Institut Paul Lambin

BAC 2 Informatique de gestion

Unix

Synthèse Unix

Auteurs :
Christopher Sacré

Professeur :
C. De Muylder
B. Henriet
A. Ninane

25 mai 2017

Table des matières

1 Introduction

Interfaces du noyau, ils sont écrit et utilisés en C (Ils peuvent également être utilisé dans d'autres langages). Il y en a un nombre finis dans le noyau Unix. (Le point d'entrée du noyau est unique, chaque appel système est un des branchements de ce switch). Il en exite différents types :

- IO
- File management
- Protection
- Processes
- Misc

Problèmes de portabilité : Les programmes utilisant directement les appels systèmes ne sont pas des standard du langage C, sont des programmes C "Unix", ne sont pas portable entre les différents types de Unix. (Problème avec les appels systèmes : les fichiers en-têtes, ainsi que le type des arguments). Afin d'éviter ce genre de soucis, utiliser une directive de compilation (CPP).

Compilation : Il faut savoir que cc passe par plusieurs étapes de compilation (Pré processing (va s'occuper de traiter les define, les include, ...). Quant à gcc il faut faire attention car il s'agit d'un compilateur non optimisé.

Types de données des primitives sytèmes : (permet de cacher les détails d'implémentation, définit au sein de ¡sys/types.h¿)

Traitement des erreurs : tous les appels systèmes retournent -1 en cas d'erreur. De ce fait le code de retour d'un appel système doit toujours être vérifié. (On peut également utiliser errno (¡errno.h¿) ainsi que perror.

Opérations I/O : Unix fournit une interface uniforme sur des ressources telles que :

- Fichiers
- Terminaux
- Pipes
- "Tapes"
- "Network Sockets"

Les ressources sont identifiées par un entier appelé "file descriptor" (il s'agit d'une abstraction du programme afin d'accéder à une ressource). Il faut également savoir que les appels systèmes ne peuvent pas être bufferisées (Chaque appel système implique une opération au niveau du noyau ainsi qu'au niveau de l'appareil).

2 File Descriptor

Les fichiers ouverts sont désignés par un "file descriptor". Ils sont obtenus à l'aide des opérations open(), creat(), socket(). Pour chaque processus le noyau maintient une table des fichiers ouverts. Il existe trois "file descriptor" réservés au système :

- *0* : Entrée Standard
- *1* : Sortie Standard
- *2* : Sortie d'erreur

Les fichiers : Les fichiers sont vus comme des séquences de bytes (pas d'enregistrement, les blocs physiques ne sont pas visibles). L'offset du fichier courant "tight" au "file descriptor". Les blocs d'appareils sont vus comme des séquences de blocs (l'entiereté des blocs sont lus/écrits en une fois).

3 Standard I/O Library

Interface uniforme permettant d'effectuer des opérations d'I/O (Il s'agit d'une interface efficace pour une programmation de haut niveau, fonctionnant avec les streams et permettant un système de buffer de haut niveau (Attention, les Appels Systèmes coûte très cher)) de manière très simple (un seul fichier à inclure (#include ¡stdio.h¿)).

Les Streams : Similaire au "file descriptor", ils désignent les appareils (Ils existent bien entendus des streams prédéfinis de base (stdin, stdout, stderr), et ils sont alloués dynamiquement(un appel à malloc n'est pas requis, la fonction fopen s'en chargeant pour nous)).

User-level buffering : la sortie n'est pas synchronisée (afin de pallier à cela on pourrait utiliser le système de "flush" (fflush, setbuf)). De plus les streams peuvant être détruit à l'aide de fclose().

Fonctions d'entrée

- int fgetc(FILE *stream) ;
- char *fgets(char *s, int size, FILE *stream) ;
- int getc(FILE *stream) ;
- int getchar(void) ;
- char *gets(char *s) ;
- int ungetc(int c, FILE *stream) ;
- scanf(char * format, ptr1, ptr2) ;
- fscanf(FILE * ioptr, char * format, ptr1, ptr2) ;
- sscanf(char * inbuf, char * format, &arg1, &arg2) ;
- strtok

Attention à un eventuel buffer overflow (il faut toujours vérifier la taille de ce que l'on reçoit, gets est d'ailleurs à éviter).

Fonctions de sortie

- int fputc(int c, FILE *stream) ;
- int fputs(const char *s, FILE *stream) ;
- int putc(int c, FILE *stream) ;
- int putchar(int c) ;
- int puts(const char *s) ;
- printf(char * format, arg1, arg2) ;
- fprintf(FILE * ioptr, char * format, arg1, arg2) ;
- sprintf(char * outbuf, char * format, arg1, arg2) ;

Les bonnes pratiques de programmation : il vaut mieux utiliser fgets afin de lire les lignes entrées. Il faut vérifier si un input est vide. Il vaut mieux utiliser sscanf afin de décoder les lignes entrées. Il faut utiliser strtok afin de recevoir un token à partir d'un input.

4 Redirection

Limite des Ressources Les ressources peuvent être limitées au niveau du shell (ulimit -aS pour une "soft limit" et ulimit -aH pour une "hard limit").

User kernel ops "mapping" : pour chaque commande/operation au niveau de l'utilisateur il y en a un équivalent au niveau du noyau.

Implémenter une redirection : fermer l'entrée/sortie standard ouvrir un fichier d'entrée/sortie (open() renvoie le plus petit "file descriptor" disponible). Il fqut faire attention car certains problèmes peuvent survenir (l'appareil originel pointé par fd=0 est fermé, par la suite fd=0 est remplacé par "in", le fichier originel perdus est le clavier), la solution à cela est d'utiliser int dup(int fd) qui permet une dupplication du "file descriptor".

Mécanismes d'entrées/sorties :

- *Mécanismes d'entrées/sorties basiques UNIX :* les données sont lues/écrites depuis les fichiers par un processus. Les i/o sont vus commes des séquences de bytes non-structurées.
- *Mécanismes d'entrées/sorties mappés en mémoire :* les objets externes sont mappés dans la mémoire virtuelle du processus.

5 Locks

creat() et unlink() peuvent être utilisés afin de créer des cadenas (Locks) (while (creat("lockfile", 0) ¡ 0) sleep() ; unlink("lockfile")). Il faut faire attention car cela peut générer une erreur (le orphan lockfile). Il faut savoir que de nombreux problèmes apparaissent lorsque de multiples processus accèdent à la même donnée simultanément. (Pour cela on peut protéger les données à l'aide

de cadenas, ces cadenas peuvent être implémentés de bien des manières mais ils doivent être atomiques et sont conseillé et non obligatoire).

6 Processus

Un processus est un programme en cours d'exécution. (Un programme est donc un objet statique (un exécutable dans le système de fichier), le processus quant à lui est un objet dynamique (il changera durant l'exécution)).

Définition d'un processus : Un processus est un programme en cours d'exécution. Un processus contient des données en mémoire dont :

- *Du texte :* les instructions qui devront être exécutées.
- *Des données globales :* initialisées ou non.
- *Des données locales :* placées sur la pile.
- *Un Heap :* de la mémoire libre pour les appels à la fonction malloc.
- *Un environnement d'exécution :* tant pour le programme (statut, ...) que pour le système.
- *Des variables d'environements :* Il s'agit de couples de clé-valeur qui peuvent être accédées par le processus tout comme par ses enfants.

Il est identifié par un numéro unique dans le système Unix (PID : process identifier). Il est exécuté par un utilisateur (UID : Identifie l'utilisateur exécutant le processus).

Les variables d'environnement : Un processus garde ses variables d'environnement et les paramètres de l'utilisateur définissent des variables d'environnement. Il s'agit de couple clé-valeur. On peut y accéder à partir d'un programme utilisateur (getenv()).

Lancer un processus : La seule manière de créer un processus est de le faire à l'intérieur d'un processus existant et en utilisant l'appel système fork().

7 Interruptions système

Un signal électrique est envoyé par un appareil au processeur afin de stopper l'exécution du processus courant et de faire quelque chose de différent. Le gestionnaire d'interruption est sélectionné par le processeur par la lecture du vecteur d'interruption récupéré de l'appareil. L'état d'exécution est sauvegardé avant l'exécution du gestionnaire d'interruptions et restauré par la suite.

Les signaux : il s'agit d'interruptions software. Les signaux peuvent être envoyés à un processus par un autre processus, par le terminal de controle, par lui-même ou tout simplement par le système. L'action par défaut d'un signal est de terminer le processus et de produire un fichier "core dump" (dépend du type de signal). Un processus peut décider d'ignorer certains signaux ou d'en

attraper certains. Un signal peut être envoyé à un groupe de processus (Mais un signal ne peut être envoyé que par un processus ayant le même uid). Liste des signaux :

- SIGHUP : Hangup/Reload (Terminate)
- SIGINT : Keyboard interrupt (Terminate)
- SIGILL : Illegal Instruction (Core dumped)
- SIGFPE : Arithmetic exception (Core Dumped)
- SIGKILL : Kill (Terminate)
- SIGBUS : Bus error (Core Dumped)
- SIGSEGV : Segmentation Violation (Core Dumped)
- SIGPIPE : Write a pipe (no reader) (Terminate)
- SIGALRM : Alarm clock (Terminate)
- SIGTERM : Software termination (Terminate)
- SIGSTOP : Stop (Suspended)
- SIGTSTP : Stop (from keyboard) (Suspended)
- SIGCONT : Continue after stop (Discarded)
- SIGCHLD : Child status has changed (Discarded)
- SIGUSR1 : User defined signal 1 (Terminate)
- SIGUSR2 : User defined signal 2 (Terminate)

Les signaux peuvent être générés par le clavier. Les signaux ne sont traités que si le processus est actif, est en attente I/O sur un appareil plutot lent, ou sur un pipe. D'ailleurs si le signal arrive lors d'une attente I/O cette attente d'I/O se termine prématurément.

8 IPC's

Introduction : Il existe différents types de structure IPC (Les files de messages, les sémaphores, les mémoires partagées). Mais ils partagent tous des caractéristiques communes : un identifiant d'IPC ainsi qu'une structure de permission.

Identifiant IPC : L'identifiant IPC permet d'identifier la structure IPC. IL s'agit d'un entier ne pouvant que s'incrémenter. Il est renvoyé par int msgget(key_t key, int msgflg) ; (pour les files de messages), int semget(key_t key, int nsems, int semflg) ; (pour les sémaphores) ainsi que int shmget(key_t key, size_t size, int shmflg) ; (pour les mémoires partagées). Les clés permettent d'accepter de nouveaux processus sur de telles structure. Les identifiant IPC sont globaux dans le système UNIX.

La clé peut prendre comme valeur : IPC_CREAT (si la clé est disponible, crée l'objet IPC) ou IPL_EXCL (Si la clé est déjà utilisée, génère une erreur). Les structures IPC sont des structures largement répandues et peuvent être utilisées par des processus totalement différents. Le seul soucis est qu'il faut se mettre d'accord au sujet de tels fichiers (Une clé commune au sein d'un fichier d'include, LE serveur pourrait passer la valeur e l'ipc à travers un fichier ou pourrait

générer la clé par ftok(const char *pathname, int project_id)).

Structure de permission des structure IPC : Chaque structure IPC possède une structure ipc_perm qui lui est reliée. Voici sa composition : uid_t uid ; /* owner's effective user id */
gid_t gid ;
uid_t cuid ; /* creator's effective used id */
gid_t cgid ;
mode_t mode ; /* access mode */
...
key_t key ;
Comme vous pouvez le remarquer, la composition de structure IPC est très similaire à celle représentant les droits d'accès sur les fichiers régulier. Celle ci peut être changée par : msgctl(), shmctl(), srmctl().

9 Message queue

Il s'agit d'une liste chaînée de message sauvegardée dans l'espace noyau.La pile est identifiée par a "message queue id". Elle est ouverte ou créée par msgget(). Les messages sont envoyés par msgsend() (Ils sont rajouté à la fin de la pile). Les messages sont lus par msgrcv() (Ils ne sont pas nécessairement lu en FIFO). Ces piles possède une structure msqid_ds. Ils permettent une communication bidirectionelle entre des processus totalement différents. Les messages ont un type et les frontières entre ces derniers sont préservés. Les piles de message sont en dehors du système de fichier et en dehors de l'arborescence de processus).

10 Semaphores

Leur but est de permettre la synchronisation des opérations. Il s'agit d'un entier strictement positif sur lequel on peut effectuer des opérations basiques (incrémenter ou décrémenter la valeur de la sémaphore ainsi que bloquer le processus tant que la sémaphore n'atteigne pas une certaine valeur (par exemple : 0)). Elles permettent de contrôler et de protéger l'accès à certaines ressources critiques.

Remarques : La valeur d'initialisation peut être 'n' ¿ 1 (Tant que 'n' processus peuvent accéder à cette ressource simultanément. Il faut également noter que les sémaphores doivent obligatoirement être atomiques. En Unix, on ne possède pas une sémaphore mais un ensemble de sémaphores, plusieurs opérations sont disponibles et sont atomiques (semget() : permet de récupérer un identifiant pour un ensemble de sémaphores, semctl() : permet d'effectuer des opérations de controle sur un ensemble de sémaphores, semop() : permet d'effectuer des opérations sur un ensemble de sémaphores).

11 Mémoires Partagées

Il s'agit d'une manière d'échanger des données entre les processus, elle est plus rapide que les manières conventionnelles car elle ne nécessite pas d'appels systèmes. Par contre elle nécessite d'être créée par une processus, les autre processus devront donc s'y attacher et dès qu'un processus cesse l'utiliser il devra se détacher.

12 Appels Systèmes

12.1 Open()

Include :

- #include ¡sys/types.h¿
- #include ¡stat.h¿
- #include ¡fcntl.h¿

Déclarations :

- int open(const char path, int flags, mode_t mode) ;
- int open(const char path, int flags) ;

Remarques : Ouvre les fichiers en écriture ou en lecture. Retourne soit le "file descriptor" soit -1 (en cas d'erreur). Le chemin, "Path", est une chaîne de cractères désignant la ressource dans le système de fichiers. Pour l'argument flags celui-ci peut prendre les valeurs de :

- O_RDONLY : ouvrir en lecture seulement.
- O_WRONLY : ouvrir en écriture seulement.
- O_RDWR : ouvrir en lecture et en écriture.
- O_CREAT : crée le fichier si il n'existe pas.
- O_APPEND : Place le pointeur sur EOF (permet d'écrire à la suite du contenu du fichier).
- O_TRUNC : Si le fichier existe, le remplace.
- O_EXCL : Crée le fichier d'erreur si il y en a.

Le mode est un argument utilisé avec O_CREAT (Il s'agit du mode de fichier Unix). Les erreur quant à elles peuvent être spécifiées par errno.

12.2 Creat()

Include :

- #include ¡sys/types.h¿
- #include ¡stat.h¿
- #include ¡fcntl.h¿

Déclarations :

- int creat(const char path, mode_t mode) ; Pour les fichiers
- int mkdir(const char path, mode_t mode) ; Pour les dossiers
- int mknod(const char path, mode_t mode, dev_t dev) ; Pour les fichiers spéciaux

12.3 Read()

Include :

- #include ¡unistd.h¿

Déclarations :

- ssize_t read(int fd, void buf, size_t len) ;

Explication des paramètres :

- *fd :* "file descriptor" du fichier dans lequel il faut lire. (Il peut s'agir d'un fichier "standard" tout comme un substitut d'un pipe ou d'un network socket).
- *buf :* adresse du pointeur pour lequel l'input doit être copié.
- *len :* Nombre maximum de bytes à lire.

Retour :

- *ret :* le nombre de bytes effectivement lus.
- *0 :* La fin de fichier est atteinte.
- *-1 :* une erreur est apparue.

Remarques : Le retour peut (et est souvent) plus petit que la longueur attendue. Il faut donc être préparé à lire moins de données que ce qui était attendu.

12.4 Write()

Include :

- #include ¡unistd.h¿

Déclarations :

- ssize_t write(int fd, void buf, size_t len) ;

Explication des paramètres :

- *fd :* "file descriptor" du fichier dans lequel il faut écrire. (Il peut s'agir d'un fichier "standard" tout comme un substitut d'un pipe ou d'un network socket).
- *buf :* adresse du pointeur pour lequel l'output doit être écrit.
- *len :* Nombre maximum de bytes à écrire.

Retour :

- *ret :* le nombre de bytes effectivement lus.
- *0 :* La fin de fichier est atteinte.
- *-1 :* une erreur est apparue.

Remarques : Le retour peut (et est souvent) plus petit que la longueur attendue. Il faut donc être préparé à lire moins de données que ce qui était attendu.

12.5 Lseek()

Include :

- #include ¡sys/types.h¿
- #include ¡unistd.h¿

Déclarations :

- off_t lseek(int fd, int offset, off_t whence) ;

Explication des paramètres :

- *fd :* "file descriptor" du fichier ouvert. (Il peut s'agir d'un fichier "standard" tout comme un substitut d'un pipe ou d'un network socket).
- *offset :* Nombre de bytes qui doivent être déplacés (évités) par le pointeur.
- *whence :* point de départ (SEEK_SET : début du fichier, SEEK_CUR : à partir de la position courante, SEEK_END à partir de la fin du fichier).

12.6 Ioctl()

Pour toutes les autres opérations I/O.

Include :

- #include ¡sys/mtio.h¿

Déclarations :

- int ioctl(int fd, MTIOCTOP, (struct mtop) mt_cmd) ;

12.7 Close()

Permet de dissocier l'entier du descripteur de fichier. Il retourne 0 (en cas de succès) ou -1 (en cas d'erreur). Il faut savoir que tous les fichiers sont fermés lors de la fermeture du programme.

12.8 Dup()

Remarques : int fd2 = dup(int fd1), fd2 est une dupplication de fd1. fd2 partage le même fichier d'entrée que fd1 (Par cela on entend que fd1 et fd2 partage les même drapeaux statut, le même mode d'accès tout comme le même offset du fichier courant). fd2 sera le plus petit "file descriptor" disponible. fd1 pourra être fermé, par contre l'appel à dup peut échouer (errno : EBADF, EMFILE).

12.9 Dup2()

Dup2() est une opération atomique permettant de regrouper close(fd2) ainsi que fd2 = dup(fd1).

12.10 Mmap()

Include :
- #include ¡sys/types.h¿
- #include ¡sys/mman.h¿

Déclarations :
- caddr_t mmap(caddr_t addr, size_t len, int prot, int fd, int off) ;

Explication des paramètres :
- *addr :* devrait être 0.
- *len :* Nombre de bytes à mettre en mémoire.
- *prot :* Type d'accès(Peut être soit : PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE).
- *fd :* "File descriptor" à mettre en mémoire.
- *off :* Origine de la map dans fd.

Retour :
- *ret :* Retourne le pointeur de base permettant d'accéder aux données.
- *-1 :* une erreur est apparue.

Remarques : Le fichier doit être ouvert, plusieurs processus peuvent mettre en mémoire le même fichier, mmap est utilisé par UNIX afin de partager les librairies.

12.11 Munmap()

Include :
- #include ¡sys/types.h¿
- #include ¡sys/mman.h¿

Déclarations :
— int munmap(caddr_t addr, size_t len) ;

Explication des paramètres :
— *addr :* l'origine de la zone mémoire à enlever de la mémoire.
— *len :* Nombre de bytes à enlever de la mémoire.

Retour :
— *0 :* en cas de succès.
— *-1 :* une erreur est apparue.

Remarques : Détruit la mise en mémoire entre un fichier et l'espace adressé en mémoire virtuelle. Pour fermer un fichier, n'appeler surtout pas munmap.

12.12 Msync()

Include :
— #include ¡sys/types.h¿
— #include ¡sys/mman.h¿

Déclarations :
— int msync(caddr_t addr, size_t len, int flags) ;

Explication des paramètres :
— *addr :* l'origine de la zone mémoire à enlever de la mémoire.
— *len :* Nombre de bytes à enlever de la mémoire.
— *flags :* MS_ASYNC (retourne immédiatement), MS_INVALIDATE (permet d'invalider le "catching").

Retour :
— *0 :* en cas de succès.
— *-1 :* une erreur est apparue.

Remarques : Force le système à écrire les données mappées sur le disque.

12.13 Link()

Include :
— #include ¡unistd.h¿

Déclarations :
— int link(const char name1, const char name2) ;

Explication des paramètres :

— *name2 :* devient un autre nom pour l'objet déjà existant (name1). (Il s'agit donc juste d'un raccourci).

Retour :

— *0 :* en cas de succès.

— *-1 :* une erreur est apparue.

Remarques : Name1 et Name2 auront donc la même inode. (Il n'y a donc aucun moyen de savoir lequel des deux est le nom originel).

12.14 Unlink()

Include :

— #include ¡unistd.h¿

Déclarations :

— int unlink(const char name) ;

Explication des paramètres :

— *name :* sera supprimer du répertoire courant (si il s'agit de la dernière référence de l'objet, celui-ci sera également supprimé).

Retour :

— *0 :* en cas de succès.

— *-1 :* une erreur est apparue.

Remarques : La permision en écriture est nécéssaire sur le répertoire parent.

12.15 Symlink()

Include :

— #include ¡unistd.h¿

Déclarations :

— int symlink(const char name1, const char name2) ;

Explication des paramètres :

— *name2 :* sera un lien symbolique de l'objet référencé par name1.

Retour :

— *0 :* en cas de succès.

— *-1 :* une erreur est apparue.

Remarques : Similaire au lien physiques sans les limitations des frontières du sytème de fichier.

12.16 Readlink()

Include :
- #include ¡unistd.h¿

Déclarations :
- int readlink(const char path, void buf, size_t bufsize) ;

Explication des paramètres :
- *buf :* où mettre le résultat.
- *bufsize :* la taille du buffer.

Retour :
- *ret :* la longueur utilisée du buffer.
- *-1 :* une erreur est apparue.

Remarques : Lit la valeur d'un lien symbolique.

12.17 Stat()

Include :
- #include ¡sys/types.h¿
- #include ¡sys/stat.h¿

Déclarations :
- int stat(const char path, struct stat buf) ; retourne des informations au sujet d'un fichier.
- int lstat(const char path, struct stat buf) ; retourne des informations au sujet du lien symbolique lui même.
- int fstat(int fd, struct stat buf) ; =¿ retourne des informations au sujet de l'objet pointé par le lien symbolique.

Explication des paramètres :
- *buf :* pointeur vers une structure de type stat.

Retour :
- *ret :* les informations au sujet de l'objet spécifié.
- *-1 :* une erreur est apparue.

12.18 Rename()

Include :
— #include ¡stdio.h¿

Déclarations :
— int rename(const char old, const char new) ;

Retour :
— *ret :* la longueur utilisée du buffer.
— *-1 :* une erreur est apparue.

Remarques : Permet de remplacer l'ancien nom de fichier par le nouveau.

12.19 flock()

Include :
— #include ¡sys/files.h¿

Déclarations :
— int flock(int fd, int operation) ;

Explication des paramètres :
— *operation :* peut prendre les valeurs : (LOCK_SH qui signifie cadenas partagé (peut être utilisé par plus d'un processus), LOCK_EX qui signifie cadenas exclusif (peut être utilisé par un seul processus), LOCK_NB qui signifie que l'on ne doit pas bloqué lorsque le cadenas est pris (retournera -1 dans le cas où les fichier est bloqué) et LOCK_UN qui permet de délocker une ressource.

12.20 Lockf()

Include :
— #include ¡unistd.h¿

Déclarations :
— int lockf(int fd, int cmd, off_t len) ;

Explication des paramètres :
— *fd :* "file descriptor".
— *cmd :* peut prendre les valeurs : F_LOCK (permet de placer le cadenas), F_TLOCK (permet de placer le cadenas et de retourner 0 ou -1), F_ULOCK (permet de nettoyer le cadenas, de le remettre à zéro) et F_TEST (permet de tester le cadenas).
— *len :* longueur de la zone à placer sous cadenas.

12.21 Fork()

Include :
- #include ¡sys/types.h¿
- #include ¡unsistd.h¿

Déclarations :
- pid_t fork(void) ;

Retour :
- *ret :* le pid du fils dans le processus père.
- *0 :* au sein du fils.
- *-1 :* une erreur est apparue.

Remarques : Les deux processus ont les mêmes copies des données mémoires.De ce fait, ils partagent le même code, les même valeurs de variables, les mêmes fichiers ouverts, les mêmes pointeurs et les mêmes "file descriptor", le même répertoire de travail ainsi que le même terminal de controle. Mais ils ne partagent pas leur compteur de temps d'exécution, les valeurs des sémaphores, les cadenas sur les fichiers, ainsi que les signaux d'alarmes.

Cas spéciaux du fork :
- *Le processus parent termine avant le processus enfant :* l'enfant est dès lors attaché au processus init.
- *L'enfant termine avant le parent :* l'enfant devient dès lors un processus zombie (Il faut savoir qu'un processus enfant terminé continue d'exister tant que le parent ne se termine pas ou que le parent ne vérifie pas la valeur d'exit du processus enfant (wait(), ..)). Un processus zombie ne consume aucune ressource (à part peut être une entrée dans la table des processus).
- *Synchronisation :* Un processus parent doit parfois attendre que l'un de ses fils se termine. (Utilisation des appels systèmes wait() et exit()).

12.22 Wait()

Attend que l'un de ses fils se termine.

Include :
- #include ¡sys/types.h¿
- #include ¡sys/wait.h¿

Déclarations :
- pid_t wait(int status) ;
- pid_t waitpid(pid_t pid, int status) ;

Explication des paramètres :
— *status :* peut contenir des valeurs d'exit (lire le man).

Retour :
— *ret :* le pid du fils que l'on attendait.
— *-1 :* une erreur est apparue.

Remarques : Il peut attendre un fils spécifique et même en attendre plusieurs.

12.23 Exit()

Include :
— #include ¡unistd.h¿

Déclarations :
— int exit() ;

Retour :
— *0 :* en cas de succès.
— *-1 :* une erreur est apparue.

Remarques : L'appel système _exit(status) : Termine le processus courant immédiatement (les "file descriptor" ouverts sont fermés, les processus enfant sont envoyés à un autre processus (init), le processus parent recoit un signal SIGCHLD, le statut est retourné au processus parent). La fonction exit(status) : exécute les fonctions de fin de processus (tels que le flushing des streams) et appelle _exit(status). Dès lors la fonction exit devrait être privilégiée à l'appel système _exit().

12.24 Exec()

Include :
— #include ¡unistd.h¿

Déclarations :
— int execl(const char pahtname, const char arg0, ... (char) 0) ; le PATHNAME doit être exact.
— int execv(const char pathname, const char argv[]) ; Le PATHNAME doit être exact.
— int execle(const char pathname, const char arg0, ... (char) 0, const char envp[]) ;
— int execve(const char pathname, const char argv[], const char envp[]) ; Il s'aggit du seul appel système.

— int execlp(const char filename, const char arg0, (char) 0) ; On recherche le PATHNAME dans PATH.
— int execvp(const char filename, const char argv[]) ; On recherche le PATHNAME dans PATH.

Retour :
— *-1 :* une erreur est apparue.

Remarques : Il exécute un processus, il ne le démarre pas (Le processus est déjà lancé par fork()) exec va dépasser le processus courant par un nouveau. Il s'agit d'une famille de fonctions mais d'un appel système.

12.25 Pipe()

Include :
— #include ¡unistd.h¿

Déclarations :
— int pipe(int filedes[2]) ;

Explication des paramètres :
— *filedes[0] :* "file descriptor" ouvert pour lecture.
— *filedes[1] :* "file descriptor" ouvert pour écriture.

Retour :
— *0 :* en cas de succès.
— *-1 :* une erreur est apparue.

Errno :
— EMFILE
— ENFILE
— EFAULT

Remarques : Plutot inutile au sein d'un seul processus. Il faut le combiner avec fork pour le rendre vraiment efficace. Les données vont dans une direction, les pipes ne peuvent être utilisées que dans des processus ayant au moins un ancêtre commun. (Il faut les paramétrer avant le fork). La synchronisation est possible par : le blocage du reader lorsque le pipe est vide, le blocage du writer lorsque le pipe est plein. Un processus pourrait être à la fois writer et reader. Les processus doivent par contre être d'accord sur la taille et le format des messages échangés. Les "file descriptor" qui ne sont pas utilisés doivent être fermés afin d'empêcher un deadlock d'apparaitre.

popen() : permet de simplifier le processus d'envoi de données à un autre processus (Il faut inclure ¡stdio.h¿ et il se déclare : FILE popen(const char cmd, const char type) ;). Il s'utilise avec pclose(FILE stream).

12.26 Kill()

Include :

— #include ¡signal.h¿

Déclarations :

— int kill(int pid, int sig) ;

Explication des paramètres :

— *pid :* processus de destination ou groupe du processus (pid ¿ 0 le signal est envoyé au processus, pid = 0, le signal est envoyés à tous les processus du groupe de l'appellant, pid = -1 le signal est envoyé à tous les processus (à part init et les processus des autres utilisateurs), pid ¡ -1 : le signal est envoyé à tous les processus du groupe du processus). On peut obtenir le pid à l'aide de fork (valeur de retour), getpid (retourne mon propre pid) ou getppid (retourne le pid du processus parent).

— *sig :* le nom du signal ou son nombre (Si sig = 0 aucune action n'est effectuée).

Errno :

— EINVAL : Le numéro de signal est invalide.

— ESRCH : Le pid n'existe pas.

— EPERM : La permission est refusée.

Retour :

— *0 :* en cas de succès.

— *-1 :* une erreur est apparue.

12.27 Signal()

Include :

— #include ¡signal.h¿

Déclarations :

— sighandler_t signal(int signum, sighandler_t handler) ;

Explication des paramètres :

— *signum :* numéro du signal.

— *handler :* Peut prendre les valeurs : SIG_DFL (signal par défaut avec un gestionnaire par défaut), SIG_IGN (permet d'ignorer le signal) ou tout simplement une valeur spécifiée par l'utilisateur.

Remarques : Queue : seulement un signal de chaque type peut être enregistré pour un processus (Si il y en a plusieurs, le processus n'en gère qu'un, les autres seront dès lors perdus).

- *SYSV :* Lorsque le signal est traité, les gestionnaires sont remis à défaut.
- *BSD :* ne remet pas le gestionnaire à défaut mais bloque les signaux tant que le gestionnaire n'a pas finis son traitement (pas de défausse, mais mise en queue, ils seront juste remis par la suite).

Vulnérabilité : une fenêtre e vulnérabilité existe pour cela il faudra utiliser sigaction qui permet d'utiliser un masque durant le traitement du gestionnaire. Il utilisera le comportement BSD et bloquera les autres signaux durant le traitement du gestionnaire.

12.28 Alarm()

Include :

- #include ¡unistd.h¿

Déclarations :

- unsigned int alarm(unsigned int seconds) ;

Remarques : Envoie le signal SIGALRM après quelque secondes.

12.29 msgget()

Déclarations :

- int qid = msgget(KEY,mode—flags) ;

Explication des paramètres :

- flags peut prendre comme valeur soit : IPC_CREATE (cela créera la structure si elle n'existait pas déjà et ouvrira la structure si celle ci existait déjà). soit IPC_EXCL (crée la structure si elle n'existe pas et envoie une erreur ou sinon).

12.30 msgsnd()

Déclarations :

- int ret = msgsnd(int qid, const void *ptr, size_t nbytes, int flag) ;

Explication des paramètres :

- qid : il s'agit de l'identifiant de la structure (l'id de la structure qid).
- ptr : un pointeur vers une structure contenant un long suivis des données du message.
- nbytes : la taille de la zone de texte (doit être plus petit que MAX_LENGTH).
- flags peut prendre comme valeur IPC_NOWAIT

12.31 msgrcv()

Déclarations :

— int ret = msgrcv(int qid, void *ptr, size_t nbytes, long type, int flag) ;

Explication des paramètres :

— qid : il s'agit de l'identifiant de la structure (l'id de la structure qid).
— ptr : un pointeur vers une structure contenant un long suivis des données du message.
— nbytes : la taille du buffer (doit être égale à BUF_MAX_LENGTH). Le message pour sa part est censé être plus petit que BUF_MAX_LENGTH.
— type : le type peut être égal à 0 (le premier message de la file est lu), au dessus de 0 (le premier message étant égal à type est lu) ou peut être en dessous de 0 (le premier message ayant un type plus petit ou égal à la valeur absolue de type est lu).
— flags peut prendre comme valeur IPC_NOWAIT (ne bloquera pas si la file est vide) ou MSG_NOERROR (qui permet de tronquer les messages).

12.32 msgctl()

Déclarations :

— int msgctl(int qid, int cmd, struct msqid_ds *ptr) ;

Explication des paramètres :

— qid : il s'agit de l'identifiant de la structure (l'id de la structure qid).
— ptr : un pointeur vers une structure IPC : msqid_ds.
— cmd qui peut prendre comme valeur : IPC_STAT (retourne la structure de controle), IPC_SET (qui permet d'écrire : msg_perm.uid, msg_perm.gid, msg_perm.mode et msg_qbytes à la structure de contrôle) et IPC_RMID (permet de supprimer la pile de message).

12.33 semget()

Déclarations :

— int semget(key_t key, int nsems, int semflg) ;

Explication des paramètres :

— key : un nombre représentant une clé.
— nsems : le nombre de sémaphores de l'ensemble.
— semflg : peut prendre comme valeur soit : IPC_CREATE (cela créera la structure si elle n'existait pas déjà et ouvrira la structure si celle ci existait déjà). soit IPC_EXCL (crée la structure si elle n'existe pas et envoie une erreur ou sinon).

Type de retour : un ensemble de sémaphore (plutôt leur identifiant), il faut par contre faire attention car ces sémaphores sont créés et non initialisées.

12.34 semctl()

Déclarations :
- int semctl(int semid, int semnum, int cmd, arg) ;

Explication des paramètres :
- semid : l'identifiant de l'ensemble de sémaphore.
- semnum : le numéro de la sémaphore au sein de son ensemble.
- cmd : Il peut prendre comme valeur :
  - GETVAL/SETVAL : permet d'obtenir / de changer la valeur d'une sémaphore.
  - GETALL/SETALL : permet d'obtenir / de changer la valeur de toutes les sémaphores.
  - GETPID : permet d'obtenir l'id du dernier processus à avoir utiliser la sémaphore.
  - GETNCNT : permet d'obtenir le nombre de processus attendant sur la sémaphore (valeur ¿ valeur actuelle).
  - GETZCNT : permet d'obtenir le nombre de processus attendant sur la sémaphore (valeur = 0).
  - IPC_STAT : retourne semid_ds à l'intérieur d'un pointeur passé à l'aide de arg.
  - IPC_SET : modifie le uid et git effectif ainsi que les modes à travers ipc_perm de semid_ds.
  - IPC_RMID : supprime l'ensemble de sémaphores.
- arg : depend de cmd.

12.35 semop()

Déclarations :
- int semop(int semid, struct sembuf sops[], int nsops) ;

Explication des paramètres :
- semid : l'identifiant de l'ensemble de sémaphore.
- sops : le tableau représentant les sem ops.
- nsops : nombre de sem ops.

Sembuf : sembuf décrit une opération sur une sémaphore.
struct sembuf
ushort sem_num ; // numéro de la sémaphore
short sem_op ; // Opération sur la sémaphore (+1, -1)
short sem_flg ; // flag de la sémaphore
;

Remarques : Semop permet d'effectuer des opérations sur un ensemble de sémaphores de manière atomique.

Opération sur la sémaphore :

- sem_op ¿ 0 : la valeur est ajoutée à la sémaphore courante.
- sem_op ¡ 0 : si —sem_op— ¡= à la valeur de la sémaphore, —sem_op— est soustrait de la valeur actuelle de la sémaphore, autrement si —sem_op— ¿ la valeur de la sémaphore alors semncnt++, le processus courant est endormis jusqu'a ce que —sem_op— ¡= la valeur de la semaphore (dès que tel est le cas, soustrait cette valeur à la valeur de la sémaphore).
- sem_op == 0 : si la valeur de la sémaphore != 0 : semznct++, le processus attend tant que la valeur de la sémaphore n'égale pas 0).

Valeur de sem_flg : Il peut prendre comme valeur : IPC_NOWAIT si la condition ferait en sorte que le processus s'endorme : l'appel système se termine directement avec comme code d'erreur : EAGAIN. La deuxième valeur qu'il peut prendre est : SEM_UNDO (Toute les modifications sur la valeur de la sémaphore du processus courant sont sauvegardées afin d'être effectuées à nouveau à la fin du processus courant.

12.36 shmget()

Déclarations :

- int shmget(key_t key, int size, int shmflg) ;

Explication des paramètres :

- key : un nombre représentant une clé.
- nsems : taille de la mémoire partagée (LA taille minimum du segment, seulement utile pour le processus créateur de la sharedMemory.
- shmflg : peut prendre comme valeur soit : IPC_CREATE (cela créera la structure si elle n'existait pas déjà et ouvrira la structure si celle ci existait déjà). soit IPC_EXCL (crée la structure si elle n'existe pas et envoie une erreur ou sinon).

12.37 shmctl()

Déclarations :

- int shmctl(int shmid, int cmd, struct shmid_ds buf) ;

Explication des paramètres :

- shmid : L'identifiant de la mémoire partagée.
- cmd : la commande a exécuté sur shm.
  - SHM_STAT : recupère la structure shmid_ds depuis le kernel.
  - SHM_SET : permet de mettre les champs autorisés de shmid_ds.
  - SHM_RMID : détruit la méoire partagée (Si tous les processus utilisant la mémoire partagée s'en détache alors cela l'a détruit également).
  - SHM_LOCK, SHM_UNLOCK : permettent de bloquer / débloquer la mémoire partagée dans la mémoire physique.

— shmflg : ci requis avec cmd.

12.38 shmat()

Déclarations :
— void shmat (int shmid, void addr, int flag)

Explication des paramètres :
— shmid : L'identifiant de la mémoire partagée.
— addr : devrait être NULL dans les applications actuelles.
— flag : SHM_RDONLY.

Retour : l'adresse mémoire que le processus appelant pourra utiliser.

void shmdt(addr) : permet de détacher un processus de la mémoire partagée à laquelle il est relié.

13 Unix Journey

Il s'agit d'une partie reprenant plutôt les commandes admins et ce que l'utilisateur utilise.

13.1 Se déplacer entre les répertoires

— cd
— cd . (Ne sert à rien (se déplacer jusqu'au répertoire courant revient à ne pas se déplacer du tout).
— cd .. : se déplacer vers le répertoire parent.
— cd ~ : se déplacer vers le répertoire login
— cd user : se déplacer au sein du répertoire user présent au sein du répertoire courant.

13.2 Lister les fichiers

— ls (Il s'agit d'une alias pour /bin/ls
— ls -l : En plus du nom, affiche le type du fichier, les permissions d'accès, le nombre de liens physiques, le nom du propriétaire et du groupe, la taille en octets, et l'horodatage (de la dernière modification, sauf si une autre date est réclamée par les options -c ou -u).
— ls -1 : Présente un fichier par ligne.
— ls -a : Affiche tous les fichiers des répertoires, y compris les fichiers commençant par un '.'.
— ls -c : Trier le contenu des répertoires en fonction des dates de changement des statuts de fichiers, plutôt qu'en fonction de la date de modification.

S'il s'agit d'un format long, afficher la date de changement de statut plutôt que la date de modification du fichier.
- ls -d : Afficher les répertoires avec la même présentation que les fichiers, sans lister leur contenu.
- ls -i : Afficher le numéro d'index (i-noeud) de chaque fichier à gauche de son nom.
- ls -r : Inverse le tri du contenu des répertoires.
- ls –t : Trie le contenu des répertoires en fonction de la date et non pas en ordre alphabétique. Les fichiers les plus récents sont présentés en premier.
- ls –u : Trie le contenu des répertoires en fonction de la date de dernier accès au fichier plutôt que selon la date de modification. Si le format d'affichage est large, c'est la date de dernier accès qui est affichée.
- ls -C : Présente les fichiers en colonnes, triés verticalement.
- ls -s : permet de lister la taille des fichiers.
- ls -F : Ajoute un caractère à chaque nom de fichier pour indiquer son type. les fichiers réguliers exécutables sont suivis d'un '*'. Les répertoires sont suivis de '/', les liens symboliques d'un '@', les FIFOs d'un '—'. Les fichiers réguliers non-exécutables ne sont suivis d'aucun caractère.
- ls - R : Affiche récursivement le contenu des sous-répertoires.
- ls -s — more –c (note different output - ls use isatty())
- ls -S : Trie par taille de fichier.

13.3 Temps des fichiers

Structure stat :
- int stat(const char file_name, struct stat buf) ; : récupère le statut du fichier pointé par file_name et remplit le buffer buf.
- int fstat(int filedes, struct stat buf) ; : est identique à stat, sauf que le fichier ouvert est pointé par le descripteur filedes, obtenu avec open(2).
- int fstat(int filedes, struct stat buf) ; : est identique à stat, sauf que dans le cas d'un lien symbolique, il donne l'état du lien lui-même plutôt que celui du fichier visé.

Temps des fichiers :
- Access time (atime) : Temps depuis la dernière fois que le fichier a été accédé (Read Time).
- Modified time (mtime) : Temps depuis la dernière fois que le fichier a été modifié (Write Time).
- Change time (ctime) : Temps depuis la dernière fois que l'inode a été modifié (Inode Change).

13.4 Déplacer / Copier des fichiers

mv : Permet de déplacer ou de renommer des fichiers. (L'inode peut changer si on change fs).

cp : Permet de copier des fichiers. Options disponibles :

— -f : Efface les fichiers cibles existants (voir ci-dessus).
— -i : Interroge l'utilisateur avant d'écraser des fichiers réguliers existants. La question est affichée sur stderr, et la réponse lue depuis stdin.
— -p Conserve le propriétaire, le groupe, les permissions d'accès et les horodatages du fichier original. Si la conservation du propriétaire ou du groupe est impossible, les bits Set-UID et Set-GID sont effacés. L'horodatage sera quand même légèrement différent entre l'original et la copie car l'opération de copie nécessite un accès en lecture au fichier source.
— -R : Copie récursivement les répertoires, et gérer correctement les copies des fichiers spéciaux ou des FIFOs. -r : Copier récursivement les répertoires mais la gestion des fichiers spéciaux n'est pas définie. En fait l'option -r est autorisée, et même encouragée à se comporter comme l'option -R, toutefois le comportement (stupide) de la version GNU n'est pas interdit.
— – : Indique la fin explicite de la liste des options.

13.5 Droits d'utilisateur et de groupe

Fichiers importants : /etc/passwd (information sur les comptes utilisateurs), etc/group (information de groupes), /etc/shadow (information sécurisée sur les comptes utilisateurs), /etc/gshadow.

useradd : Crée un nouvel utilisateur ou mets à jour l'information par défaut sur un nouvel utilisateur.

groupadd : Crée un nouveau groupe ou mets à jour l'information par défaut sur un nouveau groupe.

chown : Modifie le propriétaire et le groupe d'un fichier.

chgrp : Change le groupe propriétaire d'un fichier.

13.6 Types de fichiers et protection

Types de fichier : Répertoires, fichier régulier, lien symbolique, pipes, sockets,(block/ character) devices.

Bits spécifiques :
— setuid : spécifique aux exécutables.
— setgid : spécifique aux répertoires.
— sticky bit pour les fichiers.
— sticky bits pour les répertoires.

Cas spécifiques : rwx pour les répertoires.

13.7 Espace Disque :

Espace disque des fichiers : Les fichiers prennent de la place sur le disque. Afin de connaitre la place que prend un répertoire, on peut utiliser la commande du (Donne les statistiques sur l'utilisation du disque, donne l'espace du répertoire et non l'espace utilisé par un seul utilisateur). Options pouvant être utilisée avec ce dernier :

- -a : Affiche les statistiques pour tous les fichiers, pas seulement les répertoires.
- -k : Affiche la taille en ko (kilo-octets, 1024 octets).
- -s : Affiche seulement le total pour chaque argument.
- -x : Ignore les répertoires situés sur un système de fichiers différent de celui de l'argument étudié.

Remarques : au sujet des liens physiques : ils ne sont comptés qu'une seule fois.

Espace disque utilisé d'une partition : on utilisera la commande df (Fournit la quantité d'espace occupé des systèmes de fichiers, elle permet de savori quel est l'espace disponible) Options pouvant être utilisée avec ce dernier :

- -k : Utilise des unités de 1024 octets plutôt que les unités de 512 octets par défaut.
- -P : Affichage sur six colonnes, précédées de l'en-tête 'Filesystem N-blocks Used Available Capacity Mounted on' (avec N=512, ou N=1024 si l'option -k est utilisée).
- -h : NOT FOUND
- -H : NOT FOUND
- -P : NOT FOUND
- -t : NOT FOUND
- – : Fin explicite de la liste des options.

13.8 Trouver des fichiers

Afin de trouver des fichiers on utilisera l'appel système find. Ce dernier peut avoir pour option :

- -type c Fichier du type c :
  - b : fichier spécial en mode bloc (avec tampon)
  - c : fichier spécial en mode caractère (sans tampon)
  - d : répertoire
  - p : tube nommé (FIFO)
  - f : fichier régulier
  - l : liens symbolique
  - s : socket
- -perm mode : Fichier dont les autorisations d'accès sont fixées exactement au mode indiqué (en notation symbolique ou octale). La notation symbolique utilise le mode 0 comme point de départ.
- -perm -mode : Fichier ayant au moins toutes les autorisations indiquées dans le mode.

— -perm +mode : Fichier ayant certaines des autorisations indiquées dans le mode.
— -mtime n : Fichier dont les données ont été modifiées il y a n*24 heures.
— -atime n : dernier accès au fichier il y a n*24 heures.
— -ctime n : dernière modification du statut du fichier il y a n*24 heures.
— -size n[bckw] : Fichier utilisant n unités d'espace. Les unités sont des blocs de 512 octets par défaut (ou si un suffixe 'b' suit le nombre n), des octets si un suffixe 'c' suit n, des kilo-octets si un suffixe 'k' est utilisé, ou des mots de 2 octets si un 'w' suit le nombre n. La taille ne prend pas en compte les blocs indirects, mais elle comptabilise les blocs des fichiers éparpillés pas encore alloués.
— -name motif : Fichier dont le nom de base (sans les répertoires du chemin d'accès), correspond au motif du shell. Les méta-caractères ('*', '?', et '[]') ne sont jamais mis en correspondance avec un point '.' au début du nom. Pour ignorer un répertoire, ainsi que tous ses sous-répertoires, utiliser l'option -prune ; vous trouverez un exemple dans la description de l'option -path.
— -path motif : Fichier dont le nom complet correspond au motif fourni. Lors du développement des méta-caractères, '/' et '.' ne sont pas traités différemment des autres caractères, ainsi par exemple : find . -path './sr*sc' affichera l'élément de répertoire intitulé './src/misc' (s'il en existe un). Pour ignorer une arborescence complète de répertoires, utilisez l'option -prune plutôt que de vérifier chaque fichier de l'arbre. Par exemple, pour ignorer le répertoire 'src/emacs' et tous ses sous-répertoires, tout en affichant le nom de tous les autres fichiers, faites quelque chose comme : find . -path './src/emacs' -prune -o -print
— -user utilisateur : fichier appartenant à l'utilisateur indiqué (U-ID numérique éventuellement)
— -nouser : Fichier dont l'U-ID numérique ne correspond à aucun utilisateur.
— group nom_groupe : fichier appartenant au groupe nom_groupe (éventuellement ID numérique).
— -nogroup : Fichier dont le G-ID numérique ne correspond à aucun groupe d'utilisateurs.

Remarque : Les arguments numériques peuvent être indiqués comme suit : +n (supérieur à n), -n (inférieur à n) et n (égal à n). Actions pouvant être utilisée sur find :

— -exec commande : Exécute la commande ; vrai si le code de retour 0 est renvoyé. Tous les arguments suivants de find sont considérés comme des arguments pour la ligne de commande, jusqu'à ce qu'on rencontre un ' ;'. La chaîne '' est remplacée par le nom du fichier en cours de traitement, et ceci dans toutes ses occurrences, pas seulement aux endroits où elle est isolée, comme c'est le cas avec d'autres versions de find. Ces deux chaînes peuvent avoir besoin d'être protégées du développement de la ligne de commande par le shell, en utilisant le caractère d'échappement

(‘) ou une protection par des apostrophes. La commande est exécutée depuis le répertoire de départ.
— -ls : NOT FOUND
— -delete : NOT FOUND
— -execdir : NOT FOUND

13.9 Comparaison de fichiers

On peut pour cela utiliser la commande diff (Trouve les différences entre des fichiers). Options pouvant être utilisées sur ce dernier :
— -nb (nb est un nombre entier) Afficher nb lignes de contexte. Cette option ne précise pas le format de sortie par elle-même, elle n'a pas d'effet si elle n'est pas combinée avec -c ou -u. Cette option est obsolète. Pour fonctionner correctement, patch nécessite typiquement deux lignes de contexte.
— -a : Traiter tous les fichiers comme du texte, et les comparer ligne-à-ligne, même s'ils semblent contenir des données binaires.
— -b : ne pas tenir compte des différences concernant des espaces blancs.
— -B : Ne pas tenir compte des différences qui concernent des lignes blanches.
— –brief : Indique seulement si les fichiers diffèrent, pas les différences elles-mêmes.
— -c : Utiliser le format de sortie contextuel.
— -C nb

13.10 Fichiers d'archives / de compression

tar : la version GNU de l'utilitaire tar de gestion d'archives.

compress : Compresse ou décompresse des fichiers.

gzip : Compacte ou décompacte des fichiers.

13.11 Gestion des processus

Trouver des processus : Afin de toruver des processus, on va utiliser la commande ps (Affiche l'état des processus en cours). Celle ci peut avoir comme options :
— l : affichage long
— u : (utilisateur) présente le nom de l'utilisateur et l'heure de lancement.
— j : (job) présente les Pgid et Sid.
— s : (signal) présente les signaux bloqués, ignorés et interceptés.
— v : (vm) affiche des informations sur la mémoire virtuelle.
— m : (mémoire) affiche des informations sur l'occupation mémoire. À combiner avec l'option p pour obtenir les nombres de pages.
— f : (forêt) affiche les arbres généalogiques des processus.
— a : (autres) présente également les processus des autres utilisateurs.

— x : affiche les processus qui n'ont pas de terminal de contrôle.
— S : additionne les temps CPU et les fautes de pagination des processus fils.
— c : (commande) affiche le nom de la commande exécutée.
— e : (environnement) présente l'environnement à la suite de la ligne de commande exécutée.
— w : (wide) affichage large. Ne coupe pas les informations pour les limiter à une seule ligne. Pour être exact, chaque 'w' autorise une ligne supplémentaire pour chaque processus. Si la ligne supplémentaire n'est pas nécessaire, elle n'est pas utilisée. Il peut y avoir jusqu'à 100 w.
— h : ne pas afficher d'en-tête.
— r : ne présenter que les processus en cours d'exécution (running).
— n : Affichage numérique des champs USER et WCHAN.

nice : Exécute un programme avec une priorité d'ordonnancement modifiée.

lsof : liste les fichiers ouverts.

13.12 Filtres élémentaires

Commande grep : Affiche les lignes correspondant à un motif donné. Elle peut être utilisée avec les options :

— -num : Les correspondances seront affichées avec num lignes supplémentaires avant et après. Néanmoins, grep n'affichera jamais une ligne plus d'une fois.
— -A num : Affiche num lignes supplémentaires après la ligne correspondante.
— -B num : Affiche num lignes supplémentaires avant la ligne correspondante.
— -C : est équivalent à -2.
— -V : Affiche le numéro de version de grep sur la sortie d'erreur standard. Ce numéro de version devra être inclus dans tous les rapports de bogues (voir plus bas).
— -b : Avant chaque ligne, afficher son décalage (en octet) au sein du fichier d'entrée.
— -c : Ne pas afficher les résultats normaux. À la place, afficher un compte des lignes correspondantes pour chaque fichier d'entrée. Avec l'option -v (voir plus bas), afficher les nombres de lignes ne correspondant pas au motif.
— -e motif : Utilise le motif indiqué. Ceci permet de protéger les motifs commençant par -.
— -f fichier : Lit le motif dans le fichier indiqué.
— -h : Ne pas afficher le nom des fichiers dans les résultats lorsque plusieurs fichiers sont parcourus.
— -i : Ignore les différences majuscules/minuscules aussi bien dans le motif que dans les fichiers d'entrée.

— -L : Ne pas afficher les résultats normaux. À la place, indiquer le nom des fichiers pour lesquels aucun résultat n'aurait été affiché.
— -l : Ne pas afficher les résultats normaux. À la place, indiquer le nom des fichiers pour lesquels des résultats auraient été affiches.
— -n : Ajoute à chaque ligne de sortie un préfixe contenant son numéro dans le fichier d'entrée.
— -q : Silence. Ne pas afficher les résultats normaux.
— -s : Ne pas afficher les messages d'erreurs concernant les fichiers inexistants ou illisibles.
— -v : Inverser la mise en correspondance, pour sélectionner les lignes ne correspondant pas au motif.
— -w : Ne sélectionne que les lignes contenant une correspondance formant un mot complet. La sous-chaîne correspondante doit donc être soit au début de la ligne, soit précédée d'un caractère n'appartenant pas à un mot. De même elle doit se trouver soit à la fin de la ligne, soit être suivie par un caractère n'appartenant pas à un mot. Les caractères composants les mots sont les lettres, les chiffres et le souligné ('_'). ([NDT] Bien entendu les minuscules accentuées ne sont pas des lettres ! Elles servent donc à séparer les mots...)
— -x : Ne sélectionne que les correspondances qui occupent une ligne entière.

dirname : Ne conserve que la partie répertoire d'un chemin d'accès.

basename : Élimine le chemin d'accès et le suffixe d'un nom de fichier.

awk : Langage d'examen et de traitement de motifs.

tr : Transpose ou élimine des caractères.

sort : Trie les lignes d'un fichier texte.

head -n : Affiche le début d'un fichier (-n : nombre de lignes).

tail -n : Affiche la dernière partie d'un fichier (-n : nombre de lignes).

xargs : construire et exécuter des lignes de commandes à partir de l'entrée standard.

13.13 Plus de filtres

Editors : Discussion au sujet des éditeurs (dd, ex, vi, emacs, nemacs, pico).

Différences cat et tac : cat : Concatène des fichiers et les affiche sur la sortie standard. tac : Concatène et affiche des fichiers à l'envers.

13.14 Shells

Le shell est l'interpréteur de commandes (Il n'y a pas qu'un seul interpréteur de commandes : sh, csh, ksh, bash, tcsh, zsh, ce qui fait que chaque interpréteur à ses propres particularités, tels que la syntaxe de programmation, ou son historique), il s'agit également du nom générique d'un command file. Le shell par défaut est le dernier champ de /etc/passwd (Les shells valides sont repris dans le fichier : /etc/shells).

13.15 Bash Programming

Utilisation de variables : Les variables peuvent être locales ou globales. Elles peuvent également être prédéfinies :

- $$ (pid)
- $
- $?
- $0
- $#
- $1 $2 $3 $4 ...
- $BASH_VERSION

13.16 Shell Escaping

On peut procéder a l'échappement de caractères à l'aide de :

- /
- simple quotes
- double quotes
- prargs

14 Liens Externes