*Système d’exploitation :* un ensemble de programmes permettant á un utilisateur l’exploitation des différence ressources matérielles d’une machine donnée, se compose la plupart du temps d'un noyau d'un interpréteur de commande shell et d'un système de fichiers.

*Le noyau d’un système d’exploitation :* est le logiciel qui assure :

* La communication entre les logiciels et le matériel ;
* La gestion des divers logiciels ([tâches](https://fr.wikipedia.org/wiki/Processus_(informatique))) d’une machine (lancement des programmes, ordonnancement…) ;
* La gestion du matériel (mémoire, processeur, périphérique, stockage…).

*Type de noyau:*

* Les noyaux monolithiques placent un maximum de programmes systèmes dans l'espace noyau, (Unix, linux).
* Les micro-noyaux préfèrent au contraire placer le plus de choses dans l'espace utilisateur (Mac OS, windows nt)
* Les noyaux hybrides, est un intermédiaire entre le noyau monolithique et le micro-noyau.

Différence entre noyaux monolithique et les micro-noyaux : Le noyau monolithique est un processus unique et volumineux s’exécutant entièrement dans un seul espace d’adressage, contrairement au micro-noyau, les services utilisateur et les services du noyau sont implémentés sous le même espace d’adressage

*Appel système :* est une fonction appeler au sein d’un programme d’application pour lui permettre de communiquer avec le noyau, qui de son tour communique avec les ressources matérielles.

*Outils de développement:*

* Editeur de texte: permet de crée et de modifier les fichier sources. (vi)
* Compilateur: permet de passer d'un fichier source a un fichier .obj (Gcc)
* Editeur de lien: associer les fichiers .obj avec les bibliothèques utilisées, le résultat est un fichier exécutable. (gnu debugger)
* Déboguer : permet l'exécution pas à pas du code, pour cela il a besoin du code source et de fichier exécutable.

*Les processus :*

Un processus est un programme en train d'exécution, chaque processus est lancé par un autre processus père, le processus init est la racine des processus et il a le pid 1, lui aussi est lance par le processus inactif du system

Demons: des processus system, sont tjrs en activité, leur rôle est de fournir des services system.

*Les états de processus:*

* Running R: processus en cours d'exécution.
* Sleeping S: en attente d'un évènement comme la fin d'une entre sortie,
* Stopped T: processus a été temporairement arrêté par un signal,
* Zombie Z: le processus s'est terminé, mais son père n'a pas encore reçu son code de retour.

*PCB processus control Block:*

Le system utilise une table des processus où chaque processus possède une entrée où il y’a les informations suivantes:

* Le pid, ppid, uid, gid de processus
* Les fichiers ouverts par ce processus, dossier courant, tty de processus, mémoire dédier au processus, signaux reçus

*Les fonctions des processus*

Getpid (), return le pid de processus pid\_t;

Getppid (), return le pid du père;

L’appel system fork : permet de créer un processus fils, qui possède le même code que son père.

L’appel système Fork (), retourne 0 pour le processus fils, pid du processus fils pour le processus père, -1 en cas d'erreur.

*En cas d'erreur:*

errno = ENOMEM -> Sys n'a pas assez de mémoire disponible pour créer un processus

errno = EAGAIN ->processus père a déjà trop de processus en cours d'exécution.

exit(int statu)-> permet de quitter un processus immédiatement, le type de return est void.

wait(int\* statu) permet d’attendre la mort d'un fils: return le pid de fils mort ,return -1: si le processus n'a pas de processus fils à attendre.

WIFEXITED(int statu) return true si le fils est mort d’une façons normal.

WEXITSTATUS(int statu) return l'argument de la fct exit utilisé par le fils, on l'utilise en cas de WIFEXITED = TRUE.

*PASSER LES ARGUMENTS Á UN PROGRAMME*

On ajoute 2 arguments a la fct main (int argc, char\* argv[]) -> pour consulter les arguments passer en ligne de commande on utilise ce code :

For (i=1; i< argc ; i++) printf("argument %d = %s ",i , argv[i]);

*Variables d'environnement:*

Pour passer les variables d'environnement au programme on déclare un tableau de string globale:

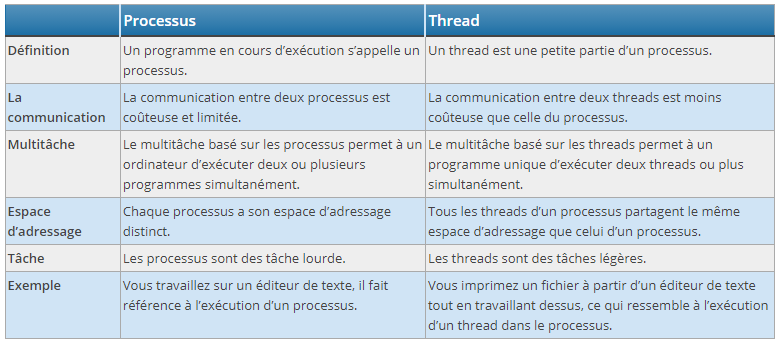
extern char \*\*environ , le dernier élément de ce tableau = NULL;

*Appel EXEEC:*

En faisant appel à exeec, en remplace notre programme par une commande ou un script Shell, le code restant ne s'exécute pas,

* execl(char\* path,char\* argv1,char\* argv2,NULL);example exeec("\bin\ls","ls","-l",NULL).
* execv(char\* path,char\* argv[]) avec argv[] = ("ls","-l",NULL);
* execlp(char\* nomdefichier,char\* arg1,...,NULL) execlp("ls","ls","-l",NULL) pour évité de spécifier le chemin des commandes.

*Threads:*



Plusieurs fils d'exécution qui partage le même espace d'adressage et les mêmes ressources, et chaque thread a sa propre pile et son propre compteur ordinal, par contre les processus à son propre espace d’adressage et ses propres ressources.

* Le processus légère s'exécutes même si certaine partie sont bloquées.
* Partage même ressources, économie d'espace mémoire est du temps.

Biblio <pthread.h> il faut ajouter -lpthread lors de la compilation d’un programme qui utilise les threads :

* pthread\_self():permet de connaitre l'identifiant du thread
* pthread\_create(&thread, NULL, &fonction ,argument fct(NULL)) associer une fonction á un thread.
* pthread\_join(thread, NULL(valeur de return de la fct de thread) attendre un thread pour quelle termine son exécution.
* pthread\_exit(&valeur de retour(NULL)) arrête immédiate du thread appelant.

*L'ordonnancement:*

L’ordonnancement : c'est l'effet de gérer l'utilisation de processeur entre les processus en état prêt.

* Efficacité: le processeur doit travailler à 100% de temps.
* Rendement: exécuter maximum des programmes en un temps donné.
* Temps de réponse: le temps moyen pour répondre aux entrées de l'utilisateur (systeme interactive).
* Temps d'exécution: chaque programme doit s'exécuter le plus vite possible.
* Equité: partager le temps processeur en équilibre entre les processus
* Le temps de séjour: en soustrait le temps d'entrée du temps de terminaison(sortie);
* Le temps d'attente: le temps de séjour moins le temps d'exécution,

*2 familles d'ordonnanceur:*

* Non-préemptif: on passe au processus suivant en cas de terminaison ou du blocage de processus courant.
* Préemptif : á un temps donné, l'ordonnanceur prend la main et choisie un autre processus.
  + Fifo : non préemptif -> premier arriver, premier servie,
  + SJF: short first job -> on choisit le processus a le moins temps d'exécution, en cas d’Egalite on revient au fifo, temps de réponse moins que fifo.
  + SRT: shortest remaing time, on compare le temps d'exécution de processus arrivant avec le processus actuel, on choisit le plus court.
  + Avec priorité: on affecte une valeur à chaque processus, le processus élu est le processus qui a la valeur la plus élevée.

***Gestion des fichiers:***

Le fichier c'est l'unité de stockage sur le support de masse (disque dur,CD-ROM).

*Le système de fichier SGF:*

Permet la gestion des fichiers sous un système d’exploitation :

Gestion point de vue utilisateur : nommage, protection et droit d’accès, opérations autorisées sur les fichiers.

Gestion Point de vue de l'implantation: organisation d'un fichier sur un disque, gestion des blocs physique d'un fichier gestion de l'espace libre.

*Gestion d'espace disque:*

Un fichier est constitué d'un ensemble de blocs physique.

* *Allocation contigu:* le fichier occupe un ensemble des blocs contigus sur le disque. (A comme difficulté, la fragmentation: trouver de l'espace suffisant pour l'allocation. Le compactage: regrouper l'espace libre, dans un seul coté).
* *Allocation par bloc chaine:* un fichier est représenté par une liste chainée des blocs, qui sont dispersé sur le disque.
* *Allocation indexé*: un fichier représenter par un tableau qui contient ses blocs.

*Gestion de l'espace libre:*

Approche statique: le système utilise une table de bits dans laquelle le bloc libre représenter par le bit 0, et le bloc non libre représenter par 1.

Approche dynamique: utiliser une liste chainée qui mémorise les blocs libres.

*Le répertoire:* structure arborescente qui contient une table d'entrée, chaque entrée correspond à un fichier, il stock pour chaque fichier l'adresse des blocs allouer à ce fichier.

*Fichier linux:*

Utilise allocation indexé, se compose d’un descripteur appelé inode, et des blocs physiques, chaque bloc code sur 4 octet, la taille du bloc est multiple d'un secteur 512 octet 🡪 1024.

*Inode :* structure crée lors de l’allocation, repérer par un numéro qui contient les infos suivant: nom, type, droit, taille, tables index des bloc, heures diverses.

Tableau des blocs contient 15 entrée, 12 premier a accès direct au blocs, ce qui reste aussi des tables indexées.

13 table d'index indirect 1 point sur 256 blocs.

14 table d'indexe indirect 2\_1 et 2\_2 chacune point sur 212 bloc

*Les Fichier dans linux;*

Il faut utiliser les biblio

#include<sys/stat.h>

#include <fcntl.h>

Pour ouvrir un fichier, on crée une variable de type entier par exemple : int fd;

fd = open(Nom\_fichiet,mode\_ouverture,droit);

mode\_ouvert:

O\_RDONLY: pour la lecture seule;

O\_WRONLY: pour l'écriture seul;

O\_RDWR: pour la lecture et écriture;\\

O\_APPEND: pour l'écriture a la fin de ficher;

O\_CREATE, pour crée un fichier, il faut spécifier les droit d’accès pour ce mode.

*Les droits d'accès:*

S\_IRUSR ,S\_IWUSR, S\_IXUSR ,rwx pour user; tous dans une seul S\_IRWXU

S\_IRGRP ,S\_IWGRP, S\_IXGRP ,rwx pour group; tous dans une seul S\_IRWXG

S\_IROTH ,S\_IWOTH, S\_IXOTH ,rwx pour others; tous dans une seul S\_IRWXO

Pour ajouter multiple droit á un fichier c'est mieux d'utiliser une variable mode\_t mode, en séparant les droits par in | ;

mode = S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP;equivalent a 660;

Les droits d'accès sont á la bibliothèque include <sys/stat.h>

Pour fermer un descripteur de fichier on utilise close(fd);

Pour écrire dans un descripteur de ficher, on utilise la fct write(desc\_fich,pointeur\_buff,la taille de buf);

Exemple buf peut être une string, sa taille strleng(buf) write(fd,chaine,strleng(chaine));

buf peut être un pointeur de type struct read(f,&struct,sizeof(struct));

Pour lire d’un descripteur de fichier, on utilise la fct read(desc\_fich,pointeur\_buf,taille\_de\_buff\_onOctet);

Fct read and write renvoie le nombre d'octets lu ou écrit;

Pour se déplacer dans un fichier on utilise la fct lseek(des\_fichier, nombre\_d'octet\_aDepasser, point\_de\_depart);

SEEK\_SET->DEBUT DE FICHIER;

SEEK\_CUR->LE POINT OU ON ETAIT ARRETER;

SEEK\_END->POINT SUR DERNIER ELEMENT DE FICHIER;

Pour l’accès aux informations d'un fichier dans un programme, on utilise la fonction une structure prédéfinie: struct stat buf;

stat(emplacemnt\_fichier,(struct stat)&buf), fct stat retourne -1 en cas d'absence de fichier, sinon il enregistres les informations du fichier passer en paramètres dans la variable buf.

buf.st\_mode: le type de ficher, et ses droits;

printf("%c":,buf.st\_mode & S\_IRUSR ? 'r':'-'); Il va afficher r si user au droit de lecture;

buf.st\_ino: nombre de l'inode; %ld

buf.st\_size: la taille de ficher; %d

buf.st\_uid : le nombre de utilateur auquel appartient ce fichier; %d

buf-st\_gid: le nomber de group auquel ce ficheier appartient;%d

buf.st\_nlink: nombre de lien physique;ld

S\_ISREG(buf.st\_mode): retourne true s’il s’agit d’un fichier ordinaire;

S\_ISDIR(buf.st\_mode): return true s’il s’agit d’un dossier;

S\_ISFIFO(buf.st\_mode): return true si est pipe.

Pour crée un lien physique on utilise la fct link(nom\_fichier,nom\_nouveauFichier);

Pour supprimer un lien physique on utilise la fct unlink(nom\_fichier);

Pour crée un lien symbolique on utilise fct symlink(nom\_fich,nouv\_fich);

readlink(nom\_fich,chaine,taille(chain)) ;chaine aura le nom de fichier auquel point le nom\_fich;

Pour renommer un fichier on utilise la fct rename(nouveau\_nom,ancien\_nom);

Pour modifier les droit on utilise la fct chmod(nom\_fichier,mode)

Pour le mode, soit on donne le code octal (0644) ou par exemple S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IROTH;

Pour manipuler les dossiers dans linux on utilise la biblio include <dirent.h>

mkdir pour crée un dossier; mkdir(nom\_dossier, mode\_t mode);

rmdir pour supprimer un dossier ;rmdir(nom\_dossier);

chdir pour changer le répertoire courant de travail; chdir(nom\_dossier);

getcwd pour savoir le dossier courant de travail;getcwd(chaine\_buf,taill\_de\_chaine);

Pour ouvrir un répertoire, on crée une variable de type DIR\* doss;

opendir pour ouvrir un répertoire: doss = opendir(nom\_doss);

readdir renvoi un pointeur de type struct dirent;struct dirent\* readdir(doss);

Pour avoir les infos de cette structure on crée un pointeur de type struct dirent

struct dirent\* entre;entre = readdir(doss);

entre->d\_name = nom de premier fichier dans le dossier;

entre->d\_ino = nombre de l'inode de premier fichier de dossier;

Si on appelle la fct une deuxième fois, on passe au second ficher;

Si on atteint la fin des fichiers on tombe sur NULL;

**Mémoire**

2 type d’adresse, adresse logique, adresse physique

Les programmes connaissent que les adresses logiques, ne voient pas les adresses physiques.

Chaque adresse physique est représentée dans le system par une adresse logique;

*Pagination*

Chaque programme est découpé en pages, chaque page a un emplacement physique ou case mémoire;

Système a une table des cases qui indique qu’une case est libre (-1) ou pas (p1, ou p2).

La MMU converti une adresse logique en une adresse physique.

Pour savoir à quelle case se trouve une certaine page, chaque processus a une table de page, qui fait la correspondance (numéro de page) @logique->@physique

*Mémoire virtuelle*

La mémoire virtuelle est une fonctionnalité qui permet d’augmenter artificiellement la mémoire vive d’un ordinateur, en transférant temporairement des données vers le disque dur de l’ordinateur, Grâce à cette astuce, l’ordinateur devient capable d’exécuter des programmes dont la taille excède celle de la mémoire vive vraiment disponible sur la machine

Le principe de la mémoire virtuelle consiste à charger en mémoire centrale que la partie couramment utilisé de l’espace d’adressage des processus, pour cela chaque entrée de la table des pages comporte un bit de validation V, 1 si la page est en mémoire centrale, 0 sinon.

Default de page: se produit lors de chercher une page dont son bit de validation vaut 0, la conversion ne se fait pas(@logique->@physique: par MMU);

Pour remplacer un page on a 2 algorithmes:

FIFO: premier page charge, premier page à éliminer;

LRU: laisser les pages récemment utilisées;

**Signaux**

Pour manipuler les signaux on utilise la biblio #include <signal.h>

kill pour envoyer un signal á un processus kill (pid,signal);

pid = 0;envoie le signal au groupe de processus;

pie = -1;envoie le signal a tous les processus;

Quelque signale :

SIGCHLD : envoyé par le processus fils vers le processus pere indiquant qu’il a terminé son travaille.

SIGUSR1 SIGUSR2 : ils ne sont pas déclenchés par une action particulière, vous pouvez explicitement les envoyer par programmation.

SIGINT : c’est le signal envoyer vers un processus lorsqu’on tape au clavier sur les touche Ctrl + C.

SIGALARM : est le signal envoyé par la fonction alarm(3) lorsque le temps passé en paramètres est écoulé.

SIGTERM : c’est le signal envoyé à un processus afin de provoquer sa fin,SIGTERM a plus pour but de demander gentiment de s'arrêter, permettant un nettoyage et la fermeture de fichiers.

SIGKILL et SIGSTOP: ce sont 2 signaux qu’on peut pas les bloqués.

sigset\_t ens: variable d'ensemble des signaux;

sigfillset(&ens) remplir l'ensemble avec tous les signaux de Sys;

sigaddset(&ens,SIGINT) ajoute le signal sigint á l'ensemble ens;

sigdelset(&ens,SIGINT) supprimer le sigint de l'ensembele ens;

sigemptyset(&ens) vider l'ensemble de tous les signaux;

sigismember(&ens,SIGINT) permet de savoir est-ce-que sigint existe dans l'ensemble;

sigpromask(SIG\_SETMASK,&ens,NULL) :avec l'option setmask permet de blocker les signaux existent dans ens pour un processus;

sigpending(&ens) permet d'enregister les signgaux blocker par le processus ,lors de leur arrivage, sinon elle reste vide.

sigsupend(&ens) permet de blocker just les signaux de ens, temporellement jusqu'à l'arrivée d'un signal non bloquée ou bien qui n'existe pas dans ens,apres il restaure l'ancien mask des signaux bloque.

signal(SIGINT,handler); permet d'exécuter les instructions de la fonction Handler lors de réception d’un signal passé comme paramètres.

sigint pour cet example ,on ingnorant le fonctionememt de sigint.il faut la placer au debut de code.

sleep(5) permet de mettre le processus en état bloqué pendant 5 seconds

pause() permet de bloquer le processus en attente de réception de signal qui est spécifier dans la fonction signal avec un fonction pour continuer.

alarm(5) permet de envoyer un signal SIGALARM au processus appelant dans 5 seconds;

sigaction(SIGSEGV, &action, 0) meme role que signal() permet d'exécuter un fonction á l'arrivage d'un signal sigsegv ,seulement ici en utilise une structure sigaction

struct sigaction action;

action.sa\_handler = handler ;handler est la fonctions a executer lors de recevoir du signal;

action.sa\_flags = 0;pas de commantaire;

sigemptyset(&action.sa\_mask) vider l'ensemble de sturcture;

action.sa\_mask :ens des signaux seront blocke lors d'execution de fonction handler

appelle á la fonction avec :sigaction(SIGSEGV, &action, 0)

**Les tubes**

Pour envoyer un message d'un processus á un autre en utilise les tubes

*Tube anonyme pere\_fils*

Pour crée un tube anonyme en déclare un tableau de taille égal à 2;

int fd[2];en le mettre dans la fct : pipe(fd);

mnt fd[1] sera utilisé pour écrire un message ,et fd[0] pour lire ce message;

Dans le processus écrivant on ferme le tube de lecture avec close(fd[0]), et on écrit avec write(fd[1],message,taille(message))

Dans le processus lisant on ferme le tube d'écriture avec close(fd[1]), et on lit le message avec read(fd[0],message,taille(message)); /\*message c'est une chaine de caratcter\*/

On peut lier le tube d'écriture avec STDOUT avec la commande dup2(fd[1],STDOUT\_FILENO);close(fd[1]); et on peut lier le tube de lecture avec stdin avec la commande dup2(fd[0],STDIN\_FILENO);close(fd[0]);

*Tube nommés processus indépendants*

On crée un tube avec la fonction mkfifo(nom\_fichie,mode) /\*mode contient les droit d'access\*/

Processus écrivant: pour écrire dans le tube on utilise la commande

fd = open(nom\_pipe,O\_WRONLY) et après en utilise la fonction write;

Processus lisant: on ouvre le tube on mode lecture seul;

fd = open(nom\_pipe,O\_RDONLY); et après on utilise la foction read; il faut exécuter les 2 programme en même temps;