Fiche de synthèse du cours
Système et programmation système
Licence 2 Informatique

Julien BERNARD

2012 – 2013

Attention ! Cette synthèse ne remplace pas les notes de cours, elle sert juste à indiquer ce qui est important dans ce cours. Il est donc primordial de prendre des notes pendant le cours de manière à avoir l'ensemble de l'information.

1 Introduction

1.1 Utilisation du système

1.1.1 Interpréteur de commande

- Prompt (`$`), saisie, analyse et exécution

1.1.2 Pages de manuel

- `man(1)`
- Sections :
  1. commandes
  2. appels système
  3. bibliothèques
- `--help`

2 Système de fichiers

2.1 Système de fichiers

2.1.1 Fichiers

- Tout est fichier : fichiers réguliers et virtuels, répertoires, périphériques, tubes, sockets.
- Nom de fichier :
  - nom + extension(s)
  - max 255 caractères
  - fichiers cachés commencent par '`.`'

2.1.2 Répertoires

- Système de fichiers = ensemble de fichiers et répertoires organisé en arbre avec racine `/`
- *Filesystem Hierarchy Standard*
- `.` : répertoire courant, `..` : répertoire parent
- Chemin relatif et chemin absolu

2.1.3 Information et navigation

- `pwd(1)`, `ls(1)`
- `cd` (*Change Directory*)

2.2 Utilisateurs et permissions

- Utilisateur = nom + UID (`/etc/passwd`)
- Groupe = nom + GID (`/etc/group`)
- `root` est le super-utilisateur : UID 0, GID 0
- lecture `r`, écriture `w`, exécution `x`
- utilisateur `u`, groupe `g`, autres `o`
- Permissions : 3 triplets `rwxr-xr-x`
- Permissions spéciales : SUID, SGID, Sticky Bit
- Représentation octale : `r` = 4, `w` = 2, `x` = 1
- Permissions : 3 (ou 4) chiffres octal 644
- SUID = 4000, SGID = 2000, Sticky Bit = 1000
- `id(1)`, `chown(1)`, `chgrp(1)`
- `umask` : droits initiaux
- `chmod(1)`
- 10è caractère dans `ls -l` : type du fichier

2.2.1 Inodes et structure sur le disque

- inode = méta-informations + *i-number*
- `ls -i`, `stat(1)`
- id du périphérique, numéro inode, permissions, nombre de liens, utilisateur, groupe, taille, *atime*, *mtime*, *ctime*

- Répertoire : ensemble de numéro inode + nom
- Fichier : adresse des blocs sur le disques et simple/double/triple indirection

2.3 Gestion de fichiers

2.3.1 Opérations de base

- `cp(1)`, `mv(1)`, `rm(1)`
- `mkdir(1)`, `rmdir(1)`

2.3.2 Liens

- Lien dur : 2 noms → 1 inode
- Lien symbolique : 1 nom → 1 nom
- `ln(1)`, `unlink(1)`

3 Shell et scripts shell

3.1 Commandes

3.1.1 Flux standard

- Flux standard :
  - Entrée, `stdin`, 0
  - Sortie, `stdout`, 1
  - Erreur, `stderr`, 2
- Tube : `commande1 | commande2`
- Redirections :
  - Entrée : `commande < fichier`
  - Sortie : `commande > fichier`
  - Erreur : `commande 2> fichier`

3.1.2 Valeur de retour

- Valeur de retour du programme : 0 ⇔ ok
- Enchaînement : `commande1 && commande2`
- Erreurs : `commande1 || commande2`

3.1.3 Variables

- Affectation : `VAR=value`
- Contenu : `${VAR}`
- Environnement : ensemble des variables d'environnement
  `env(1)`, `$PATH`, `export VAR`
- Lecture : `$IFS`, `read`

3.1.4 Développement de la ligne de commande

- Développement du tilde
- Développement des variables
- Substitution de commandes :
  `$(commande)` ou `‘commande‘`
- Développement arithmétique :
  `$((expression))`
- Découpage en mots
- Développement des chemins :
  - `*` : n'importe quel chaîne
  - `?` : n'importe quel caractère
  - `[abc]` : a, b ou c
  - `[a-z]` : a, . . ., z
  - `[[:classe:]]`

3.2 Programmation shell

3.2.1 Principes

- Script shell ou fichier de commandes
  Shebang : `#!/bin/sh`
- `type` : commande interne / externe
- Paramètres : `$0` (nom du script/shell), `$1`, . . .
- Variables spéciales :
  - `$#` : nombre de paramètres
  - `$*` : liste des paramètres
  - `$?` : valeur de retour
  - `$RANDOM` : nombre aléatoire

3.2.2 Tests et structures de contrôle

- `test(1)`
- `if then else fi`
- `for in do done`
- `while do done`
- `until do done`

3.2.3 Fonctions

- Ensemble de commandes :
  `{ commande1; commande2; }`
  `( commande1; commande2; )`
- Fonction : `nom () commande`

4 Programmation en C

4.1 Le langage C

4.1.1 Généralités

- Langage impératif, procédural, bas niveau avec gestion explicite de la mémoire

4.1.2 Types et opérateurs

- Types de base : comme en Java
- Opérateurs : comme en Java
- Pas de booléen, x est vrai $\Leftrightarrow x \neq 0$
- Types structurés : `struct`, `enum`
- Variables constantes : `const`
- Alias de type : `typedef`

4.1.3 Structures de contrôle

- `if/if-else` : comme en Java
- `while/do-while` : comme en Java
- `for` : comme en Java
- `switch` : comme en Java

4.1.4 Tableaux et pointeurs

- Tableau : éléments contigus du même type
- Tableau statique : `type nom[N];`
- Pointeur : adresse vers mémoire typée
  `type *`, `NULL`
- Opérateur référencement : `&`
- Opérateur déréférencement : `*`
- Accès aux membres d'une structure : `->`
- Arithmétique sur les pointeurs :
  - Addition : `p + n` → `q`
  - Soustraction : `q - p` → `n`
- Tableau = Pointeur sur le premier élément

4.1.5 Chaînes de caractères

- Chaîne : tableau de `char` terminée par `'\0'`

4.1.6 Fonctions

- Définition : nom, arguments, corps
- Déclaration : nom, arguments (= prototype)
- Arguments par valeur
- Fonction principale : `int main(void);`
  ou `int main(int argc, char argv[]);`

4.2 Bibliothèque standard

4.2.1 Généralités

- Bibliothèque standard : ensemble de fonctions de base dans 24 fichiers d'en-têtes

4.2.2 Entrée/Sortie simple

- Sortie : `printf(3)`
- Entrée : `scanf(3)`
- Chaîne de format : `%d`, `%f`, `%s`, ...

4.2.3 Allocation mémoire

- Type de mémoire :
  - Statique : segment, compilation, implicite
  - Automatique : pile, exécution, implicite
  - Dynamique : tas, exécution, explicite
- Allocation : `malloc(3)`
- Libération : `free(3)`
- Allocation tableau : `calloc(3)`

4.2.4 Manipulation de chaîne de caractères

- Longueur : `strlen(3)`
- Copie : `strcpy(3)`
- Conversion depuis entier : `atoi(3)`

4.2.5 Fonctions mathématiques

- `sin(3)`, `cos(3)`, `tan(3)`, `log(3)`, `pow(3)` ...
- `libm` → `-lm` pour compiler

4.3 Production de programme

4.3.1 Production simple

- Compilation : `gcc -c -o foo.o foo.c`
- Édition de liens : `gcc -o foo foo.o`

5 Développement en C

5.1 Bonnes pratiques de développement

5.1.1 Préprocesseur

- Préprocesseur : transforme le code source via des directives qui commencent par `#`
- `#include` : inclure un fichier
  - `#include <fichier.h>` : fichier système
  - `#include "fichier.h"` : fichier local
- `#define` : définir une constante
- `#if`, `#ifdef`, `#ifndef`, `#else`, `#endif` : compilation conditionnelle
- `#define` : définir une macro
  - Parenthèses autour des paramètres !
  - Parenthèses autour de la macro !

  Macro ≠ Fonction
- Option `-DFOO` = `#define FOO`

5.1.2 Unité de compilation

- Unité de compilation : `.c` + `.h`
- En-tête (*header*) : types et prototypes
- *Include guard* :
  ```
  #ifndef FOO_H
  #define FOO_H
  ...
  #endif
  ```

5.1.3 Production avancée

- Étapes de production :
  - Préprocesseur : `cpp(1)`
  - Compilateur : `cc(1)`
  - Assembleur : `as(1)`
  - Éditeur de lien : `ld(1)`
- `gcc` : pilote de compilation
- Objet = code machine + symboles indéfinis
- Liaison : résolution des symboles
  - Statique : dans d'autres fichiers objets
  - Dynamique : dans une bibliothèque externe
- Exécutable avec plusieurs fichiers source :
  - Compilation en fichiers objets
  - Édition de liens
- Programme : exécutable avec `main`
- Bibliothèque : ensemble de fonctions sans `main`
  - Statique : `libqux.a`, `ar(1)`
  - Dynamique : `libqux.so`, `gcc -shared`
- Utilisation d'une bibliothèque : `-lqux`
- Chargement d'un programme : `ld.so(8)`

5.2 Make

5.2.1 Principe d'un Makefile

- `Makefile` : production automatique
- `cible: dépendances`
  `⟼actions`
- Cibles : `all`, `clean`

5.2.2 Makefile avancé

- Variables : idem shell
- `${CC}`, `${CFLAGS}`, `${LDFLAGS}`
- `-Wall` : active tous les warnings (obligatoire)
- Variables spéciales :
  - `$@` : nom de la cible
  - `$<` : première dépendance
  - `$^` : liste de toutes les dépendances
- Règle d'inférence : `%` est un nom générique

5.2.3 Générateur de Makefile

- Script `configure`
- ```
  $ ./configure
  $ make
  $ make install
  ```
- Autotools : `automake(1)`, `autoconf(1)`
- CMake

6 Manipulation de fichiers

6.1 Fichiers en C

6.1.1 Généralités

- Deux API C :
  - Descripteur de fichier
  - Descripteur de flux

6.1.2 Descripteur de fichier

- Descripteur de fichier = entier, index dans la table des descripteurs
- Descripteurs spéciaux :
  - 0 : entrée standard
  - 1 : sortie standard
  - 2 : erreur standard
- Opérations :
  - Ouverture : `open(2)`
  - Création : `creat(2)`
  - Fermeture : `close(2)`
  - Lecture : `read(2)`
  - Écriture : `write(2)`

6.1.3 Flux `FILE*` et `DIR*`

- Descripteur de flux = pointeur sur structure opaque `FILE`
- Descripteurs spéciaux :
  - `stdin` : entrée standard
  - `stdout` : sortie standard
  - `stderr` : erreur standard
- Opérations :
  - Ouverture : `fopen(3)`
  - Fermeture : `fclose(3)`
  - Lecture : `fread(3)`
  - Écriture : `fwrite(3)`
  - Sortie formattée : `fprintf(3)`
- Descripteur de répertoire = pointeur sur structure opaque `DIR`
- Opérations :
  ```

  - Ouverture : `opendir(3)`
  - Fermeture : `closedir(3)`
  - Lecture : `readdir(3)`
- Relations avec les descripteurs de fichiers : `fileno(3)`, `dirfd(3)`

6.2 Manipulation de fichiers texte

6.2.1 Généralités

- Commande = filtre

6.2.2 Sélection

- Recherche de motif : `grep(1)`
- Sélection des champs sur une ligne : `cut(1)`

6.2.3 Modification

- Remplacement de motif : `sed(1)`
- Tri : `sort(1)`
- Élimination des lignes répétées : `uniq(1)`

6.2.4 Affichage

- Affichage page par page : `more(1)` or `less(1)`
- Début d'un fichier : `head(1)`
- Fin d'un fichier : `tail(1)`

6.2.5 Informations

- Nombre de lignes/mots/octets : `wc(1)`

7 Processus

7.1 Processus

7.1.1 Création et exécution d'un processus

- Processus = instructions + espace mémoire + ressources
- PID : Process IDentifier
- Processeur : ordonnance les processus
- Ordonnancement :
  - Multi-tâche coopératif
  - Multi-tâche préemptif
    - Temps partagé
    - Temps réel
- Processus père (PPID) et processus fils
- `init(8)`, racine de l'arbre des processus
- États d'un processus :
  - Créé
  - En attente
  - En exécution
  - Bloqué
  - Terminé
- Fin d'un processus : `exit(2)`
- Processus zombie
- Processus orphelin
- `/proc`
- SUID et SGID

7.1.2 Signaux

- Signal : communication inter-processus simple
- `SIGINT`, `SIGQUIT`, `SIGKILL`, `SIGTERM`
- `SIGCHLD`
- `SIGSTOP`, `SIGTSTP`
- `SIGBUS`, `SIGFPE`, `SIGSEGV`
- `CTRL+C` : `SIGINT` ; `CTRL+Z` : `SIGTSTP`

7.1.3 Contrôle des processus

- Processus en cours : `ps(1)`
  - `ps aux`
  - `ps -ef`
- Affichage dynamique des processus : `top(1)`
- Envoie d'un signal : `kill(1)`, `killall(1)`
- PID d'une commande : `pidof(1)`

7.1.4 Processus et shell

- Lancement d'un processus en arrière plan : `&`
- Suspension d'un processus : `CTRL+Z`
- Reprise du dernier processus suspendu...
  - ... en arrière plan : `bg(1)`
  - ... en avant plan : `fg(1)`

8 Programmation système

8.1 Programmation système

8.1.1 Duplication et recouvrement de processus

- Duplication d'un processus : `fork(2)`
- Recouvrement d'un processus : `execve(2)`
- Famille `exec(3)`
- Attente d'un processus : `wait(2)`, `waitpid(2)`
- Obtention d'un PID : `getpid(2)`, `getppid(2)`
- Terminer un processus normalement : `exit(3)`

8.1.2 Signaux

- Envoyer un signal : `kill(2)`
- Attendre un signal : `pause(2)`
- Programmer un réveil : `alarm(2)`
- Gestionnaire de signal : `signal(2)`
- Gestionnaire avancé de signal : `sigaction(2)`

8.1.3 Tubes et redirections

- Créer un tube : `pipe(2)`
- Dupliquer un descripteur :`dup(2)`, `dup2(2)`

8.2 Daemons

8.2.1 Définition d'un daemon

- Daemon, Disk And Execution MONitor
- Fils de `init(8)`, détaché d'un terminal

8.2.2 Session et groupe de processus

- Hiérarchie à deux niveaux :
  - Processus ∈ Groupe de processus
  - Groupe de processus ∈ Session
  - Un terminal = Une session
- Groupe de processus :
  - PGID, `getpgrp(2)`
  - Une commande ou un pipeline
  - Leader de groupe
- Session :
  - SID, `getsid(2)`
  - Leader de session (shell)
  - Création d'une session : `setsid(2)`
  - Terminaison du leader : `SIGHUP`
- Daemon : «Fork-Off And Die»