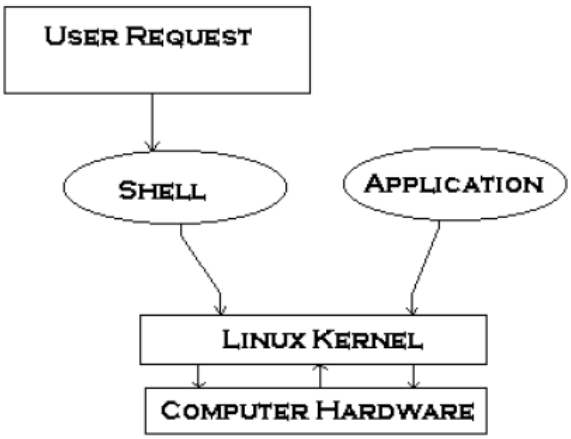
Les scripts sont utilisés à différents niveaux :

- côté OS comme une méthode d'automatisation (VBA de Microsoft, python de Blender) ;
- côté serveur web dits Server Side Includes comme une méthode pour générer des pages dynamiques (JSP, PHP, ASP, ASP.NET) ;

- côté navigateur dits Client Side sous forme d'Applet (actionscript, C#, Applet java) d'automatisation (VBA de Microsoft). c'est un langage binaire ;
- Interface graphique dit gui scripts, permettant de simuler des déplacements de la souris et des pressions de touches sur le clavier.



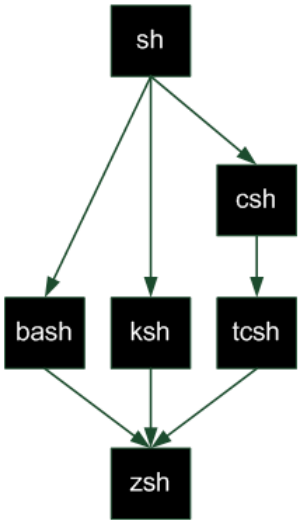
Un shell Unix est un interpréteur de commandes destiné aux systèmes d'exploitation Unix et de type Unix qui permet d'accéder aux fonctionnalités internes du système d'exploitation. Il se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal. L'utilisateur lance des commandes sous forme d'une entrée texte exécutée ensuite par le shell. Dans les différents systèmes d'exploitation Microsoft Windows, le programme analogue est command.com, ou cmd.exe.



Les systèmes d'exploitation de type Unix disposent le plus souvent d'un shell. À l'origine, l'interpréteur de commandes par défaut était sh, qui évolua en de nombreuses versions, dont csh, étendu en tcsh, ou ksh, ou encore rc... Mais aujourd'hui bash, s'inspirant de sh, ksh, et csh, est le shell le plus répandu, bien qu'il existe d'autres interpréteurs de commandes, comme zsh, ou ash.

Voici dans les grandes lignes comment ont évolué les shells. Chacun hérite de la plupart des fonctionnalités de son ancêtre (figure suivante) :

- Shell de Stephen Bourne
- Bourne shell (sh)
- Bourne-Again shell (bash)
- csh : C shell
- tcsh (Tenex C shell, version moderne du csh)
- Shell de David Korn : Korn shell (ksh)
- Shell de Kenneth Almquis (en), utilisé lorsqu'il est nécessaire d'avoir un shell prenant peu de place sur le disque, clone de la variante SVR4 du Bourne shell;
- Ash (Almquist SHell)
- Dash shell (Debian Almquist SHell)
- Z Shell (zsh), reprenant les fonctions les plus pratiques de bash, ksh et tcsh.



En résumé de cette introduction, la programmation shell :de quoi s'agit-il ?

Imaginez un minilangage de programmation intégré à Linux. Ce n'est pas un langage aussi complet que peuvent l'être le C, le C++ ou le Java par exemple, mais cela permet d'automatiser la plupart de vos tâches : sauvegarde des données, surveillance de la charge de votre machine, etc. On aurait très bien pu faire tout cela en créant un programme en C par exemple. Le gros avantage du langage shell est d'être totalement intégré à Linux : il n'y a rien à installer, rien à compiler. Et surtout : vous avez très peu de nouvelles choses à apprendre. En effet, toutes les commandes que l'on utilise dans les scripts shell sont des commandes du système que vous connaissez déjà : ls, cut, grep, sort…

Cette unité vous permettra d'acquérir la compétence de la programmation shell, en réalisantvos premiers scripts shell avant de passer aux programmes plus complexes.

Objectifs de l'unité

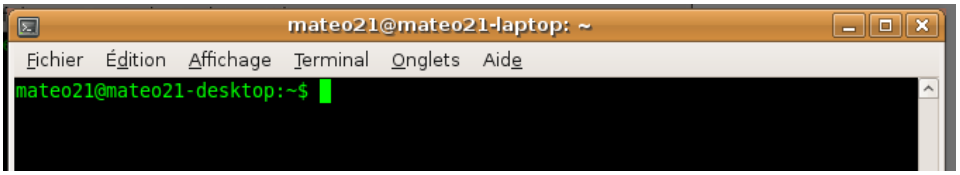
À la fin de cette unité, vous devriez être capable de:

- Définir les concepts de shell et de l'assemblage en ligne
- Utiliser le shell pour automatiser différentes tâches (utilisateur/OS)
- Comprendre les techniques de mixage du shell avec des instructions en code C
- Utiliser conjointement les programmes shell et les codes C pour accélérer l'éxécution des instructions par le processeur.

Shell	Un interpréteur de langage de commande qui exécute les commandes lues à partir du périphérique d'entrée standard (Clavier) ou d'un fichier.
Programme ou script shell	Le shell interprète les commandes de l'utilisateur, qui sont soit directement saisi par l'utilisateur, ou qui peuventêtre lues à partir d'un fichier appelé le script shell ouprogramme. Les scripts shell sont interprétés, noncompilé. Le shell lit les commandes duscript ligne par ligne et les recherches parmi celles du système, tandis que d'un compilateurconvertit un programme en forme lisible par machine,un fichier exécutable quipeut ensuite être utilisé dans unShell script.
Langage assembleur	Un langage d'assemblage ou langage assembleur est, en programmation informatique, un langage de bas niveau qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des symboles dits « mnémoniques », c'est-à-dire faciles à retenir. Le programme assembleur convertit ces mnémoniques en langage machine en vue de créer par exemple un fichier objet ou un fichier exécutable.
Noyau	Le noyau est le module central d'une exploitationsystème. Il est la partie première du système d'exploitation chargée en premier, et il reste dans la mémoire principale
Système d'exploitation (OS)	Un logiciel qui gère le matériel informatique et les ressources logicielles et qui fournit des services communspour des programmes informatiques

Activité 2: Les bases du shell

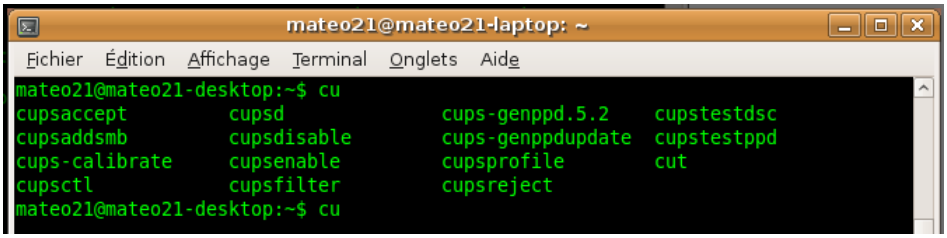
Le shell est le programme qui gère l'invité de commandes. C'est donc le programme qui attend que vous rentriez des commandes (comme l'illustre la figure suivante) :



C'est aussi le programme qui est capable par exemple de :

se souvenir quelles étaient les dernières commandes tapées (vous remontez dans votre historique en appuyant sur la flèche « Haut » ou en faisant une recherche avec un Ctrl + R) ;

autocompléter une commande ou un nom de fichier lorsque vous appuyez sur Tab (figure suivante) ;



gérer les processus (envoi en arrière-plan, mise en pause avec Ctrl + Z…) ;

rediriger et chaîner les commandes (les fameux symboles >, <, |, etc.) ;

définir des alias (par exemple ll signifie chez moi ls -lArth) ;

bref, le shell fournit toutes les fonctionnalités de base pour pouvoir lancer des commandes simples et des séries de commandes complexes (i.e des programmes shell).

Pour trouver le shell en cours d'utilisation dans un environnement Linux/Unix, il suffit de taper la commande suivante :

$ echo $SHELL

En cas de nécessité de changement de shell, vous pouvez installer un nouveau shell (par exemple, avec la commande # apt-get install kshsous Ubuntu), et demander ensuite à l›utiliser pour votre compte utilisateur. Pour cela, tapez la commande :

$ chsh

chsh signifie Change Shell.

On vous demandera où se trouve le programme qui gère le nouveau shell. Vous devrez donc indiquer /bin/ksh pour ksh, /bin/sh pour sh, /bin/bash pour bash, etc.Quelle importance a tout ceci lorsque l'on réalise un script shell ? Si nous vous parlons de cela, c'est parce qu'un script shell dépend d'un shell précis. En gros, le langage n'est pas tout à fait le même selon que vous utilisez sh, bash, ksh, etc. Il est possible d'écrire des scripts sh par exemple. Ceux-là, nous sommes sûrs qu'ils fonctionnent partout car tout le monde possède un shell sh.

Il s'agit toutefois du plus vieux shell, or écrire des scripts en sh est certes possible mais n'est franchement ni facile, ni ergonomique.

On vous propose donc d'étudier le bash dans ce cours car :

- on le trouve par défaut sous Linux et Mac OS X (cela couvre assez de monde !) ;
- il rend l'écriture de scripts plus simple que sh ;
- il est plus répandu que ksh et zsh sous Linux ;
- En clair, le bash est un bon compromis entre sh (le plus compatible) et ksh / zsh (plus puissants).

Premier script shell

Nous allons commencer par écrire un premier script bash tout simple. Il ne sera pas révolutionnaire mais va nous permettre de voir les bases de la création d'un script et comment celui-ci s'exécute. Cela sera donc essentiel pour la suite.

Création du fichier

Commençons par créer un nouveau fichier pour notre script. Le plus simple est d'ouvrir un éditeur de texte (par exemple Vim) en lui donnant le nom du nouveau fichier à créer :

$ vim essai.sh

Si essai.sh n'existe pas, il sera créé. Nous avons donné ici l'extension .sh au fichier essai. On le fait souvent par convention pour indiquer que c'est un script shell, mais sachez que ce n'est pas une obligation. Certains scripts shell n'ont d'ailleurs pas d'extension du tout. On aurai donc pu l'appeler essai tout court.

Indiquer le nom du shell utilisé par le script

Vim est maintenant ouvert et vous avez un fichier vide sous les yeux.La première chose à faire dans un script shell est d'indiquer lequel des shells est utilisé. En effet, comme nous l'avons vu plus tôt, la syntaxe du langage change un peu selon qu'on utilise sh, bash, ksh, etc.

En ce qui nous concerne, nous souhaitons utiliser la syntaxe de bash, plus répandu sous Linux et plus complet que sh. Nous indiquons où se trouve le programme bash :

#!/bin/bash

Le #! est appelé le sha-bang.

/bin/bash peut être remplacé par /bin/sh si vous souhaitez coder pour sh, /bin/ksh pour ksh, etc.

Bien que non indispensable, cette ligne permet de s'assurer que le script est bien exécuté avec le bon shell.

En l'absence de cette ligne, c'est le shell de l'utilisateur qui sera chargé. Cela pose un problème : si votre script est écrit pour bash et que la personne qui l'exécute utilise ksh, il y a de fortes chances pour que le script ne fonctionne pas correctement !

La ligne du sha-bang permet donc de « charger » le bon shell avant l'exécution du script. À partir de maintenant, vous devrez la mettre au tout début de chacun de vos scripts.

Exécution de commandes

Après le sha-bang, nous pouvons commencer à coder.

Le principe est très simple : il vous suffit d'écrire les commandes que vous souhaitez exécuter. Ce sont les mêmes que celles que vous tapiez dans l'invite de commandes :

ls : pour lister les fichiers du répertoire,

cd : pour changer de répertoire,

mkdir : pour créer un répertoire,

grep : pour rechercher un mot,

sort : pour trier des mots, etc.

On va commencer par quelque chose de très très simple : un ls. On va donc créer un script bash qui va juste se contenter d'afficher le contenu du dossier courant. Donc notre fichier essai.sh contient les deux lignes suivantes :

```
#!/bin/bash

ls
```

Ajout de commentaires

Notez que vous pouvez aussi ajouter des commentaires dans votre script. Ce sont des lignes qui ne seront pas exécutées mais qui permettent d'expliquer ce que fait votre script.

Tous les commentaires commencent par un #. Notre script essai.sh devient alors, par exemple :

```
#!/bin/bash

# Affichage de la liste des fichiers
ls
```

Vous avez sûrement remarqué que la ligne du sha-bang commence aussi par un #… Oui, c'est uncommentaire aussi, mais considérez que c'est un commentaire « spécial » qui a un sens. Il fait un peu exception.

Exécuter le script bash

Nous avons écrit un petit script sans prétention de deux-trois lignes. Notre mission maintenant est de parvenir à l'exécuter.Commencez donc par enregistrer votre fichier essai.sh et fermez votre éditeur. Sous Vim, il suffit de taper :wq ou encore :x.Vous retrouvez alors l'invite de commandes.

Donner les droits d'exécution au script

Si vous faites un ls -l pour voir votre fichier qui vient d'être créé, vous obtenez ceci :

```
$ ls -l

total 4

-rw-r--r-- 1 mateo21 mateo21 17 2009-03-13 14:33 essai.sh
```

Ce qui nous intéresse ici, ce sont les droits sur le fichier : -rw-r--r--.

Si vous vous souvenez un petit peu des notions sur les droits des fichiers, vous devriez vous rendre compte que notre script peut être lu par tout le monde (r), écrit uniquement par nous (w), et n'est pas exécutable (pas de x).

Or, pour exécuter un script, il faut que le fichier ait le droit « exécutable ». Le plus simple pour donner ce droit est d'écrire :

```
$ chmod +x essai.sh
```

Vous pouvez vérifier que le droit d'exécution a bien été donné au script essai.sh :

```
$ ls -l

total 4

-rwxr-xr-x 1 mateo21 mateo21 17 2009-03-13 14:33 essai.sh
```

Tout le monde a maintenant le droit d'exécuter le script. Si vous voulez, vous pouvez limiter ce droit à vous-mêmes mais pour cela je vous invite à revoir les notions sur les droits des fichiers.

Exécution du script

Le script s'exécute maintenant comme n'importe quel programme, en tapant « ./» devant le nom du script:

```
$ ./essai.sh

essai.sh
```

Que fait le script ? Il fait juste un ls, donc il affiche la liste des fichiers présents dans le répertoire (ici, il y avait seulement essai.sh dans notre répertoire).

Bien entendu, ce script est inutile ; il était plus simple de taper ls directement. Cependant, vous devez vous douter que l'on va pouvoir faire beaucoup mieux que ça avec la programmation shell.

Vous pouvez déjà modifier votre script pour qu'avant toute chose il vous donne également le nom du répertoire courant dans lequel vous vous trouvez :

#!/bin/bash

Affichage du répertoire courant et de la liste des fichiers qu'il contient

pwd

ls

Les commandes seront exécutées l'une après l'autre :

$ ./essai.sh

/home/mateo21/scripts

essai.sh

Exécution de débogage

Plus tard, vous ferez probablement de gros scripts et risquerez de rencontrer des bugs. Il faut donc dès à présent que vous sachiez comment déboguer un script. Pour cela, il faut l'exécuter comme ceci :

$ bash -x essai.sh

On appelle en fait directement le programme bash et on lui ajoute en paramètre un -x (pour lancer le mode débogage) ainsi que le nom de notre script à déboguer.

Le shell affiche alors le détail de l'exécution de notre script, ce qui peut nous aider à retrouver la cause de nos erreurs :

```
$ bash -x essai.sh
+ pwd
/home/mateo21/scripts
+ ls
essai.sh
```

Créer sa propre commande

Actuellement, le script doit être lancé via ./essai.sh et vous devez être dans le bon répertoire. Ou alors vous devez taper le chemin en entier, comme /home/mateo21/scripts/essai.sh.

Comment font les autres programmes pour pouvoir être exécutés depuis n'importe quel répertoire sans « ./ » devant ?Ils sont placés dans un des répertoires du PATH. Le PATH est une variable système qui indique où sont les programmes exécutables sur votre ordinateur. Si vous tapez $ echo $PATH vous aurez la liste de ces répertoires « spéciaux ».

Il vous suffit donc de déplacer ou copier votre script dans un de ces répertoires, comme /bin, /usr/bin ou /usr/local/bin (ou encore un autre répertoire du PATH). Notez qu'il faut être root pour pouvoir faire cela.

Une fois que c'est fait, vous pourrez alors taper simplement essai.sh pour exécuter votre programme et ce quel que soit le répertoire dans lequel vous vous trouverez !

```
$ essai.sh

/home/mateo21/scripts

essai.sh
```

En résumé

Contrairement aux apparences, il existe plusieurs environnements console différents : ce sont les shells. Ce sont eux qui gèrent l'invite de commandes et ses fonctionnalités comme l'historique des commandes, la recherche Ctrl + R, l'autocomplétion des commandes…

Le shell que nous avons utilisé ici est bash, mais il existe aussi sh, ksh, zsh, etc.

Il est possible d'automatiser une série de commandes. On crée pour cela un fichier contenant la liste des commandes à exécuter, appelé script shell. On dit que l'on fait de la programmation shell.

En fonction du shell utilisé, on dispose de différents outils pour créer son script shell. Nous avons utilisé ici bash, donc notre fichier de script doit commencer par la ligne #! /bin/bash.

Dans le fichier de script, il suffit d'écrire les commandes à exécuter les unes après les autres, chacune sur une ligne différente.

Pour exécuter le script (et donc exécuter la liste des commandes qu'il contient) il faut donner les droits d'exécution au fichier ($ chmod +x script.sh) et lancer l'exécution du script avec la commande ./script.sh.

En savoir plus sur les commandes

type – vous précise si la commande est issue du système d'exploitation (builtin), un alias, ouun programme (externeau shell)

which –vous dit dans quel répertoire se trouve l'utilitaire

help –affiche les informationsd'aide de la commande

info bash –des informations à lire à propos du shell BASH

Exemple:

```
$ which echo

/bin/echo

$ type –aecho

echo is a shell builtin

echo is /bin/echo

$ help echo# help provides information on builtin commands

echo: echo [-neE] [arg ...]
```

Output the ARGs. If –n is specified, the trailing newline is suppressed. If the –e option is given, interpretation of the following backslashescaped characters is turned on:

```
\a alert (bell)

\b backspace

\c suppress trailing newline

\E escape character

\f form feed

\n new line

\r carriage return

\t horizontal tab

\v vertical tab

\\ backslash

\num the character whose ASCII code is NUM (octal).
```

Fichiers d'initialisation et variables

I.Fichiers d'initialisation

Les commandes et variables placées dans les fichiers d'initialisation sont lues lorsque le shell est démarré. Si les variablessont placées dans les fichiers d'initialisation du système, elles sont mises à disposition de chaque shell. Des personnalisations doivent être faites afin de pouvoir exportercertaines commandes ou variables pour qu'elles puissent être disponibles. Les commandes et les alias ne pouvant pas être exportées doivent donc être placées dans des fichiers d'initialisation spécifiques à l'utilisateur.

/etc/profile– fichier d'initialisation du système

Il est lu en premier par le programme shell et permet d'accomplir plusieurs tâches parmi lesquelles :

Fixe les valeurs de certaines variables et les exporte : PATH, TERM, PS1, etc.

Affiche le contenu du banner /etc/motd (msg du jour)

Détermine les paramètres par défaut de création d'un fichier (permissions, umask, droits)

fichier d'initialisation spécifique à l'utilisateur

À la connexion de l'utilisateur, le shell cherche les fichiers suivants :

~/.bash_profile

~/.bash_login

~/.profile

Au contraire s'il s'agit d'un shell interactif (non de login utilisateur), il lit le fichier :

~/.bashrc

Autres personnalisations

PS1 – prompt

PATH – Le PATH est une variable système qui indique où sont les programmes exécutables sur votre ordinateur :

Si la commandeecho $PATH donne en sortie : /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games

Alors, après la commande : export PATH=$PATH:$HOME/bin

la commandeecho $PATH donnera en sortie : /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/home/user1/bin

Ceci avec /home/user1 comme répertoire home du user ayant tapé les commandes.

Options du shell

noclobber

ignoreeof

options d'édition en ligne de commande (vi or emacs) : voir le fichier .bashrc pour exemple.

II.Les variables shell

Comme dans tous les langages de programmation, on trouve en bash ce que l'on appelle des variables. Elles nous permettent de stocker temporairement des informations en mémoire. C'est en fait la base de la programmation.Les variables en bash sont assez particulières. Il faut être très rigoureux lorsqu'on les utilise.

Pour appeler une variable, par exemple HOME, il suffit de mettre un $ devant, par exemple :

echo $HOME

Ce petit code va afficher la valeur de la variable HOME à l'écran. On peut entrer par exemple la commande ls -al$HOME

Pour affecter une valeur à une variable c'est très simple :

MSG=salut

echo $MSG

HOME=/etc

PATH=/usr/bin:/usr/etc:/sbin

NEWVAR=»avu avu avu»

Variables d'environnement

Actuellement, les variables que vous créez dans vos scripts bash n'existent que dans ces scripts. En clair, une variable définie dans un programme A ne sera pas utilisable dans un programme B.

Les variables d'environnement sont des variables que l'on peut utiliser dans n'importe quel programme. On parle aussi parfois de variables globales. Vous pouvez afficher toutes celles que vous avez actuellement en mémoire avec la commande env :



Il y en a beaucoup. Certaines sont très utiles, d'autres moins, on trouve :

- **SHELL** : indique quel type de shell est en cours d'utilisation (sh, bash, ksh…) ;
- **PATH** : une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire. Nous en avons parlé un peu plus tôt. Si un programme se trouve dans un de ces dossiers, vous pourrez l'invoquer quel que soit le dossier dans lequel vous vous trouvez ;
- **EDITOR** : l›éditeur de texte par défaut qui s›ouvre lorsque cela est nécessaire ;
- **HOME** : la position de votre dossier home ;
- **PWD** : le dossier dans lequel vous vous trouvez ;
- OLDPWD : le dossier dans lequel vous vous trouviez auparavant.

Notez que les noms de ces variables sont, par convention, écrits en majuscules.Comment utiliser ces variables dans vos scripts ? C'est très simple, il suffit de les appeler par leur nom.

Exemple : le script ci-dessous affiche simplement : Votre éditeur par défaut est nano

#!/bin/bash

echo «Votre éditeur par défaut est $EDITOR»

Plus rarement, vous pourriez avoir besoin de définir votre propre variable d'environnement. Pour cela, on utilise la commande export (voir dans le fichier .bashrc).

Il existe des variables un peu spéciales :

Nom	Fonction
$*	contient tous les arguments passés à la fonction
$#	contient le nombre d'argument
$?	contient le code de retour de la dernière opération
$0	contient le nom du script
$n	contient l'argument n, n étant un nombre
$!	contient le PID de la dernière commande lancée

Pour déclarer un tableau, plusieurs méthodes :

```
tab=(«John Smith» «Jane Doe»)
```

ou bien :

```
tab[0]=John Smith
tab[1]=Jane Doe
```

Pour compter le nombre d'éléments du tableau :

```
len=${#tab[*]}
```

Pour afficher un élément :

```
echo ${tab[1]}
```

Pour afficher tous les éléments :

```
echo ${tab[@]}
```

Les expressions conditionnelles

I.La commande test

La commande test existe sous tous les Unix, elle permet de faire un test et de renvoyer 0 si tout s'est bien passé ou 1 en cas d'erreur.En mode console, faites man test pour connaître tous les opérateurs, en voici quelques-uns :

Opérateurs de test :

Syntaxe	Fonction réalisée
-e fichier	renvoie 0 si fichier existe
-d fichier	renvoie 0 si fichier existe et est un répertoire.
-f fichier	renvoie 0 si fichier existe et est un fichier 'normal'.
-w fichier	renvoie 0 si fichier existe et est en écriture.
-x fichier	renvoie 0 si fichier existe et est exécutable.
f1 -nt f2	renvoie 0 si f1 est plus récent que f2.
f1 -ot f2	renvoie 0 si f1 est plus vieux que f2.

Opérateurs de comparaison numériques :

Syntaxe	Fonction réalisée
$A -lt 5	renvoie 0 si $A est strictement inférieur à 5
$A -le 5	renvoie 0 si $A est inférieur ou égal à 5
$A -gt 5	renvoie 0 si $A est strictement supérieur à 5
$A -ge 5	renvoie 0 si $A est supérieur ou égal à 5
$A -eq 5	renvoie 0 si $A est égal à 5
$A -ne 5	renvoie 0 si $A est différent de 5

Les crochets

On peut raccourcir la commande test par des crochets. Exemple :

```
test -f /etc/passwd

echo $?

0

[ -f /etc/passwd ]

echo $?

0
```

Affichera la valeur 0 : ce fichier existe, 1 dans le cas où le fichier /etc/passwd n'existe pas. Sous unix, le code de retour est par convention et en général 0 s'il n'y a aucune erreur et différent de 0 dans les autres cas.

La syntaxe la plus appropriée dans de la programmation shell moderne est le double crochet :

```
[[ -f /etc/passwd ]]
```

Cela gère bien mieux les problèmes d'espace dans les noms de fichiers, les erreurs etc… C'est une structure propre à bash (ksh, ?) qui est le shell par défaut dans la plupart des distributions Linux. On garde en général des simples crochets pour les scripts shell qui doivent être à tout prix POSIX. (Utilisation sur des unix sans installation préalables de bash, comme BSD, Solaris...)

Les opérateurs logiques

Il y a en 3 :

- le et logique : -a
- le ou logique : -o
- le non logique : !

Exemple :

```
echo «renverra 0 si les deux expressions sont vraies»

test expr1 -a expr2

[ expr1 -a expr2 ]
```

II.Les structures de contrôle

La structure : `if`

Avant de commencer à faire des scripts de 1000 lignes, il serait intéressant de voir comment se servir des variables, et des instructions if, then, elif, else, fi. Cela permet par exemple de faire réagir le script de manière différente, selon la réponse de l'utilisateur à une question.En bash, les variables ne se déclarent généralement pas avant leur utilisation, on les utilise directement et elles sont créées lors de sa première mise en œuvre. Comme on l'a vu plus haut pour pouvoir voir la valeur d'une variable il faut faire précéder son nom du caractère « $ ».

Exemple 1 :

```
#!/bin/sh

echo -n «Entrez un nom de fichier: «

read file

if [ -e «$file» ]; then

        echo «Le fichier existe!»

else

        echo «Le fichier n'existe pas, du moins n'est pas dans le
répertoire d'exécution du script» fi

exit 0
```

Exemple 2 :

```
#!/bin/sh

echo -n «Voulez-vous voir la liste des fichiers Y/N : «

read ouinon

if [ «$ouinon» = «y» ] || [ «$ouinon» = «Y» ]; then

    echo «Liste des fichiers :»

    ls -la

elif [ «$ouinon» = «n» ] || [ «$ouinon» = «N» ]; then

    echo «Ok, bye! «

else

    echo «Il faut taper Y ou N!! Pas $ouinon»

fi
```

Question : que font ces programmes shell décrits aux exemples 1 et 2 précédents ?

Les structures while et until

La commande while exécute ce qu'il y a dans son bloc tant que la condition est respectée :

Exemple :

```
#!/bin/sh

cmpt=1

cm=3

echo -n «Mot de passe : «

read mdp

while [ «$mdp» != «ubuntu» ] && [ «$cmpt» != 4 ]

do

     echo -n «Mauvais mot de passe, plus que «$cm» chance(s): «

     read mdp

     cmpt=$(($cmpt+1))

     cm=$(($cm-1))

done

echo «Non mais, le brute-force est interdit!!»

exit 0
```

On retrouve des choses déjà abordées avec `if`. Le `&&` sert à symboliser un «et», cela implique que deux conditions sont à respecter. Le `do` sert à exécuter ce qui suit si la condition est respectée. Si elle ne l'est pas, cela saute tout le bloc (jusqu'à `done`).

Question : que fait ce programme shell ?

La structure `while` permet de faire exécuter la portion de code un nombre indéterminé de fois. La commande `until` fait la même chose que la commande `while` mais en inversant. C'est-à-dire qu'elle exécute le bloc jusqu'à ce que la condition soit vraie, donc elle s'emploie exactement comme la commande `while` : par exemple, si on a besoin d'attendre le démarrage de notre window manager pour exécuter des commandes dans notre Xsession il sera plus intéressant d'utiliser le `until`:

```
#!/bin/sh

until pidof wmaker

do sleep 1

done

xmessage «Session loaded» -buttons «Continue»:0,»That all»:1;

[ $? -eq 0 ] && xmessage «Load more...»
```

Mais on aurait pu aussi faire:

```
#!/bin/sh
while [ -z $(pidof wmaker) ]
do
 sleep 1
done
#(...)
```

La structure case

Regardons la syntaxe de cette commande, qui n'est pas une des plus simples :

case variable in

```
    modèle [ | modèle] ...) instructions;;
     modèle [ | modèle] ...) instructions;;
       ...
esac
```

Cela peut paraître complexe mais on s'y habitue quand on l'utilise.Mais à quoi sert cette commande ?

Elle sert à comparer le contenu d'une variable à des modèles différents. Les ;; sont indipensables car il est possible de placer plusieurs instructions entre un modèle et le suivant. Les ;; servent donc à identifier clairement la fin d'une instruction et le début du modèle suivant.

Exemple :

```
#!/bin/sh
echo -n «Etes-vous fatigué ? «
read on

case «$on» in
    oui | o | O | Oui | OUI ) echo «Allez prendre un café !»;;
    non | n | N | Non | NON ) echo «Programmez encore !»;;
    * ) echo «Ah bon ?»;;
esac
exit 0
```

La seule chose qui mérite vraiment d'être expliquée est sans doute `* )`. Cela indique tout simplement l'action à exécuter si la réponse donnée n'est aucune de celles données précédemment.

La structure for

L'instruction `for` exécute ce qui est dans son bloc un nombre de fois prédéfini. Sa syntaxe est la suivante :

```
for variable in valeurs; do
    instructions
done
```

Comme vous l'aurez sans doute remarqué, on assigne une valeur différente à variable à chaque itération. On peut aussi très facilement utiliser des fichiers comme «valeur». Rien ne vaut un exemple :

```
#!/bin/sh
for var in *.txt; do
    echo «$var»
done
exit 0
```

On peut voir une syntaxe un peu particulière :

```
$(sort *.txt)
```

Ceci sert à indiquer que ce qui est entre les parenthèses est une commande à exécuter.

On peut aussi utiliser cette instruction simplement avec des nombres, cela permet de connaître le nombre d'itérations :

```
#!/bin/sh
for var in 1 2 3 4 5 6 7 8 9; do
    echo $var
done
exit 0
```

On peut très bien aussi utiliser d'autres types de variables, comme par exemple des chaînes de caractères :

```
#!/bin/sh
for var in Ubuntu Breezy 5.10; do
    echo $var
done
exit 0
```

Il faut quand même faire attention au fait que Ubuntu Breezy 5.10 est différent de «Ubuntu Breezy 5.10» dans ce cas. En effet, tous les mots placés entre «» sont considérés comme faisant partie de la même chaîne de caractères. Sans les «», sh considèrera qu'il y a une liste de trois chaînes de caractères.

Les fonctions

Les fonctions sont indispensables pour bien structurer un programme mais aussi pouvoir le simplifier, créer une tâche, la rappeler… Voici la syntaxe générale de 'déclaration' d'une fonction :

```
nom_fonction(){
les_instructions
}
```

Cette partie ne fait rien en elle même, elle dit juste que quand on appellera nom_fonction, elle fera les_instructions. Pour appeler une fonction (qui ne possède pas d'argument, voir plus loin) rien de plus simple :

nom_fonction

Rien ne vaut un petit exemple :

```
#!/bin/sh!
#Definition de ma fonction
mafonction(){
  echo 'La liste des fichiers de ce répertoire'
  ls -l
}
#fin de la définition de ma fonction
echo 'Vous allez voir la liste des fichiers de ce répertoire:'
mafonction       #appel de ma fonction
exit 0
```

Comme vous l'avez sans doute remarqué, quand on appelle la fonction, on exécute simplement ce qu'on lui a défini au début, dans notre exemple, echo… et ls -l, on peut donc faire exécuter n'importe quoi à une fonction.

Les fonctions peuvent être définies n'importe où dans le code du moment qu'elle sont définies avant d'être utilisées. Même si en bash les variables sont globales, il est possible de les déclarer comme locales au sein d'une fonction en la précédant du mot clé local: local ma_fonction

Exemple: Un sleep interactif.

```
#!/bin/bash

function info(){

        echo -e «$1\nBye»

        exit

}

test -z «$1» && info «requiert 1 argument pour le temps
d'attente...» || PRINT=$(($1*500))

test -z $(echo «$1» | grep -e «^[0-9]*$») && info «'$1' est un
mauvais argument»

test $1 -gt 0 || info «Je ne prends que les entiers > 0»

function print_until_sleep(){

         local COUNT=0

         while [ -d /proc/$1 ]; do

                  test $(($COUNT%$2)) -eq 0 && echo -n «*»

                  COUNT=$(($COUNT+1))

         done

}

sleep $1 & print_until_sleep $! $PRINT

echo -e «\nBye»
```

Activité 2: Code assembleur en ligne

Introduction

Aujourd'hui, peu de programmeurs utilisent le langage assembleur. Des langagesde plus haut niveau comme le C ou le C++ s'exécutent sur quasiment toutes les architectures et permettent une productivité plus importante lors de l'écriture et de la maintenancedu code. Parfois, les programmeurs ont besoin d'utiliser des instructions assembleur dans leursprogrammes, la GNU Compiler Collection permet aux programmeurs d'ajouter des instructionsen langage assembleur dépendantes de l'architecture à leurs programmes.

Les instructions assembleur GCC en ligne ne doivent pas être utilisées de façon excessive. Lesinstructions en langage assembleur dépendant de l'architecture, aussi, des programmes utilisantdes instructions x86 ne peuvent pas être compilés sur des PowerPC. Pour les utiliser, vous aurezbesoin d'un dispositif permettant de les traduire dans le jeu d'instruction de votre architecture.Cependant, les instructions assembleur vous permettent d'accéder au matériel directement etpeuvent permettre de produire du code plus performant.

L'instruction asm vous permet d'insérer des instructions assembleur dans des programmes Cou C++. Par exemple, l'instruction :

```
asm (« fsin « : «=t» ( answer ) : «0» ( angle ));
```

est une façon spécifique à l'architecture x86 de coder cette instruction C1 :

```
answer = sin ( angle );
```

Notez que contrairement aux instructions assembleur classiques, les constructions asm vouspermettent de spécifier des opérandes en utilisant la syntaxe du C.Pour en savoir plus sur le jeu d'instructions x86, que nous utiliserons dans ce chapitre, consultezhttp://developer.intel.com/design/pentiumii/manuals/ et http://www.x86-64.org/documentation.

Quand Utiliser du Code Assembleur

Bien qu'il ne faille pas abuser des constructions asm, elles permettent à vos programmesd'accéder au matériel directement et peuvent produire des programmes qui s'exécutent rapidement.

Vous pouvez les utiliser lors de l'écriture du code faisant partie du système d'exploitationqui a besoin d'interagir avec le matériel. Par exemple, /usr/include/asm/io.h contient desinstructions assembleur pour accéder aux ports d'entrée/sortie directement. Le fichier sourcedu noyau situé dans /usr/src/linux/arch/i386/kernel/process.s offre un autre exempleen utilisant hlt dans une boucle d'inactivité. Consultez d'autres fichiers source du noyau Linuxsitués dans /usr/src/linux/arch/ et /usr/src/linux/drivers/.

Les instructions assembleur peuvent également accélérer la boucle de traitement interne decertains programmes. Par exemple, si la majorité du temps d'exécution d'un programme estconsacré au calcul du sinus et du cosinus du même angle, il est possible d'utiliser l'instructionx86 fsincos2. Consultez par exemple /usr/include/bits/mathinline.h qui encapsuledes séquences assembleur au sein de macros afin d'accélérer le calcul de certaines fonctionstransversales.

Vous ne devriez utiliser des instructions assembleur en ligne pour accélérer l'exécution qu'endernier ressort. Les compilateurs actuels sont sophistiqués et connaissent les détails des processeurspour lesquels ils génèrent du code. Ainsi, ils peuvent souvent sélectionner des séquences decode qui paraissent contre-intuitives mais qui s'exécutent en fait plus vite que d'autres. À moinsque vous ne compreniez le jeu d'instructions et les propriétés d'ordonnancement du processeurcible dans les détails, vous feriez sans doute mieux de laisser les optimiseurs du compilateurgénérer du code assembleur à votre place pour la plupart des opérations.

De temps à autre, une ou deux instructions assembleur peuvent remplacer plusieurs lignesde code d'un langage de plus haut niveau. Par exemple, déterminer la position du bit non nulle plus significatif d'un entier en utilisant le C nécessite une boucle ou des calculs en virguleflottante. Beaucoup d'architecture, y compris le x86 disposent d'une instruction assembleur (bsr)qui calcule cette position.

Assembleur en Ligne Simple

Nous allons maintenant présenter la syntaxe des instructions assembleur asm avec un exempledécalant une valeur de 8 bits vers la droite sur architecture x86 :

```
asm (« shrl $8 , %0» : «=r» ( answer ) : «r» ( operand ) : «cc»);
```

Le mot-clef asm est suivi par une expression entre parenthèses constituée de sections séparéespar deux-points. La première section contient une instruction assembleur et ses opérandes, danscet exemple, shrl décale à droite les bits du premier opérande. Celui-ci est représenté par %0, lesecond est la constante immédiate $8.

La deuxième section indique les sorties. Ici, la première sortie de l'instruction sera placée dansla variable C answer, qui doit être une lvalue. La chaîne «=r» contient un signe égal indiquantun opérande de sortie et un r indiquant que answer est stocké dans un registre.

Convertir un asm en Instructions Assembleur

Le traitement des constructions asm par GCC est très simple. Il produit des instructionsassembleur pour traiter les opérandes de l'asm et replace la construction asm par l'instructionque vous spécifiez. Il n'analyse pas du tout l'instruction.

Par exemple, GCC convertit cet extrait de code :

```
double foo , bar ;

asm (« mycool_asm %1, %0» : «=r» ( bar ) : «r» ( foo ));

en la séquence d'instructions x86 suivante :

movl -8(% ebp ) ,% edx

movl -4(% ebp ) ,% ecx

# APP

mycool_asm %edx , % edx

# NO_APP

movl %edx , -16(% ebp )

movl %ecx , -12(% ebp )
```

Souvenez-vous que foo et bar requièrent chacun deux mots d'espace de stockage dans la pile surune architecture x86 32 bits. Le registre ebp pointe vers les données sur la pile.Les deux premières instructions copient foo dans les registres EDX et ECX qu'utilise mycool_asm. Le compilateur a décidé d'utiliser les mêmes registres pour stocker la réponse, qui est copiéedans bar par les deux dernières instructions. Il choisit les registres adéquats, peut même lesréutiliser, et copie les opérandes à partir et vers les emplacements appropriés automatiquement.

Syntaxe Assembleur Avancée

Dans les sous-sections qui suivent, nous décrivons les règles de syntaxe pour les constructionsasm. Leurs sections sont séparées par deux-points.Nous utiliserons l'instruction asm suivante qui calcule l'expression booléenne x > y :

```
asm (« fucomip %% st (1) , %% st; seta %% al» :

«=a» ( result ) : «u» (y), «t» (x) : «cc», «st»);
```

Tout d'abord, fucomip compare ses deux opérandes x et y et stocke les valeurs indiquant lerésultat dans le registre de code condition. Puis, seta convertit ces valeurs en 0 ou 1.

Instructions Assembleur

La première section contient les instructions assembleur, entre guillemets. La constructionasm exemple contient deux instructions assembleur, fucomip et seta, séparées par un point-virgule.Si l'assembleur n'autorise pas les points-virgules, utilisez des caractères de nouvelle ligne(\n) pour séparer les instructions. Le compilateur ignore le contenu de cette première section,excepté qu'il supprime un niveau de signes pourcent, %% devient donc %. La signification de%%st(1) et d'autres termes similaires dépend de l'architecture.

GCC se plaindra si vous spécifiez l'option -traditional ou -ansi lors de la compilationd'un programme contenant des constructions asm. Pour éviter de telles erreurs, utilisez le motclé alternatif __asm__ comme dans les fichiers d'en-tête cités précédemment.

Sorties

La seconde section spécifie les opérandes de sortie des instructions en utilisant une syntaxe C.Chaque opérande est décrit par une contrainte d'opérande sous forme de chaîne suivie d'uneexpression C entre parenthèses. Pour les opérandes de sortie, qui doivent être des lvalues, lachaîne de contrainte doit commencer par un signe égal. Le compilateur vérifie que l'expression Cpour chaque opérande de sortie est effectivement une lvalue.

Les lettres spécifiant les registres pour une architecture donnée peuvent être trouvée dans lecode source de GCC, dans la macro REG_CLASS_FROM_LETTER. Par exemple, le fichier de configurationgcc/con ?g/i386/i386.h de GCC liste les lettres de registre pour l'architecture x863. LeTableau 9.1 en fait le résumé.

Tab. 9.1 – Lettres correspondant aux Registres sur l'Architecture Intel x86

Lettre	Registres Éventuellement Utilisés par GCC
R	Registre général (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP)
q	Registre de données général (EAX, EBX, ECX, EDX)
f	Registre virgule flottante
t	Registre en virgule flottante supérieur
u	Second registre en virgule flottante
a	Registre EAX
b	Registre EBX
c	Registre ECX
d	Registre EDX
x	Registre SSE (Streaming SIMD Extension)
y	Registres multimédias MMX
A	Valeur de 8 octets formée à partir de EAX et EDX
D	Pointeur de destination pour les opérations sur les chaînes (EDI)
S	Pointeur source pour les opérations sur les chaînes (ESI)

Lorsqu'une structure asm utilise plusieurs opérandes, elles doivent être décrites par une chaînede contrainte et une expressions C et séparées par des virgules, comme l'illustre l'exemple donnéprécédemment. Vous pouvez spécifier jusqu'à 10 opérandes, numérotées de %0 à %9 dans lessections d'entrée et de sortie. S'il n'y a pas d'opérandes de sortie ou de registre affectés, laissezla section de sortie vide ou signalez la avec un commentaire du type /* no outputs */.

Entrées

La troisième section décrit les entrées des instructions assembleur. La chaîne de contrainted'un opérande d'entrée ne doit pas contenir de signe égal, ce qui indique une lvalue. Mis à partcela, leur syntaxe est identique à celle des opérandes de sortie.

Pour indiquer qu'un registre est à la fois lu et écrit par la même construction asm, utilisezl'indice de l'opérande de sortie comme chaîne de contrainte d'entrée. Par exemple, pour indiquerqu'un registre d'entrée est le même que le premier registre de sortie, utilisez 0. Spécifier la mêmeexpression C pour des opérandes d'entrée et de sortie ne garantit pas que les deux valeurs serontplacées dans le même registre.

La section d'entrée peut être omise s'il n'y a pas d'opérandes d'entrée et que la section dedéclaration des modifications est vide.

Déclaration des Modifications

Si une instruction modifie les valeurs d'un ou plusieurs registres par effet de bord, spécifiezces registres dans la quatrième section de la structure asm. Par exemple, l'instruction fucomipmodifie le registre de code condition, qui est désigné par cc. Les registres modifiés sont décritsdans des chaînes individuelles séparées par des virgules. Si l'instruction est susceptible demodifier un emplacement mémoire arbitraire, spécifiez memory. En utilisant les informationssur la modification des registres, le compilateur détermine les valeurs qui doivent être restauréesaprès l'exécution du bloc asm. Si vous ne renseignez pas ces informations correctement, GCCpourrait supposer que certains registres contiennent des valeurs qui ont en fait été écrasées, cequi pourrait affecter le fonctionnement de votre programme.

Exemple

L'architecture x86 dispose d'instructions qui déterminent les positions du bit non nul le plussignificatif ou le moins significatif dans un mot. Le processeur peut exécuter ces instructions defaçon relativement efficace. Par contre, l'implémentation de la même opération en C nécessiteune boucle et un décalage binaire.

Par exemple, l'instruction assembleur bsrl calcule la position du bit le plus significatif de sonpremier opérande et place cette position (à partir de 0 qui représente le bit le moins significatif)dans le second. Pour placer la position du bit non nul le plus significatif de number dans position,nous pourrions utiliser cette structure asm :

```
asm (« bsrl %1, %0» : «=r» ( position ) : «r» ( number ));
```

Une façon d'implémenter la même opération en C, est d'utiliser une boucle de ce genre :

```
long i;

for (i = ( number >> 1) , position = 0; i != 0; ++ position )

i >>= 1;
```

Pour comparer les vitesses de ces deux versions, nous allons les placer dans une bouclequi calcule les positions pour de grands nombres. Le Listing 9.1 utilise l'implémentation en C.

Le programme boucle sur des entiers, en partant de 1 jusqu'à la valeur passée sur la ligne decommande. Pour chaque valeur de number, il calcule la position du bit non nul le plus significatif.

Le Listing 9.2 effectue les mêmes opérations en utilisant une construction assembleur en ligne.Notez que dans les deux versions, nous plaçons la position calculée à une variable volatileresult. Cela permet d'éviter que l'optimiseur du compilateur ne supprime le calcul ; si le résultatn'est pas utilisé ou stocké en mémoire, l'optimiseur élimine le calcul en le considérant commedu code mort.

Listing 9.1 – (bit-pos-loop.c) – Recherche d'un Bit en Utilisant une Boucle

```
1 # include <stdio .h>

2 # include <stdlib .h>

3

4 int main ( int argc , char * argv [])

5 {

6 long max = atoi ( argv [1]) ;

7 long number ;

8 long i;

9 unsigned position ;

10 volatile unsigned result ;

11

12 /* Répète l'opération pour un nombre important de valeurs . */

13 for ( number = 1; number <= max ; ++ number ) {

14 /* Décale le nombre vers la droite jusqu 'à ce qu 'il vaille

15 zéro . Mémorise le nombre de décalage qu 'il a fallu faire .
*/

16 for (i = ( number >> 1) , position = 0; i != 0; ++ position )
```

```
17 i >>= 1;
18 /* La position du nombre non nul le plus significatif est le
nombre
19 de décalages qu 'il a fallu après le premier . */
20 result = position ;
21 }
22
23 return 0;
24 }
```

Listing 9.2 – (bit-pos-asm.c) – Recherche d'un Bit en Utilisant bsrl

```
1 # include <stdio .h>
2 # include <stdlib .h>
3 int main ( int argc , char * argv [])
4 {
5 long max = atoi ( argv [1]) ;
6 long number ;
7 unsigned position ;
8 volatile unsigned result ;
9 /* Repeat the operation for a large number of values . */
10 for ( number = 1; number <= max ; ++ number ) {
11 /* Compute the position of the most signi ? cant set bit using
the
12 bsrl assembly instruction . */
13 asm (« bsrl %1, %0» : «=r» ( position ) : «r» ( number ));
14 result = position ;
15 }
16 return 0;
17 }
```

Compilons les deux versions avec les optimisations actives :

```
% cc -O2 -o bit-pos-loop bit-pos-loop.c
% cc -O2 -o bit-pos-asm bit-pos-asm.c
```

À présent, lançons-les en utilisant la commande time pour mesurer le temps d'exécution. Il estnécessaire de passer une valeur importante comme argument afin de s'assurer que chaque versionmette un minimum de temps à s'exécuter.

```
% time ./bit-pos-loop 250000000

real 0m27.042s

user 0m24.583s

sys 0m0.135s

% time ./bit-pos-asm 250000000

real 0m1.472s

user 0m1.442s

sys 0m0.013s
```

Voyez comme la version utilisant l'assembleur s'exécute beaucoup plus vite (vos propres résultatspeuvent varier).

Problèmes d'Optimisation

L'optimiseur de GCC tente de réordonner et de réécrire le code du programme pour minimiserle temps d'exécution même en présence d'expressions asm. Si l'optimiseur détermine que lesvaleurs de sortie ne sont pas utilisées, l'instruction sera supprimée à moins que le mot clévolatile ne figure entre asm et ses arguments (par ailleurs, GCC ne déplacera pas une structureasm sans opérande de sortie en dehors d'une boucle). Toute structure asm peut être déplacée defaçon difficile à prévoir, même d'un saut à l'autre. La seule façon de garantir l'ordre d'un blocassembleur est d'inclure toutes les instructions dans la même structure asm.

L'utilisation de constructions asm peut limiter l'efficacité de l'optimiseur car le compilateurne connaît pas leur sémantique. GCC est forcé de faire des suppositions qui peuvent interdirecertaines optimisations.

Problèmes de Maintenance et de Portabilité

Si vous décidez d'utiliser des constructions asm non-portables et dépendantes de l'architecture,les encapsuler dans des macros peut aider à la maintenance et améliorer la portabilité.

Placer toutes ces macros dans un fichier et les documenter facilitera le portage de l'applicationvers une autre architecture, quelque chose qui arrive souvent même pour les programmes « àla va vite ». De cette façon, le programmeur n'aura besoin de réécrire qu'un seul fichier pouradapter le logiciel à une autre architecture.

Par exemple, la plupart des instructions asm du code source Linux sont regroupées dansles fichiers d'en-tête situés sous /usr/src/linux/include/asm/ et /usr/src/linux/include/asm-i386/ et les fichiers source situés sous /usr/src/linux/arch/i386/ et /usr/src/linux/drivers/.

Exercices divers

1. Ecrire les programmes shell suivants, les sauvegarder, les exécuter et bien analyser les résultats.

```
# Script to print user information who currently login ,
current date & time

#

clear

echo «Hello $USER»

echo «Today is \c «;date

echo «Number of user login : \c» ; who | wc l

echo «Calendar»

cal

exit 0
```

2. Essayer les commandes suivantes et noter le statut du code retour :

```
$ expr 1 + 3

$ echo $?

$ echo Welcome

$ echo $?

$ wildwest canwork?

$ echo $?

$ date

$ echo $?

$ echon $?

$ echo $?
```

3. i vous souhaitez afficher votre home directory quelle est la commande que vous allez taper

```
$ echo $HOME or

$ echo HOME
```

Laquelle des commandes ci-dessus est correcte ? Pourquoi ?

4. Expliquez brièvement comment accomplir chacune de ces opérations :

   a.Définirune variable x avec comme valeur 10 et l'afficher à l'écran ?

   b.Définirune variable xnavec comme valeur Rani et l'afficher à l'écran ?

   c.afficher la sommede deux nombres entiers, par exemple 6 et 3 ?

   d.Définirdeux variablesx=20, y=5 et afficher la division de x par y (i.e. x/y) ?

   e.Modifier dans (d) pour stocker le résultat de la division de x par y dans unevariable z.

5. Répérer s'il y a lieu les erreurs contenues dans le script ci-dessous :

```
# Script to test MY knowledge about variables!

#

myname=Vivek

myos = TroubleOS

myno=5

echo «My name is $myname»

echo «My os is $myos»

echo «My number is myno, can you see this number»
```

6. Ecrire un script, pad, qui fait un décalage à gauche. Compter directement de 2 à 10.

7. Ecrire et exécuter le programme assembleur ci-dessous et analyser le résultat.

```
#include <stdio.h>

int main() {

int arg1, arg2, add, sub, mul, quo, rem ;

printf( «Enter two integer numbers : « );

scanf( «%d%d», &arg1, &arg2 );

/* Perform Addition, Subtraction, Multiplication &
Division */

  __asm__ ( «addl %%ebx, %%eax;» : «=a» (add) : «a» (arg1)
, «b» (arg2) );

  __asm__ ( «subl %%ebx, %%eax;» : «=a» (sub) : «a» (arg1)
, «b» (arg2) );
```

```
  __asm__ ( «imull %%ebx, %%eax;» : «=a» (mul) : «a» (arg1)
, «b» (arg2) );

  __asm__ ( «movl $0x0, %%edx;»

«movl %2, %%eax;»

«movl %3, %%ebx;»

«idivl %%ebx;» : «=a» (quo), «=d» (rem) : «g» (arg1), «g»
(arg2) );

printf( «%d + %d = %d\n», arg1, arg2, add );

printf( «%d %

d = %d\n», arg1, arg2, sub );

printf( «%d * %d = %d\n», arg1, arg2, mul );

printf( «%d / %d = %d\n», arg1, arg2, quo );

printf( «%d %% %d = %d\n», arg1, arg2, rem );

return 0 ;

}
```

8. Ecrire et exécuter le programme assembleur ci-dessous et analyser le résultat.

```
#include <stdio.h>

int gcd( int a, int b ) {

int result ;

/* Compute Greatest Common Divisor using Euclid's
Algorithm */

  __asm__ __volatile__ ( «movl %1, %%eax;»

«movl %2, %%ebx;»

«CONTD: cmpl $0, %%ebx;»

«je DONE;»

«xorl %%edx, %%edx;»

«idivl %%ebx;»

«movl %%ebx, %%eax;»

«movl %%edx, %%ebx;»

«jmp CONTD;»

«DONE: movl %%eax, %0;» : «=g» (result) : «g» (a), «g» (b)
```

```
);

return result ;

}

int main() {

int first, second ;

printf( «Enter two integers : « ) ;

scanf( «%d%d», &first, &second );

printf( «GCD of %d & %d is %d\n», first, second, gcd(first,
second) ) ;

return 0 ;

}
```

9. Réécrire en langage C, les programmes 7. Et 8. , les exécuter et les comparer par rapport aux programmes assembleurs en termes de rapidité d'exécution.

Conclusion

Cette unité a présenté trois aspects de la programmation d'un système informatique, à savoir, en utilisant le shell, un langage de haut niveau tel que le C / C ++, et le langage d'assemblage. Plus précisément, leshell Unix / Linuxa été mis en place le long du processus d'écriture et les constructions d'un programme shell. Nous avons présenté les shells couramment utilisés et disponibles sous Linux / Unix, et expliqué le processusd'écriture des scripts shell. La possibilité de travailler avec des variables dans des scripts shell a aussi été préentée. Enfin, les techniques de code assembleur en ligne dans un programme de C / C ++ ont également été présentées.

Évaluation de l'unité

Vérifiez votre compréhension!

Instructions

Cette évaluation doit être fait en individuel

Il s'agit d'un devoir à rendre

Lisez attentivement le cours et répondez aux questions suivantes

Exercices divers

1. Ecrire, sauvegarder, exécuter et analyser le résultat du script script suivant :

```
# Script to print user information who currently login ,
current date & time

#

clear

echo «Hello $USER»

echo «Today is \c «;date

echo «Number of user login : \c» ; who | wc l

echo «Calendar»

cal

exit 0
```

2. Entrer les commandes suivantes et noter le code retour :

a.
```
$ expr 1 + 3

$ echo $?
```

b.
```
$ echo Welcome

$ echo $?
```

c.
```
$ wildwest canwork?

$ echo $?
```

d.
```
$ date

$ echo $?
```

e. `$ echon $?`

```
$ echo $?
```

3. Si vous souhaitez afficher votre home directory, laquelle des commandes ci-dessous est correcte ?

   a. $ echo $HOME or
   b. $ echo HOME

4. Expliquer comment accomplir chacune des opérations ci-dessous :

   a. define variable x with value 10 and print it on screen?

   b. define variable xn with value Rani and print it on screen?

   c. print sum of two numbers, let's say 6 and 3?

   d. define two variable x=20, y=5 and then print division of x and y (i.e. x/y)?

   e. Modify in (d) above and store division of x and y to a variable called z.

5. Rechercher les erreurs, s'il y a lieu, du script ci-dessous :

```
# Script to test MY knowledge about variables!
#
myname=Vivek
myos = TroubleOS
myno=5
echo «My name is $myname»
echo «My os is $myos»
echo «My number is myno, can you see this number»
```

Lectures et autres ressources

- Learning the bash Shell: Unix Shell Programming (In a Nutshell (O'Reilly)), Kindle Edition, Cameron Newham (Author)
- bash Cookbook: Solutions and Examples for bash Users (Cookbooks (O'Reilly)), Kindle Edition, Carl Albing (Author), JP Vossen (Author), Cameron Newham (Author)
- Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001
- Professional Assembly Language, John Wiley & Sons, 22 Feb 2005, By Richard Blumhttp://www.codeproject.com/Articles/15971/UsingInlineAssemblyinCC

Unité 3. Processus, Threads et Gestion de la Mémoire

Introduction à l'unité

Une instance d'un programme en cours d'exécution est appelée un processus. Si vous avez deux fenêtres de terminal sur votre écran, vous exécutez probablement deux fois le même programme de terminal – vous avez deux processus de terminal. Chaque fenêtre de terminal exécute probablement un shell ; chaque shell en cours d'exécution est un processus indépendant. Lorsque vous invoquez un programme depuis le shell, le programme correspondant est exécuté dans un nouveau processus ; le processus shell reprend lorsque ce processus se termine.

Les programmeurs expérimentés utilisent souvent plusieurs processus coopérant entre eux au sein d'une même application afin de lui permettre de faire plus d'une chose à la fois, pour améliorer sa robustesse et utiliser des programmes déjà existants.

La plupart des fonctions de manipulation de processus décrites dans cette unité sont similaires à celles des autres systèmes UNIX. La majorité d'entre elles est déclarée au sein de l'entête <unistd.h> ; vérifiez la page de manuel de chaque fonction pour vous en assurer.

Les threads appelés aussi « processus légers », comme les processus,sont un mécanisme permettant à un programmede faire plus d'une chose à la fois. Comme les processus, les threads semblent s'exécuteren parallèle ; le noyau Linux les ordonnance de façon asynchrone, interrompant chaque threadde temps en temps pour donner aux autres une chance de s'exécuter.

Conceptuellement, un thread existe au sein d'un processus. Les threads sont une unité d'exécutionplus fine que les processus. Lorsque vous invoquez un programme, Linux crée un nouveauprocessus et, dans ce processus, crée un thread, qui exécute le processus de façon séquentielle.Ce thread peut en créer d'autres ; tous ces threads exécutent alors le même programme au seindu même processus, mais chaque thread peut exécuter une partie différente du programme à uninstant donné.

La mémoire est la zone principale de stockage de données pour les ordinateurs. Nous appelons l'unité de mémoire de base un bit. Un bit peut contenir deux valeurs différentes : 0 ou 1. Pourquoi les ordinateurs utilisent l'arithmétique binaire? Il est le moyen le plus fiable et efficace pour exprimer les données. Parce que les informations numériques peuvent être représentées par un niveau tension électrique. Nous pouvons distinguer chaque valeur différente. Si nous avons plus de valeurs (tels que le système décimal), il provoque moins de séparation entre les valeurs adjacentes. Mais avec deux valeurs (système binaire: 0et 1) des valeurs différentes seront distingués avec la distance maximale.

Dans cette unité nous aborderons les différents mécanismes de gestion des processus, des threads et de la mémoire.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Expliquer les concepts de gestion des processus, des threads et de la mémoire
- Créer des processus en utilisant les fonctions fork et exec du système
- Gérer et manipuler des processus utilisant plusieurs signaux systèmes
- Créer et gérer des threads
- Comparer et différencier les processus des threads
- Comprendre les techniques de gestion de la mémoire.
- Réaliser un laboratoire sur la manipulation de processus et interpréteur de commandes

Laboratoire #1 : Manipulation de processus et interpréteur de commandes

À la fin de cette unité, vous devriez réaliser un premier lab présenté en fin du document.

Termes clés

Processus: Une instance d'un programme en cours d'exécution est appelée un processus.

Thread: Les threads appelés aussi « processus légers », comme les processus,sont un mécanisme permettant à un programmede faire plus d'une chose à la fois.Les threads sont une unité d'exécutionplus fine que les processus.

Gestion de la mémoire: Gestion de la mémoire est l'acte de gestionde la mémoire de l'ordinateur au niveau du système. L'exigence essentielle de la gestion de la mémoireest de fournir des moyens d'allouer dynamiquement des partiesde la mémoire à des programmes, à leur demande, et la libérer pour une réutilisation lorsqu'elles ne sont plus nécessaires. Cela est essentiel pour tout système informatique de pointe où plusieurs processus pourraient être en cours à d'exécution à tout moment.

Activités d'apprentissage

Activité 1: Les Processus

Aperçu des Processus

Même lorsque tout ce que vous faites est être assis devant votre ordinateur, des processussont en cours d'exécution. Chaque programme utilise un ou plusieurs processus. Commençonspar observer les processus déjà présents sur votre ordinateur.

Identifiants de Processus

Chaque processus d'un système Linux est identifié par son identifiant de processus unique,quelquefois appelé pid (process ID). Les identifiants de processus sont des nombres de 16 bitsassignés de façon séquentielle par Linux aux processus nouvellement créés.

Chaque processus a également un processus parent (sauf le processus spécial init, décritdans la Section « Processus Zombies »). Vous pouvez donc vous représenter les processusd'un système Linux comme un arbre, le processus init étant la racine. L'identifiant de processusparent (parent process ID), ou ppid, est simplement l'identifiant du parent du processus.

Lorsque vous faites référence aux identifiants de processus au sein d'un programme C ouC++, utilisez toujours le typedef pid_t, défini dans <sys/types.h>. Un programme peut obtenirl'identifiant du processus au sein duquel il s'exécute en utilisant l'appel système getpid() etl'identifiant de son processus parent avec l'appel système getppid(). Par exemple, le programmedu Listing 3.1 affiche son identifiant de processus ainsi que celui de son parent :

Listing 3.1 – (print-pid.c) – Afficher l'Identifiant de Processus

```
1 # include <stdio .h>
2 # include <unistd .h>
3
4 int main ()
5 {
6 printf («L' identifiant du processus est %d\n», ( int )
getpid ());
7 printf («L' identifiant du processus parent est %d\n», (
int ) getppid ());
8 return 0;
9 }
```

Notez que si vous invoquez ce programme plusieurs fois, un identifiant de processus différentest indiqué à chaque fois car chaque invocation crée un nouveau processus. Cependant, sivous l'invoquez toujours depuis le même shell, l'identifiant du processus parent (c'est-à-direl'identifiant de processus du shell) est le même.

Voir les Processus Actifs

La commande ps affiche les processus en cours d'exécution sur votre système. La versionGNU/Linux de ps dispose d'un grand nombre d'options car elle tente d'être compatible avec lesversions de ps de plusieurs autres variantes d'UNIX. Ces options contrôlent les processus listéset les informations les concernant qui sont affichées.Par défaut, invoquer ps affiche les processus contrôlés par le terminal ou la fenêtre de terminalla commande est invoquée. Par exemple :

```
% ps

PID TTY TIME CMD

21693 pts/8 00:00:00 bash

21694 pts/8 00:00:00 ps
```

Cette invocation de ps montre deux processus. Le premier, bash, est le shell s'exécutant ausein du terminal. Le second est l'instance de ps en cours d'exécution. La première colonne, PID,indique l'identifiant de chacun des processus.

Pour un aperçu plus détaillé des programmes en cours d'exécution sur votre système GNU/-Linux, invoquez cette commande :

```
% ps -e -o pid,ppid,command
```

L'option -e demande à ps d'afficher tous processus en cours d'exécution sur le système. L'option-o pid,ppid,command indique à ps les informations à afficher sur chaque processus – dans cecas, l'identifiant de processus, l'identifiant du processus parent et la commande correspondantau processus.

Voici les premières et dernières lignes de la sortie de cette commande sur mon système. Ellespeuvent différer des vôtres, selon les programmes en cours d'exécution sur votre système :

Formats de Sortie ps

Avec l'option -o de la commande ps, vous indiquez les informations sur les processus que vousvoulez voir s'afficher sous forme d'une liste de valeurs séparées par des virgules. Par exemple, ps-o pid,user,start_time,command affiche l'identifiant de processus, le nom de l'utilisateurpropriétaire du processus, l'heure de démarrage du processus et la commande s'exécutant ausein du processus. Consultez la page de manuel de ps pour une liste complète des codes. Vouspouvez utiliser les options -f (full listing, listing complet), -l (long listing) ou -j (jobs listing)pour obtenir trois formats de listing prédéfinis.

```
% ps -e -o pid,ppid,command
PID PPID COMMAND
1 0 init [5]
2 1 [kflushd]
3 1 [kupdate]
...
21725 21693 xterm
21727 21725 bash
21728 21727 ps -e -o pid,ppid,command
```

Notez que l'identifiant du processus parent de la commande ps, 21727, est l'identifiant duprocessus de bash, le shell depuis lequel j'ai invoqué ps. L'identifiant du processus parent debash est 21725,l'identifiant du processus du programme xterm dans lequel le shell s'exécute

.

Tuer un Processus

Vous pouvez tuer un processus en cours d'exécution grâce à la commande kill. Spécifiezsimplement sur la ligne de commande l'identifiant du processus à tuer.La commande kill envoie un signal1 SIGTERM, ou de terminaison au processus. Cela terminele processus, à moins que le programme en cours d'exécution ne gère ou ne masque le signalSIGTERM. Les signaux sont décrits dans la section « Signaux ».

Créer des Processus

Deux techniques courantes sont utilisées pour créer de nouveaux processus. La premièreest relativement simple mais doit être utilisée avec modération car elle est peu performante etprésente des risques de sécurité considérables. La seconde technique est plus complexe mais offreune flexibilité, une rapidité et une sécurité plus grandes.

Utiliser la fonction system

La fonction system de la bibliothèque standard propose une manière simple d'exécuter unecommande depuis un programme, comme si la commande avait été tapée dans un shell. En fait,system crée un sous-processus dans lequel s'exécute le shell Bourne standard (/bin/sh) et passela commande à ce shell pour qu'il l'exécute. Par exemple, le programme du Listing 3.2 invoquela commande ls pour afficher le contenu du répertoire racine, comme si vous aviez saisi ls –l/ dans un shell :

Listing 3.2 – (system.c) – Utiliser la Fonction system

```
1 int main ()
2 {
3 int return_value ;
4 return_value = system («ls -l /»);
5 return return_value ;
6 }
```

La fonction system renvoie le code de sortie de la commande shell. Si le shell lui-même nepeut pas être lancé, system renvoie 127 ; si une autre erreur survient, system renvoie -1.

Comme la fonction system utilise un shell pour invoquer votre commande elle est soumiseaux fonctionnalités, limitations et failles de sécurité du shell système. Vous ne pouvez pas vousreposer sur la disponibilité d'une version spécifique du shell Bourne. Sur beaucoup de systèmesUNIX, /bin/sh est en fait un lien symbolique vers un autre shell. Par exemple, sur la plupart dessystèmes GNU/Linux, /bin/sh pointe vers bash (le Bourne-Again SHell) et des distributionsdifférentes de GNU/Linux utilisent différentes version de bash. Invoquer un programme avecles privilèges root via la fonction system peut donner des résultats différents selon le systèmeGNU/Linux. Il est donc préférable d'utiliser la méthode de fork et exec pour créer des processus.

Utiliser fork et exec

Les API DOS et Windows proposent la famille de fonctions spawn. Ces fonctions prennenten argument le nom du programme à exécuter et créent une nouvelle instance de processus pource programme. Linux ne dispose pas de fonction effectuant tout cela en une seule fois. Au lieude cela, Linux offre une fonction, fork, qui produit un processus fils qui est l'exacte copie deson processus parent. Linux fournit un autre jeu de fonctions, la famille exec, qui fait en sortequ'un processus cesse d'être une instance d'un certain programme et devienne une instance d'unautre. Pour créer un nouveau processus, vous utilisez tout d'abord fork pour créer une copie duprocessus courant. Puis vous utilisez exec pour transformer un de ces processus en une instancedu programme que vous voulez créer.

Appeler fork et exec

Lorsqu'un programme appelle fork, une copie du processus, appelée processus fils, est créée.Le processus parent continue d'exécuter le programme à partir de l'endroit où fork a été appelé.Le processus fils exécute lui aussi le même programme à partir du même endroit. Qu'est-ce qui différencient les deux processus alors ? Tout d'abord, le processus fils est unnouveau processus et dispose donc d'un nouvel identifiant de processus, distinct de celui del'identifiant de son processus parent. Un moyen pour un programme de savoir s'il fait partiedu processus père ou du processus fils est d'appeler la méthode getpid.

Cependant, la fonctionfork renvoie des valeurs différentes aux processus parent et enfant – un processus « rentre »dans le fork et deux en « ressortent » avec des valeurs de retour différentes. La valeur de retourdans le processus père est l'identifiant du processus fils. La valeur de retour dans le processusfils est zéro. Comme aucun processus n'a l'identifiant zéro, il est facile pour le programme dedéterminer s'il s'exécute au sein du processus père ou du processus fils.

Le Listing 3.3 est un exemple d'utilisation de fork pour dupliquer un processus. Notez quele premier bloc de la structure if n'est exécuté que dans le processus parent alors que la clauseelse est exécutée dans le processus fils.

Listing 3.3 – (fork.c) – Utiliser fork pour Dupliquer un Processus

```
1 int main ()

2 {

3 pid_t child_pid ;

4 printf («ID de processus du programme principal : %d\n» ,( int
) getpid ());

5 child_pid = fork ();

6 if ( child_pid != 0) {

7 printf («je suis le processus parent , ID : %d\n», ( int )
getpid ());

8 printf (« Identifiant du processus fils : %d\n», ( int ) child_
pid );

9 }

10 else

11 printf («je suis le processus fils , ID : %d\n», ( int ) getpid
());

12 return 0;

13 }
```

Utiliser la Famille de exec

Les fonctions exec remplacent le programme en cours d'exécution dans un processus parun autre programme. Lorsqu'un programme appelle la fonction exec, le processus cesse immédiatementd'exécuter ce programme et commence l'exécution d'un autre depuis le début, ensupposant que l'appel à exec se déroule correctement.

Au sein de la famille de exec existent plusieurs fonctions qui varient légèrement quant auxpossibilités qu'elles proposent et à la façon de les appeler.

Les fonctions qui contiennent la lettre p dans leur nom (execvp et execlp) reçoivent un nom de programme qu'elles recherchent dans le path courant ; il est nécessaire de passer le chemin d'accès complet du programme aux fonctions qui ne contiennent pas de p.

Les fonctions contenant la lettre v dans leur nom (execv, execvp et execve) reçoivent une liste d'arguments à passer au nouveau programme sous forme d'un tableau de pointeurs vers des chaînes terminé par NULL. Les fonctions contenant la lettre l (execl, execlp et execle) reçoivent la liste d'arguments via le mécanisme du nombre d'arguments variables du langage C.

Les fonctions qui contiennent la lettre e dans leur nom (execve et execle) prennent un argument supplémentaire, un tableau de variables d'environnement. L'argument doit être un tableau de pointeurs vers des chaînes terminé par NULL. Chaque chaîne doit être de la forme «VARIABLE=valeur».

Dans la mesure où exec remplace le programme appelant par un autre, on n'en sort jamais àmoins que quelque chose ne se déroule mal.Les arguments passés à un programme sont analogues aux arguments de ligne de commandeque vous transmettez à un programme lorsque vous le lancez depuis un shell. Ils sont accessiblespar le biais des paramètres argc et argv de main. Souvenez-vous que lorsqu'un programme estinvoqué depuis le shell, celui-ci place dans le premier élément de la liste d'arguments (argv[0])le nom du programme, dans le second (argv[1]) le premier paramètre en ligne de commande,etc. Lorsque vous utilisez une fonction exec dans votre programme, vous devriez, vous aussi,passer le nom du programme comme premier élément de la liste d'arguments.

Utiliser fork et exec

Un idiome courant pour l'exécution de sous-programme au sein d'un programme est d'effectuerun fork puis d'appeler exec pour le sous-programme. Cela permet au programme appelantde continuer à s'exécuter au sein du processus parent alors qu'il est remplacé par le sousprogrammedans le processus fils.

Le programme du Listing 3.4, comme celui du Listing 3.2, liste le contenu du répertoireracine en utilisant la commande ls. Contrairement à l'exemple précédent, cependant, il utilisela commande ls directement, en lui passant les arguments -l et / plutôt que de l'invoquerdepuis un shell.

Listing 3.4 – (fork-exec.c) – Utiliser fork et exec

```
1 # include <stdio .h>

2 # include <stdlib .h>

3 # include <sys / types .h>

4 # include <unistd .h>

5 /* Crée un processus fils exécutant un nouveau programme .
PROGRAM est le nom
```

```
6 du programme à exécuter ; le programme est recherché dans le
path .

7 ARG_LIST est une liste terminée par NULL de chaînes de
caractères à passer

8 au programme comme liste d'arguments . Renvoie l' identifiant du
processus

9 nouvellement créé . */

10 int spawn ( char * program , char ** arg_list )

11 {

12 pid_t child_pid ;

13 /* Duplique ce processus . */

14 child_pid = fork ();

15 if ( child_pid != 0)

16 /* Nous sommes dans le processus parent . */

17 return child_pid ;

18 else {

19 /* Exécute PROGRAM en le recherchant dans le path . */

20 execvp ( program , arg_list );

21 /* On ne sort de la fonction execvp uniquement si une erreur
survient . */

22 fprintf ( stderr , « une erreur est survenue au sein de execvp
\n»);

23 abort ();

24 }

25 }

26 int main ()

27 {

28 /* Liste d'arguments à passer à la commande «ls «. */

29 char * arg_list [] = {

30 «ls», /* argv [0] , le nom du programme . */

31 «-l»,

32 «/»,
```

```
33 NULL /* La liste d'arguments doit se terminer par NULL . */

34 };

35 /* Crée un nouveau processus fils exécutant la commande «ls «.
Ignore

36 l' identifiant du processus fils renvoyé . */

37 spawn («ls», arg_list );

38 printf (« Fin du programme principal \n»);

39 return 0;

40 }
```

Ordonnancement de Processus

Linux ordonnance le processus père indépendamment du processus fils ; il n'y a aucunegarantie sur celui qui sera exécuté en premier ou sur la durée pendant laquelle le premiers'exécutera avant que Linux ne l'interrompe et ne passe la main à l'autre processus (ou àunprocessus quelconque s'exécutant sur le système). Dans notre cas, lorsque le processus parentse termine, la commande ls peut s'être exécutée entièrement, partiellement ou pas du tout.

Linux garantit que chaque processus finira par s'exécuter – aucun processus ne sera à cours deressources.

Vous pouvez indiquer qu'un processus est moins important – et devrait avoir une prioritéinférieure – en lui assignant une valeur de priorité d'ordonnancement3 plus grande. Par défaut,chaque processus a une priorité d'ordonnancement de zéro. Une valeur plus élevée signifie quele processus a une priorité d'exécution plus faible ; inversement un processus avec une prioritéd'ordonnancement plus faible (c'est-à-dire, négative) obtient plus de temps d'exécution.

Pour lancer un programme avec une priorité d'ordonnancement différente de zéro, utilisez lacommande nice, en spécifiant la valeur de la priorité d'ordonnancement avec l'option -n. Parexemple, voici comment vous pourriez invoquer la commande sort input.txt > output. txt,une opération de tri longue, avec une priorité réduite afin qu'elle ne ralentisse pas trop le système :

```
% nice -n 10 sort input.txt > output.txt
```

Vous pouvez utiliser la commande renice pour changer la priorité d'ordonnancement d'unprocessus en cours d'exécution depuis la ligne de commande.

Pour changer la priorité d'ordonnancement d'un processus en cours d'exécution par programmation,utilisez la fonction nice. Son argument est une valeur d'ajustement qui est ajoutéeà la valeur de la priorité d'ordonnancement du processus qui l'appelle. Souvenez-vous qu'unevaleur positive augmente la priorité d'ordonnancement du processus et donc réduit la prioritéd'exécution du processus.

Notez que seuls les processus disposant des privilèges root peuvent lancer un processus avecune priorité d'ordonnancement négative ou réduire la valeur de la priorité d'ordonnancementd'un processus en cours d'exécution. Cela signifie que vous ne pouvez passer des valeurs négativesaux commandes nice et renice que lorsque vous êtes connecté en tant que root et seul unprocessus s'exécutant avec les privilèges root peut passer une valeur négative à la fonction nice.Cela évite que des utilisateurs ordinaires puissent s'approprier tout le temps d'exécution.

Signaux

Les signaux sont des mécanismes permettant de manipuler et de communiquer avec desprocessus sous Linux. Le sujet des signaux est vaste ; nous traiterons ici quelques-uns des signauxet techniques utilisés pour contrôler les processus.

Un signal est un message spécial envoyé à un processus. Les signaux sont asynchrones ;lorsqu'un processus reçoit un signal, il le traite immédiatement, sans même terminer la fonctionou la ligne de code en cours. Il y a plusieurs douzaines de signaux différents, chacun ayant unesignification différente. Chaque type de signal est caractérisé par son numéro de signal, maisau sein des programmes, on y fait souvent référence par un nom. Sous Linux, ils sont définisdans /usr/include/bits/signum.h (vous ne devriez pas inclure ce fichier directement dans vosprogrammes, utilisez plutôt <signal.h>).

Lorsqu'un processus reçoit un signal, il peut agir de différentes façons, selon l'action enregistrée

pour le signal. Pour chaque signal, il existe une action par défaut, qui détermine ce quiarrive au processus si le programme ne spécifie pas d'autre comportement. Pour la plupart dessignaux, le programme peut indiquer un autre comportement – soit ignorer le signal, soit appelerun gestionnaire de signal, fonction chargée de traiter le signal. Si un gestionnaire de signal estutilisé, le programme en cours d'exécution est suspendu, le gestionnaire est exécuté, puis, unefois celui-ci terminé, le programme reprend.

Le système Linux envoie des signaux aux processus en réponse à des conditions spécifiques. Par exemple, SIGBUS (erreur de bus), SIGSEGV (erreur de segmentation) et SIGFPE (exceptionde virgule flottante) peuvent être envoyés à un programme essayant d'effectuer une action nonautorisée. L'action par défaut pour ces trois signaux est de terminer le processus et de produireun fichier core.

Un processus peut également envoyer des signaux à un autre processus. Une utilisationcourante de ce mécanisme est de terminer un autre processus en lui envoyant un signal SIGTERMou SIGKILL4. Une autre utilisation courante est d'envoyer une commande à un programme encours d'exécution. Deux signaux "définis par l'utilisateur" sont réservés à cet effet : SIGUSR1et SIGUSR2. Le signal SIGHUP est également parfois utilisé dans ce but, habituellement pourréveiller un programme inactif ou provoquer une relecture du fichier de configuration.

La fonction sigaction peut être utilisée pour paramétrer l'action à effectuer en réponse à unsignal. Le premier paramètre est le numéro du signal. Les deux suivants sont des pointeurs versdes structures sigaction ; la première contenant l'action à effectuer pour ce numéro de signal,alors que la seconde est renseignée avec l'action précédente. Le champ le plus important, que cesoit dans la première ou la seconde structure sigaction est sa_handler. Il peut prendre une destrois valeurs suivantes :

SIG_DFL, qui correspond à l'action par défaut pour le signal ;

SIG_IGN, qui indique que le signal doit être ignoré ;

Un pointeur vers une fonction de gestion de signal. La fonction doit prendre un paramètre, le numéro du signal et être de type void.

Comme les signaux sont asynchrones, le programme principal peut être dans un état trèsfragile lorsque le signal est traité et donc pendant l'exécution du gestionnaire de signal. C'estpourquoi vous devriez éviter d'effectuer des opérations d'entrées/sorties ou d'appeler la plupartdes fonctions système ou de la bibliothèque C depuis un gestionnaire de signal.

Un gestionnaire de signal doit effectuer le minimum nécessaire au traitement du signal, puisrepasser le contrôle au programme principal (ou terminer le programme). Dans la plupart descas, cela consiste simplement à enregistrer que le signal est survenu. Le programme principalvérifie alors périodiquement si un signal a été reçu et réagit en conséquence.

Il est possible qu'un gestionnaire de signal soit interrompu par l'arrivée d'un autre signal. Bienque cela puisse sembler être un cas rare, si cela arrive, il peut être très difficile de diagnostiqueret résoudre le problème (c'est un exemple de conditions de concurrence critique, traité dans la section, « Threads »). C'est pourquoivous devez être très attentif à ce que votre programme fait dans un gestionnaire de signal.

Même l'affectation d'une valeur à une variable globale peut être dangereuse car elle peuten fait être effectuée en deux instructions machine ou plus, et un second signal peut survenir,laissant la variable dans un état corrompu. Si vous utilisez une variable globale pour indiquer laréception d'un signal depuis un gestionnaire de signal, elle doit être du type spécial sig_atomic_t.

Linux garantit que les affectations de valeur à des variables de ce type sont effectuées en uneseule instruction et ne peuvent donc pas être interrompues. Sous Linux, sig_atomic_t est un intordinaire ; en fait, les affectations de valeur à des types de la taille d'un int ou plus petits ouà des pointeurs, sont atomiques. Néanmoins, si vous voulez écrire un programme portable versn'importe quel système UNIX standard, utilisez le type sig_atomic_t.

Le squelette de programme du Listing 3.5, par exemple, utilise une fonction de gestion designal pour compter le nombre de fois où le programme reçoit le signal SIGUSR1, un des signauxutilisables par les applications.

Listing 3.5 – (sigusr1.c) – Utiliser un Gestionnaire de Signal

```
1 # include <signal .h>
2 # include <stdio .h>
3 # include <string .h>
4 # include <sys / types .h>
5 # include <unistd .h>
6 sig_atomic_t sigusr1_count = 0;
7 void handler ( int signal_number )
8 {
9 ++ sigusr1_count ;
10 }
11 int main ()
12 {
13 struct sigaction sa;
14 memset (&sa , 0, sizeof (sa));
15 sa. sa_handler = & handler ;
16 sigaction ( SIGUSR1 , &sa , NULL );
17 /* Faire quelque chose de long ici . */
18 /* ... */
19 printf (« SIGUSR1 a été reçu %d fois \n», sigusr1_count );
20 return 0;
21 }
```

Fin de Processus

Dans des conditions normales, un processus peut se terminer de deux façons : soit le programmeappelle la fonction exit, soit la fonction main du programme se termine. Chaqueprocessus a un code de sortie : un nombre que le processus renvoie à son Le code de sortieest l'argument passé à la fonction exit ou la valeur retournée depuis main.

Un processus peut également se terminer de façon anormale, en réponse à un signal. Parexemple, les signaux SIGBUS, SIGSEGV et SIGFPE évoqués précédemment provoquent la fin duprocessus. D'autres signaux sont utilisés pour terminer un processus explicitement. Le signalSIGINT est envoyé à un processus lorsque l'utilisateur tente d'y mettre fin en saisissant Ctrl+Cdans son terminal. Le signal SIGTERM est envoyé par la commande kill.

L'action par défautpour ces deux signaux est de mettre fin au processus. En appelant la fonction abort, un processuss'envoie à lui-même le signal SIGABRT ce qui termine le processus et produit un fichier core. Lesignal de terminaison le plus puissant est SIGKILL qui met fin à un processus immédiatementet ne peut pas être bloqué ou géré par un programme.

Chacun de ces signaux peut être envoyé en utilisant la commande kill en passant uneoption supplémentaire sur la ligne de commande ; par exemple, pour terminer un programmefonctionnant mal en lui envoyant un SIGKILL, invoquez la commande suivante, où pid est unidentifiant de processus :

% kill -KILL pid}

Pour envoyer un signal depuis un programme, utilisez la fonction kill. Le premier paramètreest l'identifiant du processus cible. Le second est le numéro du signal ; utilisez SIGTERM poursimuler le comportement par défaut de la commande kill. Par exemple, si child_pid contientl'identifiant du processus fils, vous pouvez utiliser la fonction kill pour terminer un processusfils depuis le père en l'appelant comme ceci :

kill child_pid, SIGTERM) ;

Incluez les entêtes <sys/types.h> et <signal.h> si vous utilisez la fonction kill.Par convention, le code de sortie est utilisé pour indiquer si le programme s'est exécutécorrectement. Un code de sortie à zéro indique une exécution correcte, alors qu'un code différentde zéro indique qu'une erreur est survenue. Dans ce dernier cas, la valeur renvoyée peut donnerune indication sur la nature de l'erreur. C'est une bonne idée d'adopter cette convention dans vosprogrammes car certains composants du système GNU/Linux sont basés dessus. Par exemple, lesshells supposent l'utilisation de cette convention lorsque vous connectez plusieurs programmesavec les opérateurs && (et logique) et ll (ou logique). C'est pour cela que vous devriez renvoyerzéro explicitement depuis votre fonction main, à moins qu'une erreur ne survienne.

Avec la plupart des shells, il est possible d'obtenir le code de sortie du dernier programmeexécuté en utilisant la variable spéciale $? Voici un exemple dans lequel la commande ls estinvoquée deux fois et son code de sortie est affiché après chaque invocation. Dans le premier cas,ls se termine correctement et renvoie le code de sortie 0. Dans le second cas, ls rencontre uneerreur (car le fichier spécifié sur la ligne de commande n'existe pas) et renvoie donc un code desortie différent de 0.

```
% ls /
bin coda etc lib misc nfs proc
boot dev home lost+found mnt opt root
% echo $?
0%
ls fichierinexistant
ls: fichierinexistant: Aucun fichier ou répertoire de ce type
% echo $?
1
```

Notez que même si le paramètre de la fonction exit est de type int et que la fonction mainrenvoie un int, Linux ne conserve pas les 32 bits du code de sortie. En fait, vous ne devriezutiliser que des codes de sortie entre 0 et 127. Les codes de sortie au-dessus de 128 ont unesignification spéciale – lorsqu'un processus se termine à cause d'un signal, son code de sortie estzéro plus le numéro du signal.

Attendre la Fin d'un Processus

Si vous avez saisi et exécuté l'exemple de fork et exec du Listing 3.4, vous pouvez avoirremarqué que la sortie du programme ls apparaît souvent après la fin du programme principal.Cela est dû au fait que le processus fils, au sein duquel s'exécute ls, est ordonnancéindépendamment du processus père. Comme Linux est un système d'exploitation multitâche, lesdeux processus semblent s'exécuter simultanément et vous ne pas prédire si le programme ls vas'exécuter avant ou après le processus père.

Dans certaines situations, cependant, il est souhaitable que le processus père attende la find'un ou plusieurs processus fils. Pour cela, il est possible d'utiliser les appels systèmes de lafamille de wait. Ces fonctions vous permettent d'attendre la fin d'un processus et permettentau processus parent d'obtenir des informations sur la façon dont s'est terminé son fils. Il y aquatre appels système différents dans la famille de wait ; vous pouvez choisir de récupérer peuou beaucoup d'informations sur le processus qui s'est terminé et vous pouvez indiquer si vousvoulez savoir quel processus fils s'est terminé.

Les Appels Système wait

La fonction la plus simple s'appelle simplement wait. Elle bloque le processus appelantjusqu'à ce qu'un de ses processus fils se termine (ou qu'une erreur survienne). Elle retourneun code de statut via un pointeur sur un entier, à partir duquel vous pouvez extraire desinformations sur la façon dont s'est terminé le processus fils. Par exemple, la macro WEXITSTATUSextrait le code de sortie du processus fils.

Vous pouvez utiliser la macro WIFEXITED pour déterminer si un processus s'est terminécorrectement à partir de son code de statut (via la fonction exit ou la sortie de main) ouest mort à cause d'un signal non intercepté. Dans ce dernier cas, utilisez la macro WTERMSIG pourextraire le numéro du signal ayant causé la mort du processus à partir du code de statut.

Voici une autre version de la fonction main de l'exemple de fork et exec. Cette fois, leprocessus parent appelle wait pour attendre que le processus fils, dans lequel s'exécute lacommande ls, se termine.

```
int main ()

{

int child_status ;

/* Liste d'arguments à passer à la commande «ls». */

char * arg_list [] = {

«ls», /* argv[0], le nom du programme. */

«-l»,

«/»,

NULL /* La liste d'arguments doit se terminer par NULL. */

};

/* Crée un nouveau processus fils exécutant la commande «ls».
Ignore

l'identifiant du processus fils renvoyé. */

spawn («ls», arg_list );

/* Attend la fin du processus fils. */

wait (& child_status );

if ( WIFEXITED ( child_status ))

printf (« processus fils terminé normalement , le code de sortie
%d\n»,

WEXITSTATUS ( child_status ));

else

printf (« processus fils terminé anormalement \n»);

return 0;

}
```

Plusieurs appels système similaires sont disponibles sous Linux, ils sont plus flexibles ouapportent plus d'informations sur le processus fils se terminant. La fonction waitpid peutêtre utilisée pour attendre la fin d'un processus fils spécifique au lieu d'attendre n'importequel processus fils. La fonction wait3 renvoie des statistiques sur l'utilisation du processeurpar le processus fils se terminant et la fonction wait4 vous permet de spécifier des optionssupplémentaires quant au processus dont on attend la fin.

Processus Zombies

Si un processus fils se termine alors que son père appelle la fonction wait, le processus filsdisparaît et son statut de terminaison est transmis à son père via l'appel à wait. Mais que sepasse-t-il lorsqu'un processus se termine et que son père n'appelle pas wait ? Disparaît-il toutsimplement ? Non, car dans ce cas, les informations concernant la façon dont il s'est terminé –comme le fait qu'il se soit terminé normalement ou son code de sortie – seraient perdues. Aulieu de cela, lorsqu'un processus fils se termine, il devient un processus zombie.

Un processus zombie est un processus qui s'est terminé mais dont les ressources n'ont pasencore été libérées. Il est de la responsabilité du processus père de libérer les ressources occupéespar ses fils zombies. La fonction wait le fait, il n'est donc pas nécessaire de savoir si votreprocessus fils est toujours en cours d'exécution avant de l'attendre. Supposons, par exemple,qu'un programme crée un processus fils, fasse un certain nombre d'autres opérations puis appellewait. Si le processus fils n'est pas encore terminé à ce moment, le processus parent sera bloquédans l'appel de wait jusqu'à ce que le processus fils se termine. Si le processus fils se termineavant que le père n'appelle wait, il devient un processus zombie. Lorsque le processus parentappelle wait, le statut de sortie du processus fils est extrait, le processus fils est supprimé etl'appel de wait se termine immédiatement.

Que se passe-t-il si le père ne libère pas les ressources de ses fils ? Ils restent dans le système,sous la forme de processus zombies. Le programme du Listing 3.6 crée un processus fils qui setermine immédiatement, puis s'interrompt pendant une minute, sans jamais libérer les ressourcesdu processus fils.

Listing 3.6 – (zombie.c) – Créer un Processus Zombie

```
1 # include <stdlib .h>
2 # include <sys / types .h>
3 # include <unistd .h>
4 int main ()
5 {
6 pid_t child_pid ;
7 /* Crée un processus fils . */
8 child_pid = fork ();
```

```
 9 if ( child_pid > 0) {

 10 /* Nous sommes dans le processus parent . Attente d'une minute
. */

 11 sleep (60) ;

 12 }

 13 else {

 14 /* Nous sommes dans le processus fils . Sortie immédiate . */

 15 exit (0) ;

 16 }

 17 return 0;

 18 }
```

Compilez ce fichier en un exécutable appelé make-zombie. Exécutez-le, et pendant ce temps,listez les processus en cours d'exécution sur le système par le biais de la commande suivantedans une autre fenêtre :

% ps -e -o pid,ppid,stat,cmd

Elle dresse la liste des identifiants de processus, de processus pères, de leur statut et de laligne de commande du processus. Observez qu'en plus du processus père make-zombie, un autreprocessus make-zombie est affiché. Il s'agit du processus fils ; notez que l'identifiant de son pèreest celui du processus make-zombie principal. Le processus fils est marqué comme <defunct>et son code de statut est Z pour zombie.

Que se passe-t-il lorsque le programme principal de make-zombie se termine sans appelerwait ? Le processus zombie est-il toujours présent ? Non – essayez de relancer ps et notez que lesdeux processus make-zombie ont disparu. Lorsqu'un programme se termine, un processus spécialhérite de ses fils, le programme init, qui s'exécute toujours avec un identifiant de processusvalant 1 (il s'agit du premier processus lancé lorsque Linux démarre). Le processus init libèreautomatiquement les ressources de tout processus zombie dont il hérite.

Libérer les Ressources des Fils de Façon Asynchrone

Si vous utilisez un processus fils uniquement pour appeler exec, il est suffisant d'appeler waitimmédiatement dans le processus parent, ce qui le bloquera jusqu'à ce que le processus fils setermine. Mais souvent, vous voudrez que le processus père continue de s'exécuter alors qu'un ouplusieurs fils s'exécutent en parallèle. Comment être sûr de libérer les ressources occupées partous les processus fils qui se sont terminés afin de ne pas laisser de processus zombie dans lanature, ce qui consomme des ressources ?

Une approche envisageable serait que le processus père appelle périodiquement wait3 etwait4 pour libérer les ressources des fils zombies. Appeler wait de cette façon ne fonctionne pasde manière optimale car si aucun fils ne s'est terminé, l'appelant sera bloqué jusqu'à ce que cesoit le cas. Cependant, wait3 et wait4 prennent un paramètre d'option supplémentaire auquelvous pouvez passer le drapeau WNOHANG. Avec ce drapeau, la fonction s'exécute en mode nonbloquant – elle libérera les ressources d'un processus fils terminé s'il y en a un ou se termineras'il n'y en a pas. La valeur de retour de l'appel est l'identifiant du processus fils s'étant terminédans le premier cas, zéro dans le second.

Une solution plus élégante est d'indiquer au processus père quand un processus fils se termine.Il y a plusieurs façons de le faire en utilisant les méthodes présentées dans la partie, «Communication Interprocessus », mais heureusement, Linux le fait pour vous, en utilisant lessignaux. Lorsqu'un processus fils se termine, Linux envoie au processus père le signal SIGCHLD.

L'action par défaut pour ce signal est de ne rien faire, c'est la raison pour laquelle vous pouvezne jamais l'avoir remarqué auparavant.Donc, une façon élégante de libérer les ressources des processus fils est de gérer SIGCHLD. Biensûr, lorsque vous libérez les ressources du processus fils, il est important de stocker son statutde sortie si cette information est nécessaire, car une fois que les ressources du processus filsont été libérées par wait, cette information n'est plus disponible. Le Listing 3.7 montre à quoiressemble un programme utilisant un gestionnaire pour SIGCHLD afin de libérer les ressources deses processus fils.

Listing 3.7 – (sigchld.c) – Libérer les Ressources des Fils via SIGCHLD

```
1 # include <signal .h>
2 # include <string .h>
3 # include <sys / types .h>
4 # include <sys / wait .h>
5 sig_atomic_t child_exit_status ;
6 void clean_up_child_process ( int signal_number )
7 {
8 /* Nettoie le ou les processus fils . */
9 int status ;
10 while ( waitpid (-1, & status , WNOHANG ));
11 /* Stocke la statut de sortie du dernier dans une variable
globale . */
12 child_exit_status = status ;
13 }
14 int main ()
```

```
15 {
16 /* Gère SIGCHLD en appelent clean_up_child_process . */
17 struct sigaction sigchld_action ;
18 memset (& sigchld_action , 0, sizeof ( sigchld_action ));
19 sigchld_action . sa_handler = & clean_up_child_process ;
20 sigaction ( SIGCHLD , & sigchld_action , NULL );
21 /* Faire diverses choses , entre autres créer un processus fils
. */
22 /* ... */
23 return 0;
24 }
```

Notez comment le gestionnaire de signal stocke le statut de sortie du processus fils dans unevariable globale, à laquelle le programme principal peut accéder. Comme la variable reçoit unevaleur dans un gestionnaire de signal, elle est de type sig_atomic_t.

Conclusion

Dans cette activité, nous avons présenté les concepts de processus et expliqué les différentes techniques et outils pour la création et l'utilisation des processus. En particulier, la création de processus utilisant le système, fork et les fonctions exec a été présentée. En outre, la gestion et la manipulation des processus à l'aide de divers signaux du système a été mis en évidence.

Évaluation

1. Exécuter les différents exemples de codes fournis dans cette activité

Activité 2: Threads

Introduction

Nous avons vu comment un programme peut créer un processus fils. Celui-ci exécute immédiatementle programme de son père, la mémoire virtuelle, les descripteurs de fichiers, etc. de sonpère étant copiés. Le processus fils peut modifier sa mémoire, fermer les descripteurs de fichierssans que cela affecte son père, et vice versa. Lorsqu'un programme crée un nouveau thread,par contre, rien n'est copié. Le thread créateur et le thread créé partagent tous deux le mêmeespace mémoire, les mêmes descripteurs de fichiers et autres ressources. Si un thread modifiela valeur d'une variable, par exemple, l'autre thread verra la valeur modifiée. De même, si unthread ferme un descripteur de fichier, les autres threads ne peuvent plus lire ou écrire dans cefichier. Comme un processus et tous ses threads ne peuvent exécuter qu'un seul programme à lafois, si un thread au sein d'un processus appelle une des fonctions exec, tous les autres threadsse terminent (le nouveau programme peut, bien sûr, créer de nouveaux threads).

GNU/Linux implémente l'API de threading standard POSIX (appelée aussi pthreads). Toutesles fonctions et types de données relatifs aux threads sont déclarés dans le fichier d'entête <pthread.h>. Les fonctions de pthread ne font pas partie de la bibliothèque standard du C. Ellesse trouvent dans libpthread, vous devez donc ajouter -lpthread sur la ligne de commandelors de l'édition de liens de votre programme.

Création de Threads

Chaque thread d'un processus est caractérisé par un identifiant de thread. Lorsque vousmanipulez les identifiants de threads dans des programmes C ou C++, veillez à utiliser le typepthread_t.

Lors de sa création, chaque thread exécute une fonction de thread. Il s'agit d'une fonctionordinaire contenant le code que doit exécuter le thread. Lorsque la fonction se termine, lethread se termine également. Sous GNU/Linux, les fonctions de thread ne prennent qu'unseul paramètre de type void* et ont un type de retour void*. Ce paramètre est l'argumentde thread : GNU/Linux passe sa valeur au thread sans y toucher. Votre programme peut utiliserce paramètre pour passer des données à un nouveau thread. De même, il peut utiliser la valeurde retour pour faire en sorte que le thread renvoie des données à son créateur lorsqu'il se termine.

La fonction pthread_create crée un nouveau thread. Voici les paramètres dont elle a besoin :

- Un pointeur vers une variable pthread_t, dans laquelle l'identifiant du nouveau thread sera stocké ;
- Un pointeur vers un objet d'attribut de thread. Cet objet contrôle les détails de l'interaction du thread avec le reste du programme. Si vous passez NULL comme argument de thread, le thread est créé avec les attributs par défaut. Ceux-ci sont traités dans la Section, «Attributs de Threads» ;

- Un pointeur vers la fonction de thread. Il s'agit d'un pointeur de fonction ordinaire de type : void* (*)(void*) ;
- item Une valeur d'argument de thread de type void*. Quoi que vous passiez, l'argument est simplement transmis à la fonction de thread lorsque celui-ci commence à s'exécuter.

Un appel à pthread_create se termine immédiatement et le thread original continue àexécuter l'instruction suivant l'appel. Pendant ce temps, le nouveau thread débute l'exécutionde la fonction de thread. Linux ordonnance les deux threads de manière asynchrone et votreprogramme ne doit pas faire d'hypothèse sur l'ordre d'exécution relatif des instructions dans lesdeux threads.

Le programme du Listing 4.1 crée un thread qui affiche x de façon continue sur la sortie deserreurs. Après l'appel de pthread_create, le thread principal affiche des o indéfiniment sur lasortie des erreurs.

Listing 4.1 – (thread-create.c) – Créer un thread

```
1 # include <pthread .h>
  2 # include <stdio .h>
  3 /* Affiche des x sur stderr . Paramètre inutilisé . Ne finit
  jamais . */
  4 void * print_xs ( void * unused )
  5 {
  6 while (1)
  7 fputc ('x', stderr );
  8 return NULL ;
  9 }
  10 /* Le programme principal . */
  11 int main ()
  12 {
  13 pthread_t thread_id ;
  14 /* Crée un nouveau thread . Le nouveau thread exécutera la
  fonction
  15 print_xs . */
  16 pthread_create (& thread_id , NULL , & print_xs , NULL );
  17 /* Affiche des o en continue sur stderr . */
```

```
18 while (1)
19 fputc ('o', stderr );
20 return 0;
21 }
```

Compilez ce programme en utilisant la commande suivante :

```
% cc -o thread-create thread-create.c -lpthread
```

Essayez de le lancer pour voir ce qui se passe. Notez que le motif formé par les x et les o estimprévisible car Linux passe la main alternativement aux deux threads.Dans des circonstances normales, un thread peut se terminer de deux façons. La première,illustrée précédemment, est de terminer la fonction de thread. La valeur de retour de la fonctionde thread est considérée comme la valeur de retour du thread. Un thread peut également seterminer explicitement en appelant pthread_exit. Cette fonction peut être appelée depuis lafonction de thread ou depuis une autre fonction appelée directement ou indirectement par lafonction de thread. L'argument de pthread_exit est la valeur de retour du thread.

Transmettre des Données à un Thread

L'argument de thread est une méthode pratique pour passer des données à un thread. Commeson type est void*, cependant, vous ne pouvez pas passer beaucoup de données directementen l'utilisant. Au lieu de cela, utilisez l'argument de thread pour passer un pointeur vers unestructure ou un tableau de données. Une technique couramment utilisée est de définir unestructure de données pour chaque argument de thread, qui contient les paramètres attenduspar la fonction de thread.

En utilisant l'argument de thread, il est facile de réutiliser la même fonction pour différentsthreads. Ils exécutent alors tous les mêmes traitements mais sur des données différentes.

Le programme du Listing 4.2 est similaire à l'exemple précédent. Celui-ci crée deux nouveauxthreads, l'un affiche des x et l'autre des o. Au lieu de les afficher indéfiniment, cependant, chaquethread affiche un nombre prédéterminé de caractères puis se termine en sortant de la fonction dethread. La même fonction de thread, char_print, est utilisée par les deux threads mais chacunest configuré différemment en utilisant une struct char_print_parms.

Listing 4.2 – (thread-create2.c) – Créer Deux Threads

```
1 # include <pthread .h>
2 # include <stdio .h>
3 /* Paramètres de la fonction print . */
4 struct char_print_parms
5 {
```

```
6 /* Caractère à afficher . */
7 char character ;
8 /* Nombre de fois où il doit être affiché . */
9 int count ;
10 };
11 /* Affiche un certain nombre de caractères sur stderr , selon
le contenu de
12 PARAMETERS , qui est un pointeur vers une struct char_print_
parms . */
13 void * char_print ( void * parameters )
14 {
15 /* Effectue un transtypage du pointeur void vers le bon type .
*/
16 struct char_print_parms * p = ( struct char_print_parms *)
parameters ;
17 int i;
18 for (i = 0; i < p-> count ; ++i)
19 fputc (p-> character , stderr );
20 return NULL ;
21 }
22 /* Programme principal . */
23 int main ()
24 {
25 pthread_t thread1_id ;
26 pthread_t thread2_id ;
27 struct char_print_parms thread1_args ;
28 struct char_print_parms thread2_args ;
29 /* Crée un nouveau thread affichant 30 000 x. */
30 thread1_args . character = 'x';
31 thread1_args . count = 30000;
32 pthread_create (& thread1_id , NULL , & char_print , &
thread1_args );
```

```
33 /* Crée un nouveau thread affichant 20 000 'o '. */
34 thread2_args . character = ?o?;
35 thread2_args . count = 20000;
36 pthread_create (& thread2_id , NULL , & char_print , &
thread2_args );
37 return 0;
38 }
```

Mais attendez ! Le programme du Listing 4.2 est sérieusement bogué. Le thread principal(qui exécute la fonction main) crée les structures passées en paramètre aux threads (thread1_argset thread2_args) comme des variables locales puis transmet des pointeurs sur ces structures auxthreads qu'il crée. Qu'est-ce qui empêche Linux d'ordonnancer les trois threads de façon à ceque main termine son exécution avant que l'un des deux autres threads s'exécutent ? Rien ! Maissi cela se produit, la mémoire contenant les structures de paramètres des threads sera libéréealors que les deux autres threads tentent d'y accéder.

Synchroniser des Threads

Une solution possible est de forcer main à attendre la fin des deux autres threads. Ce dontnous avons besoin est une fonction similaire à wait qui attende la fin d'un thread au lieu decelle d'un processus. Cette fonction est pthread_join, qui prend deux arguments : l'identifiantdu thread à attendre et un pointeur vers une variable void* qui recevra la valeur de retour duthread s'étant terminé. Si la valeur de retour du thread ne vous est pas utile, passez NULL commesecond argument.
Le Listing 4.3 montre la fonction main sans le bogue du Listing 4.2. Dans cette version, mainne se termine pas jusqu'à ce que les deux threads affichant des o et des x se soient eux-mêmes terminés, afin qu'ils n'utilisent plus les structures contenant les arguments.

Listing 4.3 – (thread-create2a.c) – Fonction main de thread-create2.c corrigée

```
1 int main ()
2 {
3 pthread_t thread1_id ;
4 pthread_t thread2_id ;
5 struct char_print_parms thread1_args ;
6 struct char_print_parms thread2_args ;
7 /* Crée un nouveau thread affichant 30 000 x. */
8 thread1_args . character = ?x?;
9 thread1_args . count = 30000;
```

```
10 pthread_create (& thread1_id , NULL , & char_print , &
thread1_args );
11 /* Crée un nouveau thread affichant 20 000 o. */
12 thread2_args . character = ?o?;
13 thread2_args . count = 20000;
14 pthread_create (& thread2_id , NULL , & char_print , &
thread2_args );
15 /* S'assure que le premier thread est terminé . */
16 pthread_join ( thread1_id , NULL );
17 /* S'assure que le second thread est terminé . */
18 pthread_join ( thread2_id , NULL );
19 /* Nous pouvons maintenant quitter en toute sécurité . */
20 return 0;
21 }
```

Morale de l'histoire : assurez-vous que toute donnée que vous passez à un thread par référencen'est pas libérée, même par un thread différent, à moins que vous ne soyez sûr que le threada fini de l'utiliser. Cela s'applique aux variables locales, qui sont libérées lorsqu'elles sont horsde portée, ainsi qu'aux variables allouées dans le tas, que vous libérez en appelant free (ou enutilisant delete en C++).

Valeurs de Retour des Threads

Si le second argument que vous passez à pthread_join n'est pas NULL, la valeur de retour duthread sera stockée à l'emplacement pointé par cet argument. La valeur de retour du thread,comme l'argument de thread, est de type void*. Si vous voulez renvoyer un simple int ou unautre petit nombre, vous pouvez le faire facilement est convertissant la valeur en void* puis enle reconvertissant vers le type adéquat après l'appel de pthread_join.

Le programme du Listing 4.4 calcule le nième nombre premier dans un thread distinct. Cethread renvoie le numéro du nombre premier demandé via sa valeur de retour de thread. Lethread principal, pendant ce temps, est libre d'exécuter d'autres traitements. Notez que l'algorithmede divisions successives utilisé dans compute_prime est quelque peu inefficace ; consultezun livre sur les algorithmes numériques si vous avez besoin de calculer beaucoup de nombrespremiers dans vos programmes.

Listing 4.4 – (primes.c) – Calcule des Nombres Premiers dans un Thread

```
1 # include <pthread .h>
2 # include <stdio .h>
3 /* Calcules des nombres premiers successifs ( très inefficace ).
Renvoie
4 le Nième nombre premier où N est la valeur pointée par * ARG .
*/
5 void * compute_prime ( void * arg )
6 {
7 int candidate = 2;
8 int n = *(( int *) arg );
9 while (1) {
10 int factor ;
11 int is_prime = 1;
12 /* Teste si le nombre est premier par divisions successives .
*/
13 for ( factor = 2; factor < candidate ; ++ factor )
14 if ( candidate % factor == 0) {
15 is_prime = 0;
16 break ;
17 }
18 /* Est -ce le nombre premier que nous cherchons ? */
19 if ( is_prime ) {
20 if (--n == 0)
21 /* Renvoie le nombre premier désiré via la valeur de retour du
thread . */
22 return ( void *) candidate ;
23 }
24 ++ candidate ;
25 }
26 return NULL ;
```

```
27 }
28 int main ()
29 {
30 pthread_t thread ;
31 int which_prime = 5000;
32 int prime ;
33 /* Démarre le thread de calcul jusqu 'au 5 000 ème nombre
premier . */
34 pthread_create (& thread , NULL , & compute_prime , & which_
prime );
35 /* Faire autre chose ici ... */
36 /* Attend la fin du thread de calcul et récupère le résultat .
*/
37 pthread_join ( thread , ( void *) & prime );
38 /* Affiche le nombre premier calculé . */
39 printf («Le % dème nombre premier est %d.\n», which_prime ,
prime );
40 return 0;
41 }
```

Plus d'Informations sur les Identifiants de Thread

De temps à autre, il peut être utile pour une portion de code de savoir quel thread l'exécute.La fonction pthread_self renvoie l'identifiant du thread depuis lequel elle a été appelée. Cetidentifiant de thread peut être comparé à un autre en utilisant la fonction pthread_equal.Ces fonctions peuvent être utiles pour déterminer si un identifiant de thread particuliercorrespond au thread courant. Par exemple, un thread ne doit jamais appeler pthread_joinpour s'attendre lui-même (dans ce cas, pthread_join renverrait le code d'erreur EDEADLK). Pourle vérifier avant l'appel, utilisez ce type de code :

```
if (! pthread_equal ( pthread_self () , other_thread ))
pthread_join ( other_thread , NULL );
```

Attributs de Thread

Les attributs de thread proposent un mécanisme pour contrôler finement le comportement dethreads individuels. Souvenez-vous que pthread_create accepte un argument qui est un pointeurvers un objet d'attributs de thread. Si vous passez un pointeur nul, les attributs par défaut sontutilisés pour configurer le nouveau thread. Cependant, vous pouvez créer et personnaliser unobjet d'attributs de threads pour spécifier vos propres valeurs.

Pour indiquer des attributs de thread personnalisés, vous devez suivre ces étapes :

- Créez un objet pthread_attr_t. La façon la plus simple de le faire est de déclarer une variable automatique de ce type.
- Appelez pthread_attr_init en lui passant un pointeur vers cet objet. Cela initialise les attributs à leur valeur par défaut.
- Modifiez l'objet d'attributs pour qu'ils contiennent les valeurs désirées.
- Passez un pointeur vers l'objet créé lors de l'appel à pthread_create.
- Appelez pthread_attr_destroy pour libérer l'objet d'attributs. La variable pthread_attr_t n'est pas libérée en elle-même ; elle peut être réinitialisée avec pthread_attr_init.

Un même objet d'attributs de thread peut être utilisé pour démarrer plusieurs threads. Iln'est pas nécessaire de conserver l'objet d'attributs de thread une fois qu'ils ont été créés.

Pour la plupart des applications GNU/Linux, un seul attribut de thread a généralement del'intérêt (les autres attributs disponibles sont principalement destinés à la programmation entemps réel). Cet attribut est l'état de détachement (detach state) du thread. Un thread peutêtre un thread joignable (par défaut) ou un thread détaché. Les ressources d'un thread joignable,comme pour un processus, ne sont pas automatiquement libérées par GNU/Linux lorsqu'il setermine. Au lieu de cela, l'état de sortie du thread reste dans le système (un peu comme pour unprocessus zombie) jusqu'à ce qu'un autre thread appelle pthread_join pour récupérer cette valeurde retour. Alors seulement, les ressources sont libérées. Les ressources d'un thread détaché, aucontraire, sont automatiquement libérées lorsqu'il se termine. Cela peut impliquer qu'un autre threadne peut attendre qu'il se termine avec pthread_join ou obtenir sa valeur de retour.

La fonction pthread_attr_setdetachstate permet de définir l'état de détachement dans unobjet d'attributs de thread. Le premier argument est un pointeur vers l'objet d'attributs dethread et le second est l'état désiré. Comme l'état joignable est la valeur par défaut, il n'est nécessaired'effectuer cet appel que pour créer des threads détachés ; passez PTHREAD_CREATE_DETACHEDcomme second argument.

Le code du Listing 4.5 crée un thread détaché en activant l'attribut de détachement dethread :

Listing 4.5 – (detached.c) – Programme Squelette Créant un Thread Détaché

```
1 # include <pthread .h>
2 void * thread_function ( void * thread_arg )
3 {
4 /* Effectuer les traitements ici ... */
5 }
6 int main ()
7 {
8 pthread_attr_t attr ;
9 pthread_t thread ;
10 pthread_attr_init (& attr );
11 pthread_attr_setdetachstate (& attr , PTHREAD_CREATE_DETACHED
);
12 pthread_create (& thread , &attr , & thread_function , NULL );
13 pthread_attr_destroy (& attr );
14 /* Effectuer les traitements ici ... */
15 /* Pas besoin d'attendre le deuxième thread . */
16 return 0;
17 }
```

Même si un thread est créé dans un état joignable, il peut être transformé plus tard en unthread détaché. Pour cela, appelez pthread_detach. Une fois qu'un thread est détaché, il ne peutplus être rendu joignable.

Conclusion

Dans cette activité, nous avons présenté les concepts de thread et expliqué divers techniques et outils pour la création, la gestion et l'exécution des threads. En particulier, la création de trhead en utilisant la fonction pthread_create qui a été présentée. D'autres techniques de gestion de threads seront étudiées plus tard.

Évaluation

1. Exécuter les différents exemples de codes fournis dans cette activité

Activité 3: Gestion de la mémoire

Introduction

La mémoire comprend un certain nombre de cellules pouvant stocker un certain nombre de bits. La mémoire est juste un tableau d'octets. Chaque cellule a un certain nombre permettant de l'identifier, appelé son adresse. Les programmes se réfèrent aux adresses pour accéder à la cellule mémoire. Des cellules adjacentes ont des adresses consécutives. Si la mémoire a m cellules, les cellules auront pour adresses 0 à m-1.Si le CPU prend en charge n bits, il peut se référer 0-2 ^ n – 1adresses. Par exemple, Intel Pentium II est un processeur 32 bits et peut traiter 4 Go de mémoire. Chaque cellule stockeun nombre entier. Un entier est un nombre de nbit.

Mémoire Partagée

Une des méthodes de communication interprocessus les plus simples est d'utiliser la mémoirepartagée. La mémoire partagée permet à deux processus ou plus d'accéder à la même zonemémoire comme s'ils avaient appelé malloc et avaient obtenu des pointeurs vers le mêmeespace mémoire. Lorsqu'un processus modifie la mémoire, tous les autres processus voient lamodification.

Communication Locale Rapide

La mémoire partagée est la forme de communication interprocessus la plus rapide car tousles processus partagent la même mémoire. L'accès à cette mémoire partagée est aussi rapide quel'accès à la mémoire non partagée du processus et ne nécessite pas d'appel système ni d'entréedans le noyau. Elle évite également les copies de données inutiles.

Comme le noyau ne coordonne pas les accès à la mémoire partagée, vous devez mettre en placevotre propre synchronisation. Par exemple, un processus ne doit pas effectuer de lecture avantque des données aient été écrites et deux processus ne doivent pas écrire au même emplacementen même temps. Une stratégie courante pour éviter ces conditions de concurrence est d'utiliserdes sémaphores, ce dont nous parlerons dans la prochaine section. Nos programmes d'illustration,cependant, ne montrent qu'un seul processus accédant à la mémoire, afin de se concentrersur les mécanismes de la mémoire partagée et éviter d'obscurcir le code avec la logique desynchronisation.

Le Modèle Mémoire

Pour utiliser un segment de mémoire partagée, un processus doit allouer le segment. Puis,chaque processus désirant accéder au segment doit l'attacher. Après avoir fini d'utiliser lesegment, chaqueprocessus le détache. À un moment ou à un autre, un processus doit libérer lesegment.

La compréhension du modèle mémoire de Linux aide à expliquer le processus d'allocation etd'attachement. Sous Linux, la mémoire virtuelle de chaque processus est divisée en pages. Chaqueprocessus conserve une correspondance entre ses adresses mémoire et ces pages de mémoirevirtuelle, qui contiennent réellement les données. Même si chaque processus dispose de ses propresadresses, plusieurs tables de correspondance peuvent pointer vers la même page, permettant lepartage de mémoire.

L'allocation d'un nouveau segment de mémoire partagée provoque la création de nouvellespages de mémoire virtuelle. Comme tous les processus désirent accéder au même segment demémoire partagée, seul un processus doit allouer un nouveau segment de mémoire partagée.

Allouer un segment existant ne crée pas de nouvelles pages mais renvoie l'identifiant des pagesexistantes. Pour qu'un processus puisse utiliser un segment de mémoire partagée, il doit l'attacher,ce qui ajoute les correspondances entre sa mémoire virtuelle et les pages partagées dusegment. Lorsqu'il en a terminé avec le segment, ces correspondances sont supprimées. Lorsqueplus aucun processus n'a besoin d'accéder à ces segments de mémoire partagée, un processusexactement doit libérer les pages de mémoire virtuelle.

Tous les segments de mémoire partagée sont alloués sous forme de multiples entiers de lataille de page du système, qui est le nombre d'octets dans une page mémoire. Sur les systèmesLinux, la taille de page est de 4 Ko mais vous devriez vous baser sur la valeur renvoyée pargetpagesize.

Allocation

Un processus alloue un segment de mémoire partagée en utilisant shmget (« SHared MemoryGET », obtention de mémoire partagée). Son premier paramètre est une clé entière qui indique lesegment à créer. Des processus sans lien peuvent accéder au même segment partagé en spécifiantla même valeur de clé. Malheureusement, d'autres processus pourraient avoir choisi la mêmevaleur de clé fixé, ce qui provoquerait un conflit. Utiliser la constante spéciale IPC_PRIVATEcomme valeur de clé garantit qu'un nouveau segment mémoire est créé.

Le second paramètre indique le nombre d'octets du segment. Comme les segments sontalloués en utilisant des pages, le nombre d'octets effectivement alloués est arrondi au multiplede la taille de page supérieur.

Le troisième paramètre est un ou binaire entre des indicateurs décrivant les options demandéesà shmget. Voici ces indicateurs :

- IPC_CREAT – Cet indicateur demande la création d'un nouveau segment. Cela permet la création d'un nouveau segment tout en spécifiant une valeur de clé.
- IPC_EXCL – Cet indicateur, toujours utilisé avec IPC_CREAT, provoque l'échec de shmget si la clé de segment spécifiée existe déjà. Donc, cela permet au processus appelant d'avoir un segment « exclusif ». Si cette option n'est pas précisée et que la clé d'un segment existant est utilisée, shmget renvoie le segment existant au lieu d'en créer un nouveau.
- Indicateurs de mode – Cette valeur est constituée de 9 bits indiquant les permissions du propriétaire, du groupe et des autres utilisateurs pour contrôler l'accès au segment. Les bits d'exécution sont ignorés. Une façon simple de spécifier les permissions est d'utiliser les constantes définies dans <sys/stat.h>.

Par exemple, S_IRUSR et S_IWUSR spécifient des permissions de lecture et écriturepour le propriétaire du segment de mémoire partagée et S_IROTH et S_IWOTH spécifientdes permissions de lecture et écriture pour les autres utilisateurs.

L'appel suivant à shmget crée un nouveau segment de mémoire partagée (ou accède à un segmentexistant, si shm_key est déjà utilisé) qui peut être lu et écrit par son propriétaire mais pas parles autres utilisateurs.

int segment_id = shmget (shm_key , getpagesize () ,

IPC_CREAT | S_IRUSR | S_IWUSR);

Si l'appel se passe bien, shmget renvoie un identifiant de segment. Si le segment de mémoirepartagée existe déjà, les permissions d'accès sont vérifiées et le système s'assure que le segmentn'est pas destiné à être détruit.

Attachement et Détachement

Pour rendre le segment de mémoire partagée disponible, un procesus doit utiliser shmat («SHared Memory ATtach », attachement de mémoire partagée) en lui passant l'identifiant dusegment de mémoire partagée SHMID renvoyé par shmget. Le second argument est un pointeurqui indique où vous voulez que le segment soit mis en correspondance dans l'espace d'adressagede votre processus ; si vous passez NULL, Linux sélectionnera une adresse disponible. Le troisièmeargument est un indicateur, qui peut prendre une des valeurs suivantes

SHM_RND indique que l'adresse spécifiée par le second paramètre doit être arrondie à un multiple inférieur de la taille de page. Si vous n'utilisez pas cet indicateur, vous devez aligner le second argument de shmat sur un multiple de page vous-même.

SHM_RDONLY indique que le segment sera uniquement lu, pas écrit. Si l'appel se déroule correctement, il renvoie l'adresse du segment partagé attaché. Les processus fils créés par des appels à fork héritent des segments partagés attachés ; ils peuvent les détacher s'ils le souhaitent.

Lorsque vous en avez fini avec un segment de mémoire partagée, le segment doit être détachéen utilisant shmdt (« SHared Memory DeTach », Détachement de Mémoire Partagée) et luipassant l'adresse renvoyée par shmat. Si le segment n'a pas été libéré et qu'il s'agissait dudernier processus l'utilisant, il est supprimé. Les appels à exit et toute fonction de la familled'exec détachent automatiquement les segments.

Contrôler et Libérer la Mémoire Partagée

L'appel shmctl (« SHared Memory ConTroL », contrôle de la mémoire partagée) renvoie desinformations sur un segment de mémoire partagée et peut le modifier. Le premier paramètre estl'identifiant d'un segment de mémoire partagée.

Pour obtenir des informations sur un segment de mémoire partagée, passez IPC_STAT commesecond argument et un pointeur vers une struct shmid_ds.

Programme Exemple

Le programme du Listing 5.1 illustre l'utilisation de la mémoire partagée.

Listing 5.1 – (shm.c) – Utilisation de la Mémoire Partagée

```
1 # include <stdio .h>
  2 # include <sys / shm .h>
  3 # include <sys / stat .h>
  4 int main ()
  5 {
  6 int segment_id ;
  7 char * shared_memory ;
  8 struct shmid_ds shmbuffer ;
  9 int segment_size ;
  10 const int shared_segment_size = 0 x6400 ;
  11 /* Alloue le segment de mémoire partagée . */
  12 segment_id = shmget ( IPC_PRIVATE , shared_segment_size ,
  13 IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR );
```

```
14 /* Attache le segment de mémoire partagée . */
15 shared_memory = ( char *) shmat ( segment_id , 0, 0);
16 printf (« mémoire partagée attachée à l'adresse %p\n»,
shared_memory );
17 /* Détermine la taille du segment . */
18 shmctl ( segment_id , IPC_STAT , & shmbuffer );
19 segment_size = shmbuffer . shm_segsz ;
20 printf (« taille du segment : %d\n», segment_size );
21 /* Écrit une chaîne dans le segment de mémoire partagée . */
22 sprintf ( shared_memory , «Hello , world .»);
23 /* Détache le segment de mémoire partagée . */
24 shmdt ( shared_memory );
25 /* Réattache le segment de mémoire partagée à une adresse
différente . */
26 shared_memory = ( char *) shmat ( segment_id , ( void *) 0
x5000000 , 0);
27 printf (« mémoire partagée réattachée à l'adresse %p\n»,
shared_memory );
28 /* Affiche la chaîne de la mémoire partagée . */
29 printf («%s\n», shared_memory );
30 /* Détache le segment de mémoire partagée . */
31 shmdt ( shared_memory );
32
33 /* Libère le segment de mémoire partagée . */
34 shmctl ( segment_id , IPC_RMID , 0);
35
36 return 0;
37 }
```

Conclusion

Dans cette activité,nous avons passé en revueles techniques de gestionde la mémoire dans les programmesC. En particulier, les mécanismes de la création, de l'allocation et de la libération de la mémoireont étémis en évidence.

Evaluation

1. Ecrire les codes suivants qui illustrent des segments.

```
//mem.c

#include <stdio.h>

#include <stdlib.h>

int uig;

int ig = 5;

int func()
        {

return 0;
        }

int main()

{

int local;

int *ptr;

ptr = (int *) malloc(sizeof(int));

printf(«An address from BSS: %p\n», &uig);

printf(«An address from Data segment: %p\n», &ig);

printf(«An address from Code segment: %p\n», &func);

printf(«An address from Stack segment: %p\n», &local);

printf(«An address from Heap: %p\n», ptr);

printf(«Another address from Stack: %p\n», &ptr);

free(ptr);

return 0;
        }
```

Exécuter le code et examinerla dernière de la sortie produite, qui montre une autre adresse de la pile :

a.Pourquoi &ptr est dans le segment de la pile ?

b.Pourquoi la distance est de 4 byte entre local et &ptr ?

c.Pourquoi &ptr est plus petit que &local ?

2. Ecrire, coder, exécuter et analyser les résultats des programmes fournis en exemple.

Résumé de l'unité

Au cours de cette unité, vous avez appris à manipuler les concepts de gestion des processus, des threads et de la mémoire ; avec les techniques de création des processus en utilisant les fonctions fork et exec du système. Les techniques de création et de gestion des threads, ainsi que celles de gestion de la mémoire ont été également abordées.

Évaluation de l'unité

Vérifiez votre compréhension!

Instructions :

Cette évaluation doit être fait en individuel

Il s'agit d'un devoir à rendre

Lisez attentivement le cours et répondez aux questions suivantes

1. Ecrire, coder, exécuter et analyser les résultats du programme suivant :

```
int main()
{
Pid_t pid;
/* fork another process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, «Fork Failed»);
exit(1)
;
```

```
}
else if (pid == 0) { /* child process */
execlp(«/bin/ls», «ls», NULL);
}
else { /* parent process */
/* parent will wait for the child to complete */
wait (NULL);
printf («Child Complete»);
exit(0);
}
}
```

2. En considérant le programme suivant, décrire le statut de la mémoire aux lignes 10, 12 et 14. Quel est le résultat produit par ce programme :

```
1 #include <stdio.h>
2 int main() {
3 int i = 0;
4 int j = 2;
5 int** p1;
6 int* p2;
7 int* p3;
8 p2 = &i;
9 p3 = &j;
10 p1 = &p2;
11 j = *p3 + 1;
12 *p2 = **p1 - 1;
13 p1 -= 2;
14 *(p1[2]) = 3 * *p2;
15 printf(«i = %d, j= %d\n», i, j);
16 return 0;
17 }
```

Lectures et autres ressources

- Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001
- Linux System Programming: Talking Directly to the Kernel and C Library, Robert Love «O'Reilly Media, Inc.», 14 May 2013
- Linux Kernel Development, Robert Love, Pearson Education, 22 Jun 201

Laboratoire #1 : Manipulation de processus et interpréteur de commandes

Objectif du laboratoire :

L'objectif de ce laboratoire est de vous permettre d'expérimenter les opérations de manipulation desprocessus dans un système d'exploitation du type UNIX. Pour cela, vous allez progressivement concevoiret implémenter quelques fonctionnalités d'un interpréteur de commandes rudimentaire. Vous pouvezréaliser ce laboratoire en utilisant votre PC Linux ou à défaut une machine virtuelle.

- Vous pouvez réaliser ce laboratoire en utilisant toute machine Linux.
- Vous n'avez pas besoinde privilège root.
- Nombre de semaines pour la réalisation de ce laboratoire : 02

Première partie : Exécution d'une commande

Concevoir et implémenter un programme RunCmd qui prend en entrée une commande Unix (avec ses propres arguments et options) comme argument et exécute l'action associée à cette commande et ensuite affiche des statistiques relatives à l'exécution de cette commande.

Exemple :

```
$ RunCmd ls –al /usr/src
```

Dans cet exemple, RunCmd va invoquer la commande UNIX (Linux) ls –al /usr/src qui elle-même liste le contenu du répertoire spécifié (/usr/src) en tenant compte des options spécifiées (-al). Notez bien que $ ne fait pas partie de la commande c'est le prompt du système.

Après exécution de la commande spécifiée, RunCmd doit afficher des statistiques relatives à l'utilisation des ressources du système par l'exécution de cette commande. Ces statistiques sont :

- Le temps (wall-clock time) écoulé pour l'exécution de la commande en millisecondes
- Le temps CPU utilisé en millisecondes

- Le nombre de fois que le processus a été interrompu involontairement (par exemple pour cause d'expiration du quantum de temps, préemption par un processus de plus haute priorité, etc.)
- Le nombre de fois que le processus a libéré (cédé) volontairement la CPU (par exemple pour une attente d'une opération d'E/S, ou attente d'une ressource)
- Le nombre de défauts de pages
- Le nombre de défauts de page qui a pu être satisfait de la mémoire cache du noyau

Vous ne pouvez pas utiliser l'appel système system() pour exécuter la commande spécifiée. Vous devez scanner la ligne d'entrée de l'utilisateur pour extraire la commande, créer un nouveau processus et faites en sorte que ce processus nouvellement créé exécute la commande avec ces arguments en utilisant un appel système adéquat.

Comment obtenir de l'information et les statistiques ?

Un des objectifs secondaires de ce laboratoire est de vous permettre d'apprendre à chercher de l'information dans la documentation en ligne d'UNIX (Linux) appelée les pages man. A partir de cette documentation vous allez apprendre comment invoquer les diverses fonctionnalités du système depuis votre programme.

Par exemple, pour avoir de l'information sur la fonction fork(), il suffit d'entrer la commande :

```
$ man fork
```

Les pages man sont organisés en sections. Section 1 est dédiée pour les commandes du shell, section 2 pour les appels système et section 3 pour les routines de la librairie, etc.

Pour cette première partie du laboratoire, les appels système suivants vous seront nécessaires : fork(), execvp() (ou un de ses variantes), wait(), getrusage(), gettimeofday(), et strtok().

Deuxième partie : Interpréteur de commandes de base

L'objectif de cette partie du laboratoire est d'étendre le programme RunCmd pour en faire un interpréteur de commandes (shell). Pour cela, vous devez modifier RunCmd pour qu'il agisse comme un interpréteur de commandes : Log710Shell. Ce dernier est essentiellement un programme qui continuellement (en boucle) (1) affiche un prompt,qui est la chaine Log710H2014%> (2) attend que l'utilisateur saisisse une commande, qui peut avoir plusieurs arguments et options séparés par des espaces, (3) l'exécute de la même manière que RunCmd et (4) affiche les statistiques mentionnées ci-dessus.

Votre interpréteur de commandes doit en particulier reconnaitre et traiter deux commandes internes «built-in». Ces commandes sont implémentées par l'interpréteur de commandes lui-même et ne nécessitent pas la création d'un nouveau processus.

exit — permet de terminer l'interpréteur de commandes.

cd rep — permet de changer de répertoire de travail vers le répertoire cible rep.

A titre d'illustration, une session de votre interpréteur de commande serait comme suit :

```
% /* Au niveau du shell de votre machine virtuel, on exécute Le
programme Log710Shell*/
% Log710Shell
Log710H2014%>cat /etc/motd
    /* affiche le message du jour*/
    /* affiche les statistiques relative à la commande cat */
Log710H2014%>cd rep
    /* le repertoire courant est change vers rep */
Log710H2014%>ls
    /* Liste les fichiers du répertoire courant */
/   * affiche les statistiques relative à la commande ls */
Log710H2014%>exit
    % /* De retour au shell de votre machine */
```

Troisième partie : Interpréteur de commandes de base

L'objectif de cette troisième partie de ce laboratoire est d'améliorer votre interpréteur de commandes Log710Shell2 pour traiter les tâches en arrière-plan (background tasks). Une tâche en arrière-plan est indiquée par le caractère ('&') en fin de ligne d'entrée. Quand une tâche s'exécute en arrière-plan, Log710Shell2 ne doit pas attendre qu'elle complète mais plutôt il doit retourner immédiatement vers l'utilisateur (il affiche le numérode la tâche, l'identificateur du processus associé excitant cette tâche et affiche de nouveau le prompt pour une nouvelle commande). Notez bien que tout sortie (output) de la commande en arrière-plan va être dirigée vers le terminal et nécessairement sera mélangée avec la sortie de votre propre interpréteur de commandes et/ou les autres commandes exécutés par votre interpréteur de commandes.

Votre interpréteur de commandes Log710Shell2 doit reconnaitre et traiter la commande interne aptaches suivante. Pour chaque tâche active en arrière-plan, Il doit afficher le numéro de la tâche, l'identificateur du processus de la tâche et le nom de la commande.

aptaches – liste toutes les tâches en arrière-plan actives

Un exemple d'une session de votre interpréteur de commandes Log710Shell2 avec les tâches en arrière-plan serait comme suit :

```
% Log710Shell2
Log710H2014%> faireqlechose &
[5] 12345 /* indique numéro de la tâche en arrière-plan #5 et
    l'identificateur de processus */
```

```
Log710H2014%> aptaches
[5] 12345 faireqlqchose
    /* Affiche le numéro de la tâche, l'identificateur du
    processus et le nom de la commande des taches */
Log710H2014%> ls
    /* liste les fichiers du répertoire courant */
    /* Affiche les statistiques sur la commande ls */
Log710H2014%> exit
    %     /* Retour au prompt de la machine */
```

Si l'utilisateur essaye de sortir de Log710Shell2 avec exit avant que toutes les tâches en arrière-plan ont fini, Log710Shell2 refuse la sortie immédiate, affiche un message et attend que toutes les taches en arrière-plan soient complétées.

Évaluation et procédure de remise du laboratoire

Ce laboratoire doit se faire en individuel et il vaut 10 points réparties comme suit : Partie 1 (3 points), partie 2 (3 points) et partie 3 (2 points)

Soumettre par émail un rapport du laboratoire (2 points) et les fichiers des programmes (code source) RunCmd, Log710Shell et Log710Shell2.

Le rapport de laboratoire doit présenter essentiellement l'analyse des différentes décisions de conception pour la réalisation de ce laboratoire, les algorithmes et les structures de données utilisées dans vos programmes, les détails des appels systèmes utilisés pour chacun des programmes, une discussion des limites de l'implémentation que vous avez réalisée et bien identifier la contribution de votre travail.

Soumission :

Soumettez votre laboratoire par courriel
Sujet du email : LOG710 LAB01

Unité 4. Communication Interprocessus

Introduction à l'unité

Cette unité présente des moyens de communication interprocessus.Nous présenterons différentes façons de communiquer entre père et fils, entre des processus« sans liens » et même entre des processus s'exécutant sur des machines distinctes.La communication interprocessus (interprocess communication, IPC) consiste à transférerdes données entre les processus. Par exemple, un navigateur Internet peut demander une pageà un serveur, qui envoie alors les données HTML. Ce transfert utilise des sockets dans uneconnexion similaire à celle du téléphone. Dans un autre exemple, vous pourriez vouloir imprimerles noms des fichiers d'un répertoire en utilisant une commande du type ls l lpr. Le shell créeun processus ls et un processus lpr distincts et connecte les deux au moyen d'un tube (ou pipe)représenté par le symbole «l». Un tube permet une communication à sens unique entre deuxprocessus. Le processus ls écrit les données dans le tube et le processus lpr les lit à partir dutube.

Nous traiterons donc trois types de communication interprocessus :

- Les tubes permettent une communication séquentielle d'un processus à l'autre.
- Les files FIFO sont similaires aux tubes excepté que des processus sans lien peuvent communiquer car le tube reçoit un nom dans le système de fichiers.
- Les sockets permettent la communication entre des processus sans lien, pouvant se trouver sur des machines distinctes.

Ces types d'IPC diffèrent selon les critères suivants :

- Ils restreignent ou non la communication à des processus liés (processus ayant un ancêtre commun), à des processus partageant le même système de fichiers ou à tout ordinateur connecté à un réseau.
- Un processus communiquant n'est limité qu'à la lecture ou qu'à l'écriture de données.
- Le nombre de processus pouvant communiquer.
- Les processus qui communiquent sont synchronisés par l'IPC – par exemple, un processus lecteur s'interrompt jusqu'à ce qu'il y ait des données à lire.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Expliquer les concepts de communications interprocessus (IPC)
- Comparer3 types d'IPC
- Créer chacun des 3 types d'IPC
- Utiliserles mécanismes d'IPC pour gérer la collaboration entre processus.
- Conception et implémentation d'un ordonnanceur de processus.
- Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation
- Réaliser deux labs relatifs à la gestion de processus et de mémoire.

Laboratoire #2 : Conception et implémentation d'un ordonnanceur de processus avec multi-niveau de priorités

À la fin de cette unité, vous devriez réaliser un second lab décrit à la fin des activités.

Laboratoire #3 : Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation

À la fin de cette unité, vous devriez réaliser également un 3ème lab présenté en fin du document.

Termes clés

Communication Interprocessus (IPC): Echanges de données entre processus

Tubes (Pipes): Un tube permet une communication à sens unique d'un processus à un autre processus

FIFOs: FIFOs Similaires aux tubes, sauf que les processus n'ayant pas de liens pourront communiquer entre eux.

Sockets: Permet la communication entre plusieurs processus sur différents serveurs physiques.

Activités d'apprentissage

Activité 1: Tubes ou pipes

Tubes

Un tube est un dispositif de communication qui permet une communication à sens unique.Les données écrites sur l'« extrémité d'écriture » du tube sont lues depuis l'« extrémité de lecture». Les tubes sont des dispositifs séquentiels ; les données sont toujours lues dans l'ordre où ellesont été écrites. Typiquement, un tube est utilisé pour la communication entre deux threads d'unmême processus ou entre processus père et fils.

Dans un shell, le symbole | crée un tube. Par exemple, cette commande provoque la créationpar le shell de deux processus fils, l'un pour ls et l'autre pour less :

% ls | less

Le shell crée également un tube connectant la sortie standard du processus ls avec l'entréestandard de less. Les noms des fichiers listés par ls sont envoyés à less dans le même ordreque s'ils étaient envoyés directement au terminal.

La capacité d'un tube est limitée. Si le processus écrivain écrit plus vite que la vitesse àlaquelle le processus lecteur consomme les données, et si le tube ne peut pas contenir de donnéessupplémentaires, le processus écrivain est bloqué jusqu'à ce qu'il y ait à nouveau de la placedans le tube. Si le lecteur essaie de lire mais qu'il n'y a plus de données disponibles, il estbloqué jusqu'à ce que ce ne soit plus le cas. Ainsi, le tube synchronise automatiquement les deuxprocessus.

Créer des Tubes

Pour créer un tube, appelez la fonction pipe. Passez-lui un tableau de deux entiers. L'appelà pipe stocke le descripteur de fichier en lecture à l'indice zéro et le descripteur de fichier enécriture à l'indice un. Par exemple, examinons ce code :

```
int pipe_fds [2];

int read_fd ;

int write_fd ;

pipe ( pipe_fds );

read_fd = pipe_fds [0];

write_fd = pipe_fds [1];
```

Les données écrites via le descripteur write_fd peuvent être relues via read_fd.

Communication entre Processus Père et Fils

Un appel à pipe crée des descripteurs de fichiers qui ne sont valide qu'au sein du processusappelant et de ses fils. Les descripteurs de fichiers d'un processus ne peuvent être transmis à desprocessus qui ne lui sont pas liés ; cependant, lorsqu'un processus appelle fork, les descripteursde fichiers sont copiés dans le nouveau processus. Ainsi, les tubes ne peuvent connecter que desprocessus liés.

Dans le programme du Listing 5.7, un fork crée un nouveau processus fils. Le fils héritedes descripteurs de fichiers du tube.

Le père écrit une chaîne dans le tube et le fils la lit. Leprogramme exemple convertit ces descripteurs de fichiers en flux FILE* en utilisant fdopen.

Comme nous utilisons des flux plutôt que des descripteurs de fichiers, nous pouvons utiliser desfonctions d'entrées/sorties de la bibliothèque standard du C de plus haut niveau, comme printfet fgets.

Listing 5.7 – (pipe.c) – Utiliser un Tube pour Communiquer avec un Processus Fils

```
1 # include <stdlib .h>
2 # include <stdio .h>
3 # include <unistd .h>
4
5 /* Écrit COUNT fois MESSAGE vers STREAM , avec une pause d'une
seconde
6 entre chaque . */
7
8 void writer ( const char * message , int count , FILE * stream
)
9 {
10 for (; count > 0; --count ) {
11 /* Écrit le message vers le flux et purge immédiatement . */
12 fprintf ( stream , «%s\n», message );
13 fflush ( stream );
14 /* S'arrête un instant . */
15 sleep (1) ;
16 }
```

```
17 }
19 /* Lit des chaînes aléatoires depuis le flux
20 aussi longtemps que possible . */
21
22 void reader ( FILE * stream )
23 {
24 char buffer [1024];
25 /* Lit jusqu 'à ce que l'on atteigne la fin du flux . fgets lit
jusqu 'à
26 ce qu 'une nouvelle ligne ou une fin de fichier survienne . */
27 while (! feof ( stream )
28 && ! ferror ( stream )
29 && fgets ( buffer , sizeof ( buffer ), stream ) != NULL )
30 fputs ( buffer , stdout );
31 }
32
33 int main ()
34 {
35 int fds [2];
36 pid_t pid ;
37
38 /* Crée un tube . Les descripteurs de fichiers pour les deux
bouts du tube
39 sont placés dans fds . */
40 pipe ( fds );
41 /* Crée un processus fils . */
42 pid = fork ();
43 if ( pid == ( pid_t ) 0) {
44 FILE * stream ;
45 /* Nous sommes dans le processus fils . On ferme notre copie de
46 l'extrémité en écriture du descripteur de fichiers . */
```

```
47 close ( fds [1]) ;
48 /* Convertit le descripteur de fichier de lecture en objet FILE
49 et lit à partir de celui -ci. */
50 stream = fdopen ( fds [0] , «r»);
51 reader ( stream );
52 close ( fds [0]) ;
53 }
54 else {
55 /* Nous sommes dans le processus parent . */
56 FILE * stream ;
57 /* Ferme notre copie de l'extrémité en lecture
58 du descripteur de fichier . */
59 close ( fds [0]) ;
60 /* Convertit le descripteur de fichier d'écriture en objet FILE
61 et y écrit des données . */
62 stream = fdopen ( fds [1] , «w»);
63 writer (« Coucou .», 5, stream );
64 close ( fds [1]) ;
65 }
66
67 return 0;
68 }
```

Au début de la fonction main, fds est déclaré comme étant un tableau de deux entiers.L'appel à pipe crée un tube et place les descripteurs en lecture et en écriture dans ce tableau.

Le programme crée alors un processus fils. Après avoir fermé l'extrémité en lecture du tube, leprocessus père commence à écrire des chaînes dans le tube. Après avoir fermé l'extrémité enécriture du tube, le processus fils lit les chaînes depuis le tube.

Notez qu'après l'écriture dans la fonction writer, le père purge le tube en appelant fflush.Dans le cas contraire, les données pourraient ne pas être envoyées dans le tube immédiatement. Lorsque vous invoquez la commande ls | less, deux divisions de processus ont lieu :une pour le processus fils ls et une pour le processus fils less. Ces deux processus héritentdes descripteurs de fichier du tube afin qu'ils puissent communiquer en l'utilisant. Pour fairecommuniquer des processus sans lien, utilisez plutôt des FIFO.

Rediriger les Flux d'Entrée, de Sortie et d'Erreur Standards

Souvent, vous aurez besoin de créer un processus fils et de définir l'extrémité d'un tube commeson entrée ou sa sortie standard. Grâce à la fonction dup2, vous pouvez substituer un descripteurde fichier à un autre. Par exemple, pour rediriger l'entrée standard vers un descripteur de fichierfd, utilisez ce code :

```
dup2 (fd , STDIN_FILENO );
```

La constante symbolique STDIN_FILENO représente le descripteur de fichier de l'entrée standard,qui a la valeur 0. L'appel ferme l'entrée standard puis la rouvre comme une copie de fd de façonà ce que les deux puissent être utilisés indifféremment. Les descripteurs de fichiers substituéspartagent la même position dans le fichier et le même ensemble d'indicateurs de statut de fichier.Ainsi, les caractères lus à partir de fd ne sont pas relus à partir de l'entrée standard.Le programme du Listing 5.8 utilise dup2 pour envoyer des données d'un tube vers la commandesort2. Une fois le tube créé, le programme se divise. Le processus parent envoie quelqueschaînes vers le tube. Le processus fils connecte le descripteur de fichier en lecture du tube à sonentrée standard en utilisant dup2. Il exécute ensuite le programme sort.

Listing 5.8 – (dup2.c) – Rediriger la Sortie d'un Tube avec dup2

```
1 # include <stdio .h>
2 # include <sys / types .h>
3 # include <sys / wait .h>
4 # include <unistd .h>
5
6 int main ()
7 {
8 int fds [2];
9 pid_t pid ;
10
11 /* Crée un tube . Les descripteurs de fichiers des deux
extrémités
12 du tube sont placées dans fds . */
13 pipe ( fds );
14 /* Crée un processus fils . */
15 pid = fork ();
16 if ( pid == ( pid_t ) 0) {
```

```
17 /* Nous sommes dans le processus fils . On ferme notre copie du
18 descripteur de fichier en écriture . */
19 close ( fds [1]) ;
20 /* Connexion de l'extrémité en lecture à l'entrée standard .
*/
21 dup2 ( fds [0] , STDIN_FILENO );
22 /* Remplace le processus fils par le programme « sort «. */
23 execlp (« sort «, « sort «, 0);
24 }
25 else {
26 /* Processus père . */
27 FILE * stream ;
28 /* Ferme notre copie de l'extrémité en lecture du descripteur
. */
29 close ( fds [0]) ;
30 /* Convertit le descripteur de fichier en écriture en objet
FILE , et
31 y écrit . */
32 stream = fdopen ( fds [1] , «w»);
33 fprintf ( stream , «C'est un test .\n»);
34 fprintf ( stream , « Coucou .\n»);
35 fprintf ( stream , « Mon chien a des puces .\n»);
36 fprintf ( stream , «Ce programme est excellent .\n»);
37 fprintf ( stream , «Un poisson , deux poissons .\n»);
38 fflush ( stream );
39 close ( fds [1]) ;
40 /* Attend la fin du processus fils . */
41 waitpid (pid , NULL , 0);
42 }
44 return 0;
45 }.
```

Popen et pclose

Une utilisation courante des tubes est d'envoyer ou de recevoir des données depuis unprogramme en cours d'exécution dans un sous-processus. Les fonctions popen et pclose facilitentcette pratique en éliminant le besoin d'appeler pipe, fork, dup2, exec et fdopen.

Comparez le Listing 5.9, qui utilise popen et pclose, à l'exemple précédent (Listing 5.8).

Listing 5.9 – (popen.c) – Exemple d'Utilisation de popen

```
1 # include <stdio .h>
2 # include <unistd .h>
3
4 int main ()
5 {
6 FILE * stream = popen (« sort «, «w»);
7 fprintf ( stream , «C'est un test .\n»);
8 fprintf ( stream , « Coucou .\n»);
9 fprintf ( stream , « Mon chien a des puces .\n»);
10 fprintf ( stream , «Ce programme est excellent .\n»);
11 fprintf ( stream , «Un poisson , deux poissons .\n»);
12 return pclose ( stream );
13 }
```

L'appel à popen crée un processus fils exécutant la commande sort, ce qui remplace les appelsà pipe, fork, dup2 et execlp. Le second argument, «w», indique que ce processus désire écrireau processus fils. La valeur de retour de popen est l'extrémité d'un tube ; l'autre extrémité estconnectée à l'entrée standard du processus fils. Une fois que le processus père a terminé d'écrire,pclose ferme le flux du processus fils, attend la fin du processus et renvoie son code de retour.

Le premier argument de popen est exécuté comme s'il s'agissait d'une commande shell, dansun processus exécutant /bin/sh. Le shell recherche les programmes à exécuter en utilisant lavariable d'environnement PATH de la façon habituelle. Si le deuxième argument est «r», la fonctionrenvoie le flux de sortie standard du processus fils afin que le père puisse lire la sortie. Si le secondargument est «w», la fonction renvoie le flux d'entrée standard du processus fils afin que le pèrepuisse envoyer des données. Si une erreur survient, popen renvoie un pointeur nul.

Appelez pclose pour fermer un flux renvoyé par popen. Après avoir fermé le flux indiqué,pclose attend la fin du processus fils.

Conclusion

Cette activité présente les tubes (pipes) comme un dispositif de communication interprocessu (IPC) qui permet une communication unidirectionnelle. Nous avons décrit les constructions pour créer un tube, et comment le tuyau peut être utilisé pour la communication entre processus parents et enfants. En outre, nous avons montré comment rediriger un processus standard d'entrée, de sortie et de flux d'erreur, et l'utilisation des fonctions de popen et pclose qui permettent d'éliminer la nécessité d'invoquer pipe, fork, dup, exec et fdopen.

Évaluation

1. Pratiquer les exemples de code produits dans cette activité

Activité 2: FIFO

Une file premier entré, premier sorti (first-in, first-out, FIFO) est un tube qui dispose d'unnom dans le système de fichiers. Tout processus peut ouvrir ou fermer la FIFO; les processusraccordés aux extrémités du tube n'ont pas à avoir de lien de parenté. Les FIFO sont égalementappelés canaux nommés.

Vous pouvez créer une FIFO via la commande mkfifo. Indiquez l'emplacement où elle doitêtre créée sur la ligne de commande. Par exemple, créez une FIFO dans /tmp/fifo en invoquantces commandes :

```
% mkfifo /tmp/fifo

% ls -l /tmp/fifo

prw-rw-rw- 1 samuel users 0 Jan 16 14:04 /tmp/fifo
```

Le premier caractère affiché par ls est p ce qui indique que le fichier est en fait une FIFO (canalnommé, named pipe). Dans une fenêtre, lisez des données depuis la FIFO en invoquant cettecommande :

```
% cat < /tmp/fifo
```

Dans une deuxième fenêtre, écrivez dans la FIFO en invoquant cela :

```
% cat > /tmp/fifo
```

Puis, saisissez du texte. À chaque fois que vous appuyez sur Entrée, la ligne de texte est envoyée dans la FIFO et apparaît dans la première fenêtre. Fermez la FIFO en appuyant sur Ctrl+Ddans la seconde fenêtre. Supprimez la FIFO avec cette commande :

```
% rm /tmp/fifo
```

Créer une FIFO

Pour créer une FIFO par programmation, utilisez la fonction mkfifo. Le premier argumentest l'emplacement où créer la FIFO; le second paramètre spécifie les permissions du propriétairedu tube, de son groupe et des autres utilisateurs. Comme un tube doit avoir un lecteur etun écrivain, les permissions doivent comprendre des autorisations en lecture et en écriture. Sile tube ne peut pas être créé (par exemple, si un fichier possédant le même nom existe déjà),mkfifo renvoie -1. Incluez <sys/types.h> et <sys/stat.h> si vous appelez mkfifo.

Accéder à une FIFO

L'accès à une FIFO se fait de la même façon que pour un fichier ordinaire. Pour communiquervia une FIFO, un programme doit l'ouvrir en écriture. Il est possible d'utiliser des fonctions d'E/Sde bas niveau (open, write, read, close, etc.) ou des fonctions d'E/S de la bibliothèque C (fopen, fprintf, fscanf, fclose, etc.).

Par exemple, pour écrire un tampon de données dans une FIFO en utilisant des routines debas niveau, vous pourriez procéder comme suit :

```
int fd = open ( fifo_path , O_WRONLY );

write (fd , data , data_length );

close (fd);
```

Pour lire une chaîne depuis la FIFO en utilisant les fonctions d'E/S de la bibliothèque C, vouspourriez utiliser ce code :

```
FILE * fifo = fopen ( fifo_path , «r»);

fscanf (fifo , «%s», buffer );

fclose ( fifo );
```

Une FIFO peut avoir plusieurs lecteurs ou plusieurs écrivains. Les octets de chaque écrivain sontécrits de façon atomique pour une taille inférieure à PIPE_BUF (4Ko sous Linux). Les paquetsd'écrivains en accès simultanés peuvent être entrelacés. Les mêmes règles s'appliquent à deslectures concurrentes.

Différences avec les Canaux Nommés Windows

Les tubes des systèmes d'exploitation Win32 sont très similaires aux tubes Linux (consultezla documentation de la bibliothèque Win32 pour plus de détails techniques). Les principales différencesconcernent les canaux nommés, qui, sous Win32, fonctionnent plus comme des sockets.

Les canaux nommés Win32 peuvent connecter des processus d'ordinateurs distincts connectésvia un réseau. Sous Linux, ce sont les sockets qui sont utilisés pour ce faire. De plus, Win32permet de multiples connexions de lecteur à écrivain sur un même canal nommé sans que lesdonnées ne soient entrelacées et les tubes peuvent être utilisés pour une communication à doublesens.

Conclusion

Cette activité a présenté un FIFO (également dénommé «tubes nommés») en tant que dispositif IPC qui permet la communication entre des processus indépendants. Nous décrivons les constructions pour la création et l'accès à une FIFO et mis en évidence les différences entre Linux et Windows.

Évaluation

1. Pratiquer les exemples de code produits dans cette activité

Activité 3: Sockets

Un socket est un dispositif de communication bidirectionnel pouvant être utilisé pour communiqueravec un autre processus sur la même machine ou avec un processus s'exécutant surd'autres machines. Les sockets sont la seule forme de communication interprocessus dont noustraiterons dans ce chapitre qui permet la communication entre processus de différentes machines.

Les programmes Internet comme Telnet, rlogin, FTP, talk et le World Wide Web utilisent dessockets.

Par exemple, vous pouvez obtenir une page depuis un serveur Web en utilisant le programmeTelnet car tous deux utilisent des sockets pour la communication via le réseau4. Pour ouvrir uneconnexion vers un serveur Web dont l'adresse est www.codesourcery.com, utilisez la commandetelnet www.codesourcery.com 80. La constante magique 80 demande une connexion au serveurWeb de www.codesourcery.com plutôt qu'à un autre processus. Essayez d'entrer GET /une fois la connexion établie. Cela envoie un message au serveur Web à travers le socket, celui-cirépond en envoyant la source HTML de la page d'accueil puis en fermant la connexion – parexemple :

```
% telnet www.codesourcery.com 80

Trying 206.168.99.1...

Connected to merlin.codesourcery.com (206.168.99.1).

Escape character is «^]».

GET /

<html>

<head>

<meta http-equiv=»Content-Type» content=»text/html;
charset=iso-8859-1»>

...
```

Concepts relatifs aux Sockets

Lorsque vous créez un socket, vous devez indiquer trois paramètres : le style de communication,l'espace de nommage et le protocole.

Un style de communication contrôle la façon dont le socket traite les données transmises etdéfinit le nombre d'interlocuteurs. Lorsque des données sont envoyées via le socket, elles sontdécoupées en morceaux appelés paquets. Le style de communication détermine comment sontgérés ces paquets et comment ils sont envoyés de l'émetteur vers le destinataire.

Le style connexion garantit la remise de tous les paquets dans leur ordre d'émission. Si des paquets sont perdus ou mélangés à cause de problèmes dans le réseau, le destinataire demande automatiquement leur retransmission à l'émetteur. Un socket de type connexion ressemble à un appel téléphonique : les adresses de l'émetteur et du destinataire sont fixées au début de la communication lorsque la connexion est établie.

Le style datagramme ne garantit pas la remise ou l'ordre d'arrivée des paquets. Des paquets peuvent être perdus ou mélangés à cause de problèmes dans le réseau. Chaque paquet doit être associé à sa destination et il n'y a aucune garantie quant à sa remise. Le système ne garantit que le « meilleur effort » (best effort), des paquets peuvent donc être perdus ou être remis dans un ordre différent de leur émission.

Un socket de style datagramme se comporte plus comme une lettre postale. L'émetteurspécifie l'adresse du récepteur pour chaque message.

L'espace de nommage d'un socket spécifie comment les adresses de socket sont écrites. Uneadresse de socket identifie l'extrémité d'une connexion par socket. Par exemple, les adresses desocket dans « l'espace de nommage local » sont des noms de fichiers ordinaires. Dans « l'espacede nommage Internet », une adresse de socket est composée de l'adresse Internet (égalementappelée adresse IP) d'un hôte connecté au réseau et d'un numéro de port. Le numéro de portpermet de faire la distinction entre plusieurs sockets sur le même hôte.

Un protocole spécifie comment les données sont transmises. Parmi ces protocoles, on peutciter TCP/IP ; les deux protocoles principaux utilisés pour Internet, le protocole réseau AppleTalk; et le protocole de communication locale d'UNIX. Toutes les combinaisons de styles,d'espace de nommage et de protocoles ne sont pas supportées.

Appels Système

Les sockets sont plus flexibles que les techniques de communication citées précédemment.

Voici les appels systèmes utiles lors de l'utilisation de sockets :

- socket – Crée un socket.
- close – Détruit un socket.
- connect – Crée une connexion entre deux sockets.
- bind – Associe un socket serveur à une adresse.
- listen – Configure un socket afin qu'il accepte les connexions.
- accept – Accepte une connexion et crée un nouveau socket pour celle-ci.

Les sockets sont représentés par des descripteurs de fichiers.

Créer et Détruire des Sockets

Les fonctions socket et close créent et détruisent des sockets, respectivement. Lorsque vouscréez un socket, spécifiez ses trois propriétés : espace de nommage, style de communication etprotocole. Pour le paramètre d'espace de nommage, utilisez les constantes commençant par PF_(abréviation de « protocol family », famille de protocoles). Par exemple, PF_LOCAL ou PF_UNIXspécifient l'espace de nommage local et PF_INET correspond à l'espace de nommage Internet.

Pour le paramètre du style de communication, utilisez les constantes commençant par SOCK_. SOCK_STREAM demande un socket de style connexion, SOCK_DGRAM un socket de style datagramme.

Le troisième paramètre, le protocole, spécifie le mécanisme de bas niveau utilisé pour transmettreet recevoir des données. Chaque protocole est valide pour une combinaison d'espace denommage et de type de communication particulière. Comme il y a souvent un protocole adaptépour chaque paire, indiquer 0 sélectionne généralement le bon protocole. Si l'appel à socket sedéroule correctement, il renvoie un descripteur de fichier pour le socket. Vous pouvez lire ouécrire sur un socket en utilisant read, write, etc. comme avec les autres descripteurs de fichiers.Lorsque vous en avez fini avec le socket, appelez close pour le supprimer.

Appeler connect

Pour créer une connexion entre deux sockets, le client appelle connect, en lui passant l'adressed'un socket serveur auquel se connecter. Un client est un processus initiant une connexion etun serveur est un processus en attente de connexions. Le client appelle connect pour initierune connexion depuis un socket local, dont le descripteur est passé en premier argument, versle socket serveur spécifié par le deuxième argument. Le troisième argument est la longueur, enoctets, de la structure d'adresse pointée par le second argument. Les formats d'adresse de socketsdiffèrent selon l'espace de nommage du socket.

Envoyer des Informations

Toutes les techniques valides pour écrire dans un descripteur de fichier le sont également pourécrire dans un socket. Reportez-vous à l'Appendice B pour une présentation des fonctions d'E/Sde bas niveau Linux et de certains des problèmes ayant trait à leur utilisation. La fonction send,qui est spécifique auxdescripteurs de fichiers socket, est une alternative à write avec quelqueschoix supplémentaires ; reportez-vous à sa page de manuel pour plus d'informations.

Serveurs

Le cycle de vie d'un serveur consiste à créer un socket de type connexion, le lier à une adresse,appeler listen pour permettre au socket d'accepter des connexions, appeler accept régulièrementpour accepter les connexions entrantes, puis fermer le socket. Les données ne sont pas lues etécrites directement via le socket serveur ; au lieu de cela, chaque fois qu'un programme accepteune nouvelle connexion, Linux crée un socket séparé utilisé pour le transfert de données via cetteconnexion. Dans cette section, nous introduirons bind, listen et accept.

Une adresse doit être liée au socket serveur en utilisant bind afin que les clients puissent letrouver. Son premier argument est le descripteur de fichier du socket. Le second argument estun pointeur vers une structure d'adresse de socket ; son format dépend de la famille d'adressedu socket. Le troisième argument est la longueur de la structure d'adresse en octets. Une foisqu'une adresse est liée à un socket de type connexion, il doit invoquer listen pour indiquerqu'il agit en tant que serveur. Son premier argument est le descripteur de fichier du socket. Lesecond spécifie combien de connexions peuvent être mise en file d'attente. Si la file est pleine,les connexions supplémentaires seront refusées. Cela ne limite pas le nombre total de connexionsqu'un serveur peut gérer ; cela limite simplement le nombre de clients tentant de se connecterqui n'ont pas encore été acceptés.

Un serveur accepte une demande de connexion d'un client en invoquant accept. Son premierargument est le descripteur de fichier du socket. Le second pointe vers une structure d'adressede socket, qui sera renseignée avec l'adresse de socket du client. Le troisième argument estla longueur, en octets, de la structure d'adresse de socket. Le serveur peut utiliser l'adressedu client pour déterminer s'il désire réellement communiquer avec le client. L'appel à acceptcrée un nouveau socket pour communiquer avec le client et renvoie le descripteur de fichiercorrespondant. Le socket serveur original continue à accepter de nouvelles connexions de clients.

Pour lire des données depuis un socket sans le supprimer de la file d'attente, utilisez recv. Cettefonction prend les mêmes arguments que read ainsi qu'un argument FLAGS supplémentaire. Passerla valeur MSG_PEEK permet de lire les données sans les supprimer de la file d'attente.

Sockets Locaux

Les sockets mettant en relation des processus situés sur le même ordinateur peuvent utiliserl'espace de nommage local représenté par les constantes PF_LOCAL et PF_UNIX. Ils sont appeléssockets locaux ou sockets de domaine UNIX. Leur adresse de socket, un nom de fichier, n'estutilisée que lors de la création de connexions.

Le nom du socket est spécifié dans une struct sockaddr_un. Vous devez positionner le champsun_family à AF_LOCAL, qui représente un espace de nommage local. Le champ sun_path spécifiele nom de fichier à utiliser et peut faire au plus 108 octets de long. La longueur réelle de la structsockaddr_un doit être calculée en utilisant la macro SUN_LEN. Tout nom de fichier peut être utilisé,mais le processus doit avoir des autorisations d'écriture sur le répertoire, qui permettent l'ajoutde fichiers. Pour se connecter à un socket, un processus doit avoir des droits en lecture sur lefichier. Même si différents ordinateurs peuvent partager le même système de fichier, seuls desprocessus s'exécutant sur le même ordinateur peuvent communiquer via les sockets de l'espacede nommage local.Le seul protocole permis pour l'espace de nommage local est 0.Comme il est stocké dans un système de fichiers, un socket local est affiché comme un fichier.Par exemple, remarquez le s du début :

```
% ls -l /tmp/socket

srwxrwx--x 1 user group 0 Nov 13 19:18 /tmp/socket
```

Appelez unlink pour supprimer un socket local lorsque vous ne l'utilisez plus.

Un Exemple Utilisant les Sockets Locaux

Nous allons illustrer l'utilisation des sockets au moyen de deux programmes. Le programmeserveur, Listing 5.10, crée un socket dans l'espace de nommage local et attend des connexions.Lorsqu'il reçoit une connexion, il lit des messages au format texte depuis celle-ci et les affichejusqu'à ce qu'elle soit fermée. Si l'un de ces messages est « quit », le programme serveur supprimele socket et se termine. Le programme socket-server prend en argument de ligne de commandele chemin vers le socket.

Listing 5.10 – (socket-server.c) – Serveur Utilisant un Socket Local

```
1 # include <stdio .h>
2 # include <stdlib .h>
3 # include <string .h>
4 # include <sys / socket .h>
5 # include <sys /un.h>
6 # include <unistd .h>
7
8 /* Lit du texte depuis le socket et l'affiche . Continue jusqu
'à ce que
9 le socket soit fermé . Renvoie une valeur différente de zéro si
le client
10 a envoyé un message « quit «, zéro sinon . */
11
12 int server ( int client_socket )
13 {
14 while (1) {
15 int length ;
16 char * text ;
17
18 /* Commence par lire la longueur du message texte depuis le
socket .
19 Si read renvoie zéro , le client a fermé la connexion . */
20 if ( read ( client_socket , & length , sizeof ( length )) ==
0)
21 return 0;
22 /* Alloue un tampon pour contenir le texte . */
23 text = ( char *) malloc ( length );
24
25 /* Lit le texte et l'affiche . */
26 read ( client_socket , text , length );
```

```
27 printf («%s\n», text );
28 /* Si le client a envoyé le message « quit «, c'est fini . */
29 if (! strcmp (text , « quit «)) {
30 /* Libère le tampon . */
31 free ( text );
32 return 1;
33 }
34 /* Libère le tampon . */
35 free ( text );
36 }
37 }
38
39 int main ( int argc , char * const argv [])
40 {
41 const char * const socket_name = argv [1];
42 int socket_fd ;
43 struct sockaddr_un name ;
44 int client_sent_quit_message ;
45
46 /* Crée le socket . */
47 socket_fd = socket ( PF_LOCAL , SOCK_STREAM , 0);
48 /* Indique qu 'il s'agit d'un serveur . */
49 name . sun_family = AF_LOCAL ;
50 strcpy ( name . sun_path , socket_name );
51 bind ( socket_fd , ( struct sockaddr *) &name , SUN_LEN (&
name ));
52 /* Se met en attente de connexions . */
53 listen ( socket_fd , 5);
54
55 /* Accepte les connexions de façon répétée , lance un server
() pour traiter
```

```
56 chaque client . Continue jusqu 'à ce qu 'un client
57 envoie un message « quit «. */
58 do {
59 struct sockaddr_un client_name ;
60 socklen_t client_name_len ;
61 int client_socket_fd ;
62
63 /* Accepte une connexion . */
64 client_socket_fd =
65 accept ( socket_fd , ( struct sockaddr *) & client_name , &
client_name_len );
66 /* Traite la connexion . */
67 client_sent_quit_message = server ( client_socket_fd );
68 /* Ferme notre extrémité . */
69 close ( client_socket_fd );
70 }
71 while (! client_sent_quit_message );
72
73 /* Supprime le fichier socket . */
74 close ( socket_fd );
75 unlink ( socket_name );
76
77 return 0;
78 }
```

Le programme client, Listing 5.11, se connecte à un socket local et envoie un message. Lechemin vers le socket et le message sont indiqués sur la ligne de commande.

Listing 5.11 – (socket-client.c) – Client Utilisant un Socket Local

```
1 # include <stdio .h>
2 # include <string .h>
3 # include <sys / socket .h>
4 # include <sys /un.h>
5 # include <unistd .h>
6
7 /* Écrit TEXT vers le socket indiqué par le descripteur
SOCKET_FD . */
8
9 void write_text ( int socket_fd , const char * text )
10 {
11 /* Écrit le nombre d'octets de la chaîne , y compris l'octet
12 nul de fin . */
13 int length = strlen ( text ) + 1;
14 write ( socket_fd , & length , sizeof ( length ));
15 /* Écrit la chaîne . */
16 write ( socket_fd , text , length );
17 }
18
19 int main ( int argc , char * const argv [])
20 {
21 const char * const socket_name = argv [1];
22 const char * const message = argv [2];
23 int socket_fd ;
24 struct sockaddr_un name ;
25
26 /* Crée le socket . */
27 socket_fd = socket ( PF_LOCAL , SOCK_STREAM , 0);
```

```
28 /* Stocke le nom du serveur dans l'adresse du socket . */
29 name . sun_family = AF_LOCAL ;
30 strcpy ( name . sun_path , socket_name );
31 /* Se connecte au socket . */
32 connect ( socket_fd , ( struct sockaddr *) &name , SUN_LEN (&
name ));
33 /* écrit le texte de la ligne de commande vers le socket . */
34 write_text ( socket_fd , message );
35 close ( socket_fd );
36 return 0;
37 }
```

Avant que le client n'envoie le message, il envoie sa longueur contenue dans la variable entièrelength. De même, le serveur lit la longueur du texte en lisant une variable entière depuis le socket.Cela permet au serveur d'allouer un tampon de taille adéquate pour contenir le message avantde le lire.

Pour tester cet exemple, démarrez le programme serveur dans une fenêtre. Indiquez le cheminvers le socket – par exemple, /tmp/socket.

```
% ./socket-server /tmp/socket
```

Dans une autre fenêtre, lancez plusieurs fois le client en spécifiant le même socket ainsi que desmessages à envoyer au client :

% ./socket-client /tmp/socket «Coucou.»

% ./socket-client /tmp/socket «Ceci est un test.»

Le programme serveur reçoit et affiche les messages. Pour fermer le serveur, envoyez le message« quit » depuis un client :

```
% ./socket-client /tmp/socket «quit»
```

Le programme serveur se termine alors.

Sockets Internet

Les sockets de domaine UNIX ne peuvent être utilisés que pour communiquer entre deuxprocessus s'exécutant sur le même ordinateur. Les sockets Internet, par contre, peuvent êtreutilisés pour connecter des processus s'exécutant sur des machines distinctes connectées par unréseau.

Les sockets connectant des processus via Internet utilisent l'espace de nommage Internetreprésenté par PF_INET. Le protocole le plus courant est TCP/IP. L'Internet Protocol (IP),un protocole de bas niveau transporte des paquets sur Internet, en les fragmentant et lesréassemblant si nécessaire. Il ne garantit que le « meilleur effort » de remise, des paquetspeuvent donc disparaître ou être mélangés durant le transport. Tous les ordinateurs participantssont définis en utilisant une adresse IP unique. Le Transmission Control Protocol (TCP), quis'appuie sur IP, fournit un transport fiable orienté connexion. Il permet d'établir des connexionssemblables aux connexions téléphoniques entre des ordinateurs et assure que les données sontremises de façon fiable et dans l'ordre.

Les adresses de sockets Internet comprennent deux parties : un numéro de machine et unnuméro de port. Ces informations sont stockées dans une variable de type struct sockaddr_in.

Positionnez le champ sin_family à AF_INET pour indiquer qu'il s'agit d'une adresse de l'espacede nommage Internet. Le champ sin_addr stocke l'adresse Internet de la machine cible sousforme d'une adresse IP entière sur 32 bits. Un numéro de port permet de faire la distinctionentre plusieurs sockets d'une machine donnée. Comme des machines distinctes peuvent stockerles octets de valeurs multioctets dans des ordres différents, utilisez htons pour représenter lenuméro de port dans l'ordre des octets défini par le réseau. Consultez la page de manuel de ippour plus d'informations.

Pour convertir des noms d'hôtes, des adresses en notation pointée (comme 10.0.0.1) oudes noms DNS (comme www.codesourcery.com) adresse IP sur 32 bits, vous pouvez utilisergethostbyname. Cette fonction renvoie un pointeur vers une structure struct hostent ; le champh_addr contient l'IP de l'hôte. Reportez-vous au programme exemple du Listing 5.12 qui illustre l'utilisation de sockets de domaine Internet. Le programme rapatriela page d'accueil du serveur Web dont le nom est passé sur la ligne de commande.

Listing 5.12 – (socket-inet.c) – Lecture à partir d'un Serveur WWW

```
1 # include <stdlib .h>
2 # include <stdio .h>
3 # include <netinet /in.h>
4 # include <netdb .h>
5 # include <sys / socket .h>
6 # include <unistd .h>
7 # include <string .h>
8
9 /* Affiche le contenu de la page d'accueil du serveur
correspondant
```

```
10 au socket . */
11
12 void get_home_page ( int socket_fd )
13 {
14 char buffer [10000];
15 ssize_t number_characters_read ;
16 /* Envoie la commande HTTP GET pour la page d'accueil . */
17 sprintf ( buffer , « GET /\n»);
18 write ( socket_fd , buffer , strlen ( buffer ));
19
20 /* Lit à partir du socket . L'appel à read peut ne pas
renvoyer toutes
21 les données en une seule fois , on continue de lire jusqu 'à
ce qu 'il
22 n'y ait plus rien . */
23 while (1) {
24 number_characters_read = read ( socket_fd , buffer , 10000) ;
25 if ( number_characters_read == 0)
26 return ;
27 /* Ecrit les données vers la sortie standard . */
28 fwrite ( buffer , sizeof ( char ), number_characters_read ,
stdout );
29 }
30 }
31
32 int main ( int argc , char * const argv [])
33 {
34 int socket_fd ;
35 struct sockaddr_in name ;
36 struct hostent * hostinfo ;
37
```

```
38 /* Crée le socket . */
39 socket_fd = socket ( PF_INET , SOCK_STREAM , 0);
40 /* Place le nom du serveur dans l'adresse du socket . */
41 name . sin_family = AF_INET ;
42 /* Convertit la chaîne en adresse IP sur 32 bits . */
43 hostinfo = gethostbyname ( argv [1]) ;
44 if ( hostinfo == NULL )
45 return 1;
46 else
47 name . sin_addr = *(( struct in_addr *) hostinfo -> h_addr );
48 /* Les serveurs Web utilisent le port 80. */
49 name . sin_port = htons (80) ;
50
51 /* Se connecte au serveur Web . */
52 if ( connect ( socket_fd , ( struct sockaddr *) &name ,
53 sizeof ( struct sockaddr_in )) == -1) {
54 perror (« connect «);
55 return 1;
56 }
57 /* Récupère la page d'accueil du serveur . */
58 get_home_page ( socket_fd );
59 return 0;
60 }
```

Ce programme lit le nom du serveur Web à partir de la ligne de commande (pas l'URL– c'est-à-dire sans le « http :// »). Il appelle gethostbyname pour traduire le nom d'hôte enadresse IP numérique puis connecte un socket de type connexion (TCP) au port 80 de cet hôte.
Les serveurs Web parlent l'Hypertext Transport Protocol (HTTP), donc le programme émet lacommande HTTP GET et le serveur répond en envoyant le texte correspondant à la page d'accueil.

Par exemple, pour rapatrier la page d'accueil du site Web www.codesourcery.com, invoquezla commande suivante :

```
% ./socket-inet www.codesourcery.com

<html>

<meta http-equiv=»Content-Type» content=»text/html;
charset=iso-8859-1»>

...
```

Couples de Sockets

Comme nous l'avons vu précédemment, la fonction pipe crée deux descripteurs de fichierschacune étant l'extrémité d'un tube. Les tubes sont limités car les descripteurs de fichiersdoivent être utilisés par des processus liés et car la communication est unidirectionnelle. Lafonction socketpair crée deux descripteurs de fichiers pour deux sockets connectés sur le mêmeordinateur. Ces descripteurs de fichiers permettent une communication à double sens entredes processus liés. Ses trois premiers paramètres sont les mêmes que pour l'appel socket : ilsindiquent le domaine, le type de connexion et le protocole. Le dernier paramètre est un tableaude deux entiers, qui sera renseigné avec les descripteurs de fichiers des deux sockets, comme pourpipe. Lorsque vous appelez socketpair, vous devez utiliser PF_LOCAL comme domaine.

Conclusion

Cette activité a présenté les sockets en tant que dispositif de communication bidirectionnel qui peut être utilisé pour communiquer avec un autre processus sur la même machine ou en cours d'exécution sur d'autres machines. Différents concepts liés ont été décrits et démontrés, y compris les appels système impliquant sockets, les serveurs et le cycle de vie du serveur, sockets de domaine de l'internet, et une paire de sockets.

Évaluation

1. Pratiquer les exemples de code produits dans cette activité

Résumé de l'unité

Cette unité a présenté trois types de dispositifs de communication interprocessus : les tubes, les FIFO et

Les Sockets. Un tube permetune communication unidirectionnelle entre deux processus connexes. Les FIFOs sont semblables aux tubes, à l'exception que des processus indépendants peuvent communiquer grâce à l'utilisation d'un nom dans le système de fichiers. Les sockets permettent la prise en charge de la communication entre processus indépendants, même sur des ordinateurs différents. Les concepts derrière chaque dispositif de communicationont été décrits, et les constructions pour la création et la manipulation de chaque dispositif ont également été soulignées.

Évaluation de l'unité

Vérifiez votre compréhension!

Instructions

Cette évaluation doit être fait en individuel

Il s'agit d'un devoir à rendre

Lisez attentivement le cours et répondez aux questions suivantes

1. Considérons le codedonné ci-dessousqui illustre le problèmedetampon borné, qui modélise leparadigme deprocessuscoopérant. Procédédeproductionqui produitl'informationqui est consomméepar un processus consommateur.Ecrireun ensemble de programmes, qui utilisedes tubes (Pipes), des FIFO, et des mécanismesde SocketsIPCrespectivementcomme méthodederésolution du problèmedetampon borné.

```
//Shared data

#define BUFFER_SIZE 10

Typedef struct {

} item;

item buffer[BUFFER_SIZE];

int in = 0;

int out = 0;

// code for Insert function

while (true) {

/* Produce an item */

while (((in + 1) % BUFFER SIZE) == out) ; /*do nothing no

free

buffers */

buffer[in] = item;

in = (in + 1) % BUFFER SIZE;

{

//code for remove function

while (true) {
```

```
        while (in == out) ; // do nothing nothing

        to consume

        // remove an item from the buffer

        item = buffer[out];

        out = (out + 1) % BUFFER SIZE;

        return item;

        {
```

Lectures et autres ressources

- Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001
- UNIX Systems Programming: Communication, Concurrency, and Threads, By Kay A. Robbins, Steven Robbins, Prentice Hall Professional, 2003
- Interprocess Communications in Linux, By John Shapley Gray, Prentice Hall Professional, 2003
- UNIX Network Programming: Interprocess communications, Volume 2 , W. Richard Stevens Prentice Hall PTR, 1999

Évaluation du cours

Les évaluations«sommatives» sont définiespour permettre aux tuteursd'évaluerà quel pointlesapprenantsont comprisun cours. C'est essentielque les travaux, les capacités et les progrèsdes apprenantssoientsuivis et évaluéstout au long duprocessus d'apprentissage, il est également impératifque les instructeurs sûrs que lesapprenantsontappris tout au long du processus. La plupart du temps, c'estl'évaluationsommativequi est utilisée pourdéterminer les noteset les orientations futurespour les étudiants. Plus précisément, l'évaluation sommativea lieu après que la réalisation complète de la formation et elle fournit des informationset commentairesqui résumentle processus d'enseignementet d'apprentissage.

Dans ce cours, il y a beaucoup d'évaluation qui permettent à l'étudiant de s'auto-évaluerprogressivement sur la façon dont ils ont comprisle sujet. Et la charge de travail est suffisamment importante. L'évaluation globale du cours est donc composé des évaluations des unités (où il apparait la mention : « il s'agit d'un devoir à rendre ») et des labs : lab#1, lab#2, et lab#3.

Résumé du Module

Ce module d'acquérir les compétences dans les domaines suivante qui composent essentiellement la programmation système qui est une activité importante des ingénieurs et administrateurs systèmes et réseaux :

- Librairie C et appels systèmes d'E/S
- Programmation shell et Code assembleur en ligne
- Gestion des Processus, Threads et de la mémoire
- Communications interprocessus

Laboratoire #2 : Conception et implémentation d'un ordonnanceur de processus avec multi-niveau de priorités

Objectif du laboratoire

L'objectif de ce laboratoire est de concevoir et implémenter un ordonnanceur de processus avec quatre-niveaux de priorités sous contraintes de ressources disponibles limitées.

Un code de programme en Clog710h14process.c est fourni en annexe.

Nombre de semaines pour la réalisation de ce laboratoire : 03

Ordonnanceur de processus à quatre niveaux de priorité

L'ordonnanceur qu'on veut implémenter dans ce laboratoire utilise quatre niveaux de priorités en distinguant deux types de processus :

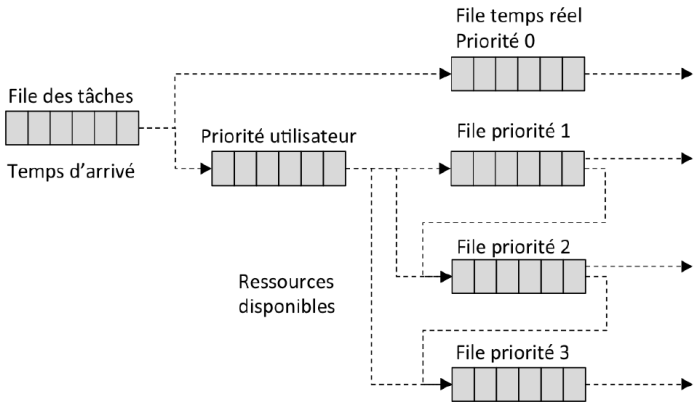
- Les processus temps réel ont la plus haute priorité. Ils doivent être exécutés immédiatement selon la logique « premier arrivé, premier servi » (FCFS). Ces processus une fois lancés ne sont pas interrompus. L'ordonnanceur les laisse s'exécuter jusqu'à la fin (ordonnanceur temps réel - FCFS).
- Les processus utilisateurs normaux sont ordonnancés selon trois niveaux de priorités avec feedback. Le quantum de temps de base est une 1 seconde qui est également le quantum de l'ordonnanceur de feedback (ordonnanceur utilisateur – Multi niveau avec retour).

L'ordonnanceur doit intégrer ces deux ordonnanceurs (temps réel et utilisateur) dans un système unique qui gère tous les processus.

L'ordonnanceur doit maintenir deux files de soumissions : une file « temps réel » et une file « priorité utilisateur ». Ces files sont alimentées à partir d'une liste d'ordonnancement de tâches ou jobs. Cette liste est examinée à chaque pas d'exécution de l'ordonnanceur (chaque quantum de temps) et les tâches qui seront arrivées sont transférées dans la file de soumission correspondante. Les files de soumissions sont alors examinées : tout processus temps réel est exécuté jusqu'à la fin en interrompant (préemption) tout autre processus en cours d'exécution.

La file « temps réel » doit être vide avant que l'ordonnanceur utilisateur puisse être réactivé. Toutes les tâches utilisateurs dans la file utilisateur qui peuvent s'exécuter compte tenu des ressources disponibles sont transférées dans la file de priorité appropriée. Le fonctionnement normal d'une file de feedback est d'accepter les tâches de plus haute priorité et décrémenter la priorité après chaque quantum complété. L'ordonnanceur peut accepter les tâches de plus basse priorité en les insérant dans la file appropriée. Ceci permet de simuler un ordonnanceur du type round-robin si toutes les tâches acceptées sont de priorité minimale.

Quand les tâches prêtes de plus haute priorité sont complétées, l'ordonnanceur utilisateur reprend en démarrant ou en continuant le processus en tête de la file de plus haute priorité non vide. Au prochain pas d'exécution (quantum), s'il y a d'autres tâches de la même ou d'une plus haute priorité qui sont prêtes, la tâche courante est suspendue (ou terminée et ses ressources retournées). La logique est illustrée par la figure.



Contraintes de ressources

On suppose que le système est doté des ressources suivantes : 2 imprimantes, 1 scanneur, 1 modem, 2 CD.

Les processus de basse priorité peuvent utiliser n'importe lesquelles de ces ressources. L'ordonnanceur est informé des besoins en ressources des processus une fois que le processus est soumis. L'ordonnanceur garantit que chaque ressource requise par un processus est exclusivement disponible pour le processus durant sa durée de vie, à partir du moment où il est transféré à la liste de priorité approprié jusqu'à la fin de son exécution.

Les processus temps réel n'utilisent pas de ressources d'entrée/sortie.

Processus

Les processus gérés par l'ordonnanceur dans ce laboratoire sont simulés en créant de nouveaux processus. Ceux-ci sont des processus génériques qui sont créés en exécutant un programme fournis (log710h14process.c) avec l'énoncé du laboratoire. Ce programmeest utilisé pour tous les processus, quelle que soit la priorité. Le programme « Log710h14process.c » affiche, à chaque seconde, son ID, un message et son état (suspendu, continué, terminé, etc.). Le processus se termine après 20 secondes s'il n'est pas terminé par l'ordonnanceur (temps

d'exécution expiré).

Le cycle de vie du processus est comme suit :

- Le processus est initialement soumis à l'ordonnanceur via une liste de processus qui désigne le temps d'arrivée, la priorité, le temps d'exécution, et les ressources requises.
- Un processus est prêt quand il est arrivé et toutes les ressources requises sont disponibles.
- Les processus « temps réel » en attente d'exécution sont ordonnancés en FCFS.
- Quand les ressources requises par un processus utilisateur deviennent disponibles, ce processus est transféré vers la file d'attente de priorité appropriée pour être ordonnancé par l'ordonnanceur utilisateur et l'état des ressources disponibles est mis à jour.
- Quand un processus est créé et lancé, fork et exec(''log710h14proc'',…), l'ordonnanceur affiche les paramètres du processus (ID du processus, priorité, temps d'exécution restant, ressources requises) avant d'exécuter la commande « exec ».
- Un processus « temps réel » s'exécute jusqu'à sa terminaison. L'ordonnanceur le supprime ('kill') quand son temps d'exécution expire en lui envoyant un signal (SIGINT).
- Un processus utilisateur est autorisé à s'exécuter pendant un pas d'ordonnancement (un quantum de temps) avant d'être suspendu par l'ordonnanceur (signal SIGTSTP) ou terminé (SIGINT) si son temps d'exécution est expiré. Si le processus utilisateur est suspendu, sa priorité est décrémentée (quand c'est possible) et il est remis dans la file de priorité appropriée. Pour synchroniser les sorties entre l'ordonnanceur et le processus fils, l'ordonnanceur doit attendre que le processus réponde à un signal SIGINT ou SIGTSTP (waitpid(pid, …)).
- Pourvu qu'il n'y ait pas de processus temps réel en attente, le processus utilisateur de plus haute priorité dans les files feedback est démarré ou relancé (SIGCONT).
- Quand un processus est terminé, toutes ses ressources sont retournées pour être allouées à d'autres processus.
- Quand il n'y a plus aucun processus dans la liste de soumissions, la file d'entrée et les files de feedback, l'ordonnanceur termine.

La liste des soumissions

C'est la liste des processus à traiter par l'ordonnanceur. Cette liste est initialement contenue dans un fichier texte « liste-taches » qui est spécifié comme argument dans la ligne de commande de l'ordonnanceur comme suit :

```
>LOG710H14_Lab02 liste-taches
```

Chaque ligne de la liste décrit un processus avec les données suivantes délimitées par des «

virgules-espaces » :

```
<temps d'arrivée>, <priorité>, <temps d'exécution>, <nombre
d'imprimantes>,

<nombre de scanneurs>, <nombre de modems>, <nombre de CD>
```

Exemple :

12, 0, 1, 0, 0, 0, 0

12, 1, 2, 1, 0, 0, 1

13, 3, 6, 1, 0, 1, 2

Cette liste indique ce qui suit :

- Premier processus : arrive à l'instant 12, avec une priorité 0 (la plus haute, temps réel), a un temps d'exécution de 1 seconde et n'a besoin d'aucun périphérique d'E/S.
- Deuxième processus : arrive à l'instant 12, avec une priorité 1 (processus utilisateur de plus haute priorité), a besoin de deux secondes d'exécution, utilise une imprimante et un CD.
- Troisième processus : arrive à l'instant 13, avec une priorité 3 (processus utilisateur de plus faible priorité), a besoin de 6 secondes d'exécution, utilise une imprimante, un modem et deux CDs.

Évaluation et procédure de remise du laboratoire

Ce laboratoire doit se faire en individuel et il vaut 12 points

Soumettre par émail un rapport du laboratoire (2 points) et les fichiers des programmes.

Le rapport de laboratoire doit présenter essentiellement l'analyse des différentes décisions de conception pour la réalisation de ce laboratoire, les algorithmes et les structures de données utilisées dans vos programmes, les détails des appels systèmes utilisés pour chacun des programmes, une discussion des limites de l'implémentation que vous avez réalisée.

Soumission :

Adresse de soumission

Sujet du email : LOG710 LABO2 …

Annexe du lab#2 : Code de programme en C--> log710h14process.c

```
/*

  usage:


    log710h14process [n]
```

```
    [n] is time for process to exist - default 20 seconds

  program ticks away reporting process id and tick count every
  second. the program traps and reports the following signals:

    SIGINT, SIGQUIT, SIGHUP, SIGTERM, SIGABRT, SIGCONT, SIGTSTP

  program can not trap SIGSTOP or SIGKILL
  */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/times.h>
#include <limits.h>
#include <sys/resource.h>
#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif
static void SignalHandler(int);
void        PrintUsage(char*);   // for error exit & info
```

```
char       *StripPath(char*);    // strip path from filename

#define DEFAULT_TIME 20

/*#define DEFAULT_OP   stdout*/

#define DEFAULT_NAME «log710h14process»

#define BLACK   «\033[30m»       // foreground colours

#define RED     «\033[31m»

#define GREEN   «\033[32m»

#define YELLOW  «\033[33m»

#define BLUE    «\033[34m»

#define MAGENTA «\033[35m»

#define CYAN    «\033[36m»

#define WHITE   «\033[37m»

#define ON_BLACK   «\033[40m»    // background colours

#define ON_RED     «\033[41m»

#define ON_GREEN   «\033[42m»

#define ON_YELLOW  «\033[43m»

#define ON_BLUE    «\033[44m»

#define ON_MAGENTA «\033[45m»

#define ON_CYAN    «\033[46m»

#define ON_WHITE   «\033[47m»

#define NORMAL      «\033[0m»     // not bold/underline/flashing/...

char * colours [] = { BLACK ON_WHITE, CYAN ON_RED, GREEN
ON_MAGENTA,

                      BLUE ON_YELLOW, BLACK ON_CYAN, WHITE ON_RED,

                      BLUE ON_GREEN, YELLOW ON_MAGENTA, BLACK
ON_GREEN,

                      YELLOW ON_RED, BLUE ON_CYAN, MAGENTA
ON_WHITE,

                      BLACK ON_YELLOW, GREEN ON_RED, BLUE
ON_WHITE,

                      CYAN ON_MAGENTA,

                      WHITE ON_BLACK, RED ON_CYAN, MAGENTA
```

```
ON_GREEN,
                        YELLOW ON_BLUE, CYAN ON_BLACK, RED ON_WHITE,
                        GREEN ON_BLUE, MAGENTA ON_YELLOW, GREEN
ON_BLACK,
                        RED ON_YELLOW, CYAN ON_BLUE, WHITE
ON_MAGENTA,
                        YELLOW ON_BLACK, RED ON_GREEN, WHITE
ON_BLUE,
                        MAGENTA ON_CYAN };
#define N_COLOUR 32
char * colour;                          // choice of colour for
this process
/*FILE * output = DEFAULT_OP;*/
FILE * output;
static int signal_SIGINT = FALSE;      // flags set by signal
handler
static int signal_SIGQUIT = FALSE;     // (all response done in
main process
static int signal_SIGHUP = FALSE;      //  rather than in
interrupt routine)
static int signal_SIGTERM = FALSE;
static int signal_SIGABRT = FALSE;
static int signal_SIGCONT = FALSE;
static int signal_SIGTSTP = FALSE;
/*****************************************************************/
int main(int argc, char *argv[])
{
    pid_t pid = getpid();              // get process id
    int i, cycle, rc;
    long clktck = sysconf(_SC_CLK_TCK);
    struct tms t;
    clock_t starttick, stoptick;
    sigset_t mask;
```

```
   output = stdout;
    colour = colours[pid % N_COLOUR]; // select colour for this
process
    if (argc > 2 || (argc == 2 && !isdigit((int)argv[1][0])))
        PrintUsage(argv[0]);
    fprintf(output,»%s%7d; START» BLACK NORMAL «\n», colour,
(int) pid);
    fflush(output);
    signal (SIGINT, SignalHandler);   // hook up signal handler
    signal (SIGQUIT, SignalHandler);
    signal (SIGHUP, SignalHandler);
    signal (SIGTERM, SignalHandler);
    signal (SIGABRT, SignalHandler);
//  signal (SIGCONT, SignalHandler);  // do this intrinsically
after return from SIGTSTP
                                      // due to Darwin/BSD
inconsistent SIGCONT behaviour
    signal (SIGTSTP, SignalHandler);
    rc = setpriority(PRIO_PROCESS, 0, 20); // be nice, lower
priority by 20
    cycle = argc < 2 ? DEFAULT_TIME : atoi(argv[1]);  // get tick
count
    if (cycle <= 0) cycle = 1;

    for (i = 0; i < cycle;) {          // tick

        if (signal_SIGCONT) {
            signal_SIGCONT = FALSE;
            fprintf(output,»%s%7d; SIGCONT» BLACK NORMAL «\n»,
colour, (int) pid);
            fflush(output);
        }
```

```
    starttick = times (&t);        // use timer to ascertain
whether 'tick' should be
        rc = sleep(1);                  //  reported
        stoptick = times (&t);
        if (rc == 0 || (stoptick-starttick) > clktck/2)
            fprintf(output,»%s%7d; tick %d» BLACK NORMAL «\n»,
colour, (int) pid, ++i);
        if (signal_SIGINT) {
            fprintf(output,»%s%7d; SIGINT» BLACK NORMAL «\n»,
colour, (int) pid);
            exit(0);
        }
        if (signal_SIGQUIT) {
            fprintf(output,»%s%7d; SIGQUIT» BLACK NORMAL «\n»,
colour, (int) pid);
            exit(0);
        }
        if (signal_SIGHUP) {
            fprintf(output,»%s%7d; SIGHUP» BLACK NORMAL «\n»,
colour, (int) pid);
            exit(0);
        }
        if (signal_SIGTSTP) {
            signal_SIGTSTP = FALSE;
            fprintf(output,»%s%7d; SIGTSTP» BLACK NORMAL «\n»,
colour, (int) pid);
            fflush(output);
            sigemptyset (&mask);            // unblock SIGSTP if
necessary (BSD/OS X)
            sigaddset (&mask, SIGTSTP);
            sigprocmask (SIG_UNBLOCK, &mask, NULL);
            signal(SIGTSTP, SIG_DFL);       // reset trap to
default
```

```
            raise (SIGTSTP);                // now suspend
ourselves
            signal(SIGTSTP, SignalHandler); // reset trap on
return from suspension
            signal_SIGCONT = TRUE;          // set flag here
rather than trap signal
        }
        if (signal_SIGABRT) {
            fprintf(output,»%s%7d; SIGABRT» BLACK NORMAL «\n»,
colour, (int) pid);
            fflush(output);
            signal (SIGABRT, SIG_DFL);
            raise (SIGABRT);
        }
        if (signal_SIGTERM) {
            fprintf(output,»%s%7d; SIGTERM» BLACK NORMAL «\n»,
colour, (int) pid);
            exit(0);
        }
        fflush(output);
    }
    exit(0);
}
/*****************************************************************
**
  static void SignalHandler(int sig)
  trap and report the following signals:
    SIGINT, SIGQUIT, SIGHUP, SIGTERM, SIGABRT, SIGCONT, SIGTSTP
  program can not trap SIGSTOP or SIGKILL .
  Note minimal time in signal handler
 *****************************************************************
***/
static void SignalHandler(int sig)        // trap signals from
```

```
shell/system

{

    switch (sig) {

        case SIGINT:

            signal_SIGINT = TRUE;

            break;

        case SIGQUIT:

            signal_SIGQUIT = TRUE;

            break;

        case SIGHUP:

            signal_SIGHUP = TRUE;

            break;

        case SIGCONT:

            signal_SIGCONT = TRUE;

            break;

    case SIGTSTP:

            signal_SIGTSTP = TRUE;

            break;

        case SIGABRT:

            signal_SIGABRT = TRUE;

            break;

        case SIGTERM:

            signal_SIGTERM = TRUE;

            break;

    }

}

/*****************************************************************
***

  void PrintUsage(char * pgmName)


  print program usage
```

```
   pgmName - program name

             if NULL defaults to DEFAULT_NAME

  ****************************************************************
***/

 void PrintUsage(char * pgmName)

 {

    char * actualName;

     if (!(actualName = StripPath(pgmName))) actualName =
 DEFAULT_NAME;

     printf(«\n»

            «  program: %s - trap and report process control
 signals\n\n»

            «    usage:\n\n»

            «      %s [seconds]\n\n»

            «      where [seconds] is the lifetime of the program
- default = 20s.\n\n»

            «    the program sleeps for a second, reports process
 id and tick count\n»

            «    before sleeping again. any process control
 signals: SIGINT, SIGQUIT\n»

            «    SIGHUP, SIGTERM, SIGABRT, SIGCONT, SIGTSTP, are
 trapped and\n»

            «    reported before being actioned.\n\n»,

            actualName, actualName );

     exit(127);

 }

 /****************************************************************
***

 char * StripPath(char * pathname);

 strip path from file name

 pathname - file name, with or without leading path

 returns pointer to file name part of pathname

 if NULL or pathname is a directory ending in a '/'
```

```
returns NULL

**********************************************************************
**/

char * StripPath(char * pathname)

{

    char * filename = pathname;

    if (filename && *filename) {           // non-zero length
string

        filename = strrchr(filename, '/'); // look for last '/'

        if (filename)                       // found it

            if (*(++filename))              //  AND file name exists

                return filename;

            else

                return NULL;

        else

            return pathname;                // no '/' but non-zero
length string

    }                                       // original must be
file name only

    return NULL;

}
```

Laboratoire #3 : Conception et implémentation d'un gestionnaire de mémoire avec plusieurs stratégies d'allocation

Objectif du laboratoire

L'objectif de ce laboratoire est de concevoir et implémenter un gestionnaire de mémoire qui utilise plusieurs stratégies d'allocation.

Nombre de semaines pour la réalisation de ce laboratoire : 02

Spécification du laboratoire

Dans ce laboratoire vous allez implémenter un gestionnaire d'un bloc de mémoire. Pour cela vous devez implémenter les fonctions d'allocation et de libération de mémoire et garder la trace de la mémoire utilisée. Vous devez implémenter quatre (04) stratégies pour sélectionner un bloc libre pour satisfaire une requête d'allocation de mémoire.

Stratégies d'allocation

Les stratégies d'allocation sont celles étudiées en cours : (1) First-fit, (2) Best-fit, (3) Worst-fit et (4) next-fit. Cette dernière signifie la sélection du premier bloc libre convenable après le dernier bloc alloué avec un effet circulaire. Convenable, dans ce contexte, signifie un bloc suffisamment grand pour satisfaire la requête.

Fonctions à implémenter

L'interface du gestionnaire de mémoire est composée des fonctions suivantes que vous devez implémenter dans ce laboratoire.

```
initmem() :

    Initialise les structures de données du gestionnaire.

pBloc = alloumem() :

    Cette fonction alloue un nouveau bloc de mémoire.

liberemem(pBloc) :

    Cette fonction libère un bloc de mémoire similaire à free().

nBlocs = nbloclibres() :

    Cette fonction retourne le nombre de blocs de mémoire libres

nBlocAll = nblocalloues() :

    Cette fonction retourne le nombre de blocs alloués
présentement

mLib = memlibre() :

    Combien de mémoire libre (non allouée) au total.

tGraBloc = mem_pgrand_libre() :

    Taille du plus grand bloc libre.

mem_small_free(maxTaillePetit) :

    Combien de petits blocs (plus petit que maxTaillePetit) non
alloués y a-t-il présentement en mémoire?

mem_est_alloue(pOctet) :

    Est-ce qu'un octet (byte) particulier est alloué
```

Structure de données

Vous devez concevoir une structure de données qui vous permet d'implémenter les fonctions du gestionnaire de mémoire spécifiées plus haut. Ça doit être une structure extensible et efficace permettant de parcourir les blocs occupés et libres. Chaque requête d'allocation ou de libération de mémoire doit créer un nouveau bloc ou combiner des blocs existants.

Quelle que soit votre implémentation du gestionnaire de mémoire vous devez garantir qu'il n'y a jamais de blocs libres adjacents. De tels blocs doivent toujours être combinés en un seul bloc plus grand.

Vous devez réaliser ce laboratoire d'une façon incrémentale. Dans une première phase, vous devez implémenter et tester toutes les fonctions en utilisant seulement la stratégie first-fit. Dans une deuxième phase, vous généralisez votre implémentation en utilisant la stratégie d'allocation comme un paramètre.

Évaluation et procédure de remise du laboratoire

- Ce laboratoire doit se faire en individuel et il vaut 8 points répartis comme suit : Phase 1 (3 points), Phase 2 (3 points) et le rapport du laboratoire (2 points).
- Soumettre par émail le rapport du laboratoire et les fichiers des programmes.
- Le rapport de laboratoire doit présenter essentiellement l'analyse des différentes décisions de conception pour la réalisation de ce laboratoire, les algorithmes et les structures de données utilisées dans vos programmes, une discussion des limites de l'implémentation que vous avez réalisée.
- Soumission :
- Sujet du courriel : LOG710 LABO3

Siège de l'Université Virtuelle Africaine

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Bureau Régional de l'Université Virtuelle Africaine à Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org

2017 UVA