Système d'exploitation

Thibault Bernard

thibault.bernard@univ-reims.fr

Organisation

- Cours (3h)
  - Historique
  - Rôle d'un Système d'exploitation
  - Partage de ressources
    - Ordonnancement de processus
    - Gestion Mémoire
    - Notion de Concurrence
    - Systèmes de Fichiers
  - Notions de Virtualisation
  - Commandes Unix
  - Notion de Script

- Travaux Pratiques (3h)
  - Découverte de l'environnement virtualisé
  - Tour d'horizon des commandes de bases
  - Notions d'administration système
  - Scripts

Historique

- **Machine de Charles Babbage :** non fonctionnel, mécanique pas assez précise (1792-1871)

- **1945-55 Première Génération :** Tubes à vide et tableaux d'interrupteurs.
  - Programme codé dans un langage absolu : par des tableaux d'interrupteurs
  - Années 50 : introduction de la carte perforée

- **1955-65 Deuxième Génération :** Transistors et systèmes par lots (Mainframes ou ordinateurs centraux)
  - Accroissement de la fiabilité
  - Soumission en 4 étapes (Ecriture du programme, soumission à l'opérateur, introduction des cartes perforées, compilation fortran)
  - Perte de temps à cause des manipulations humaines
  - Batch ou traitement par lots (FMS)

Historique

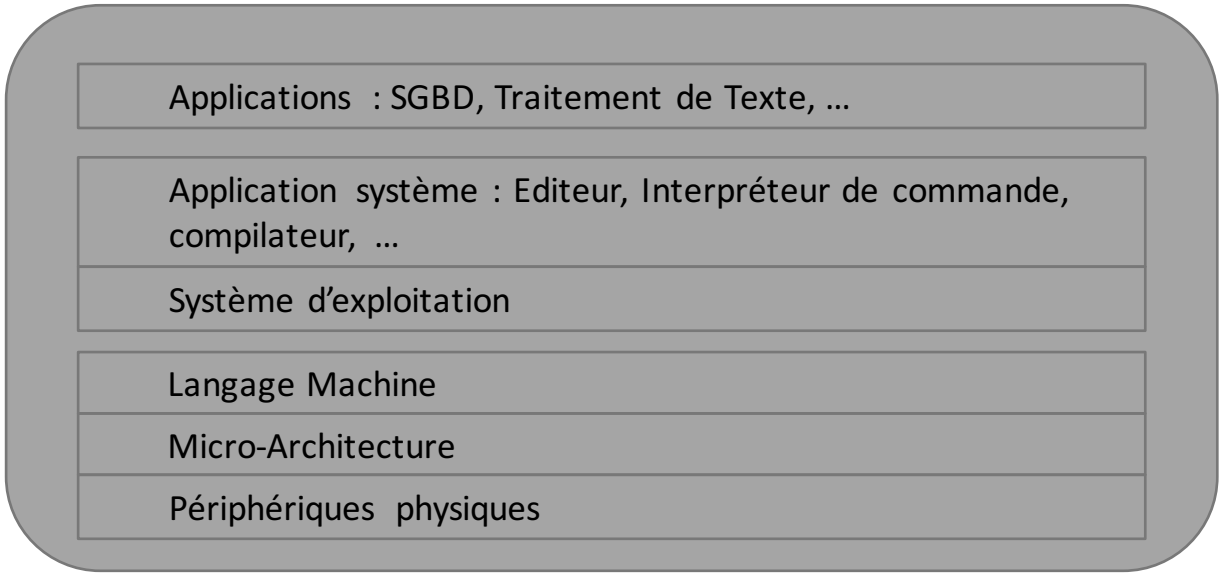
- **1965-80 Troisième génération :** 2 architectures incompatibles (Unité de données : Mots VS Caractères)
  - IBM introduit son system/360 et fournit une compatibilité au niveau logiciel (OS/360)
  - Multiprogrammation (Occupation CPU proche de 100%)
  - Spoolage
  - Temps partagé
  - Début d'Unix (System V et BSD)

- **1980-2010 Quatrième Génération :** Ordinateurs personnels
  - Développement des Circuits intégrés à haute densité
  - Intel 8080
  - MSDOS (Bill Gates Tim Paterson)
  - Lisa, Macintosh (Steve Jobs introduit les IHM)
  - Windows (d'abord comme surcouche à MSDOS, puis comme Système d'exploitation à part entière)
  - Unix
  - Développement des réseaux informatiques
  - Système d'exploitation distribué

- Et Maintenant ?
  - Cloud computing
  - Virtualisation
  - …

Rôle d'un système d'exploitation

- Gestion de périphériques
- Interface Programmeur/Matériel



Rôle d'un système d'exploitation

- Gestion des ressources (périphériques, mémoire, processeurs)
- Le système est une extension de la machine
- Exemple : Ecriture sur disque
  - Vision haut niveau : Système de fichiers, bibliothèques système

```
int main(int argc, char *argv[]){
        char c;
        int f = open(argv[1],O_WRONLY);
        while(read(f,&c,sizeofchar)!=0){
                printf(« %c »,c);
        }
        close(f)
        exit(0);
}
```

  - Vision bas niveau :
    - Détecter la surface d'enregistrement
    - Positionner la tête de lecture
    - Déterminer la vitesse du moteur
    - ...

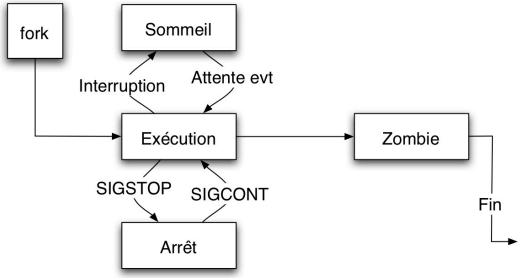
Partage de ressources

- Dépendamment de la nature de la ressource, on peut partager sur deux dimensions :
  - Le temps
  - L'espace

  Exemple : Imprimante et Disque …
  - Une imprimante ne peut pas être partagée par plusieurs processus en même temps
  - Un disque ne peut pas être utilisé exclusivement par un seul processus

Gestion de Processus

- Création de processus :
  - Premier processus : init
  - Création de processus par d'autres processus
  - Arborescence de processus
- Gestion des processus
  - Visualisation (commande ps; top; pstree)
  - Suppression de processus (commande kill)
- Interruption



- Allocation temps CPU pour les processus

Ordonnancement

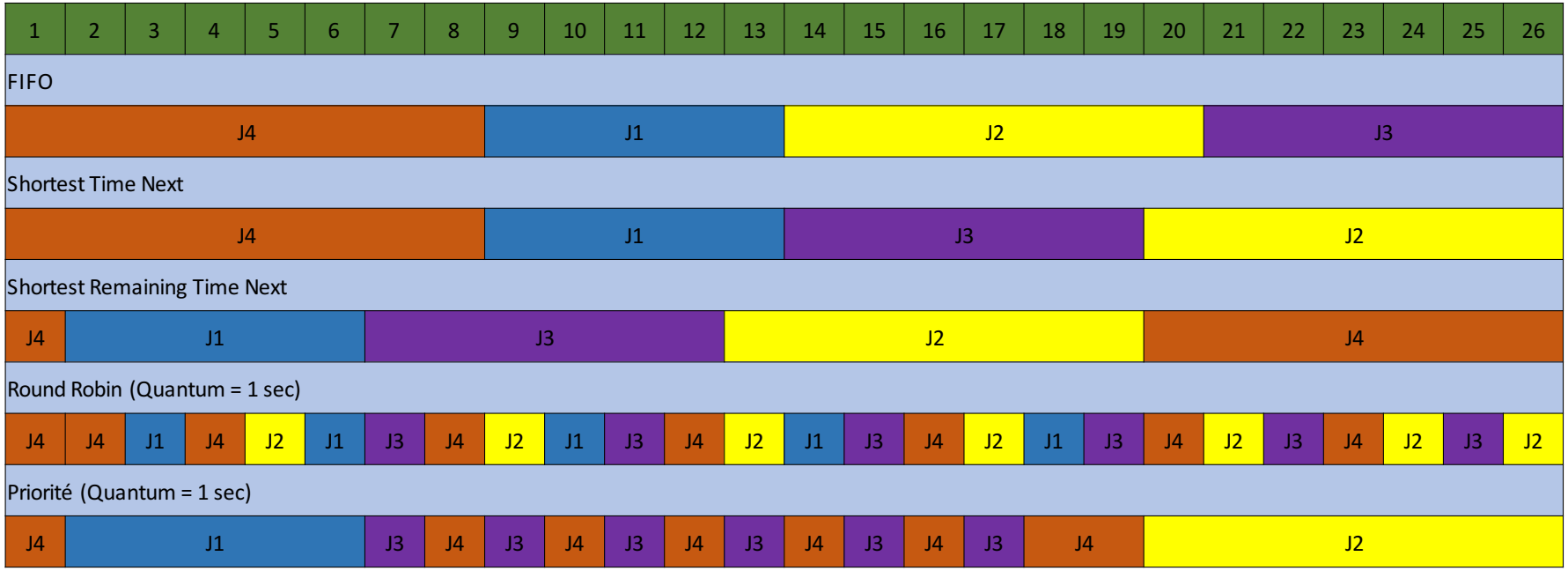
- L'ordonnanceur est un algorithme qui élit un processus. Ce processus a le privilège d'accéder au CPU.
  - Non préemptif : sélectionne un processus qui s'exécute jusqu'a ce qu'il bloque ou qu'il libère volontairement le processeur.
  - Préemptif : sélectionne un processus puis il s'exécute pendant un délai. Si le processus est toujours en cours d'exécution après ce délai alors il est suspendu et un autre processus est choisi.
- Algortihmes d'ordonnancement :
  - FIFO (non préemptif)
  - Shortest Job First (non préemptif)
  - Shortest Remaining Time Next (préemptif)
  - Round Robin
  - Priorité.

Exemple d'ordonnancement

Soit le scénario suivant :

Processus	Durée	Date de soumission	Priorité
J1	5 sec	H=1	5
J2	7 sec	H=2	2
J3	6 sec	H=3	4
J4	8 sec	H=0	4

Exemple d'ordonnancement

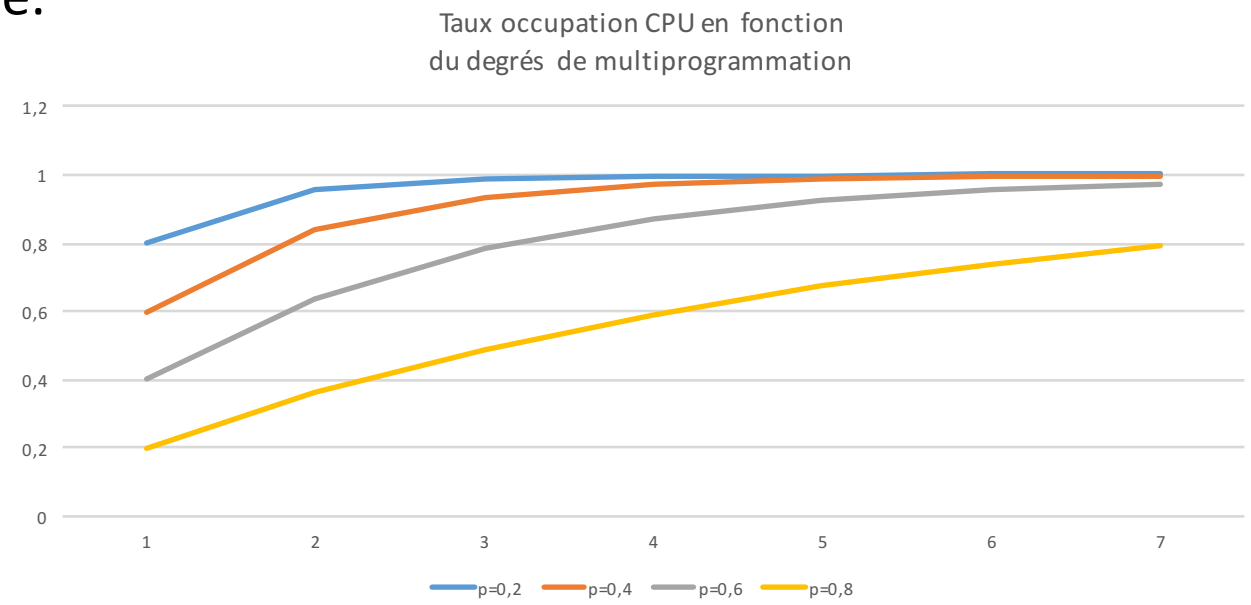


D'autres ordonnancements existent :

- Ordonnancement équitable
- Ordonnancement garanti
- Ordonnancement spécifique (temps réel, machine parallèle, …)

Gestion Mémoire

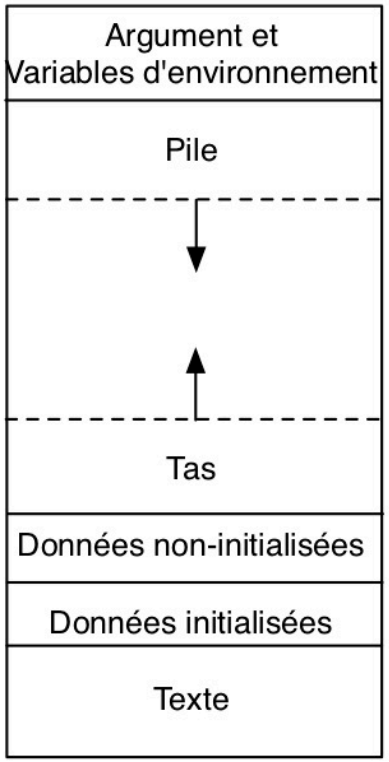
- Multiprogrammation => nécessité d'héberger plusieurs processus en mémoire.



Gestion Mémoire

- Taille mémoire d'un processus
- La pile sert à l'appel de fonction
- Le tas sert à l'allocation dynamique
  ⇒Nécessité de pouvoir augmenter la taille d'un processus en cours d'exécution
- La commande *size* permet d'obtenir la taille des différents segments
- 2 approches complémentaires :
  - Le va et vient
  - La mémoire virtuelle

Schéma mémoire d'un processus

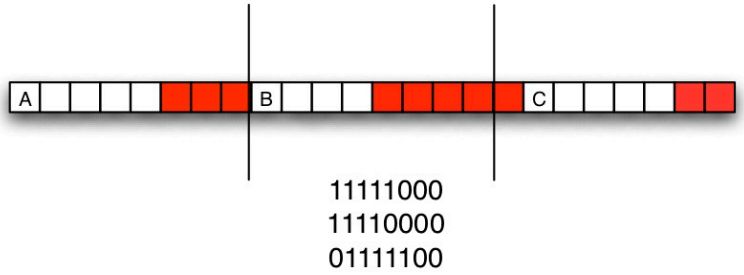


Le va et vient

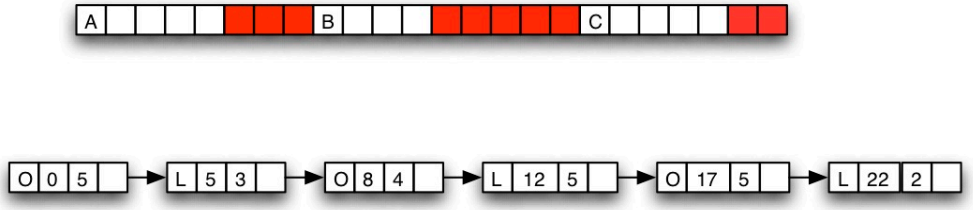
- On considère chaque processus dans son intégralité
- Le processus est exécuté puis éventuellement sauvegardé sur le disque
- Sauvegarde complète sur le disque
  - Au chargement : nécessite de relocaliser
  - L'allocation dynamique complique le chargement et la libération
- Présence de trous en mémoire : technique du compactage de mémoire (coûteux)
- Quand la taille des processus est fixe, le S.E. alloue exactement la taille nécessaire
- Quand un programme a besoin d'augmenter sa taille :
  - Soit il y a de la place juste a cote : aucun problème
  - Soit il est contigu a un autre processus : déplacer tout le processus
  - Sinon, s'il n'y a plus de place en mémoire : attente ou terminaison

Le Va et Vient (2 représentations)

Tableau de bits



Liste chainée



- La mémoire est répartie en unités d'allocation (ua)
- Chaque unité d'allocation possède une taille comprise entre quelques octets à plusieurs kilo-octets.
- Problème : trouver la bonne taille pour les unités d'allocation

Le Va et Vient

- Où placer les processus ?
  - Dans quel espace libre ?
  - Peut on optimiser la taille des espaces libres ?
  - Peut on optimiser la recherche d'un espace libre ?

- Différentes stratégies :
  - First fit
  - Next fit
  - Best Fit
  - Worst Fit

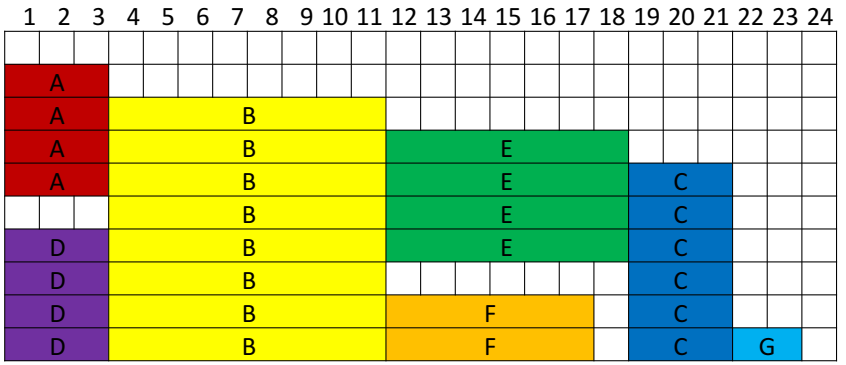
Le Va et Vient

Soit le scénario :

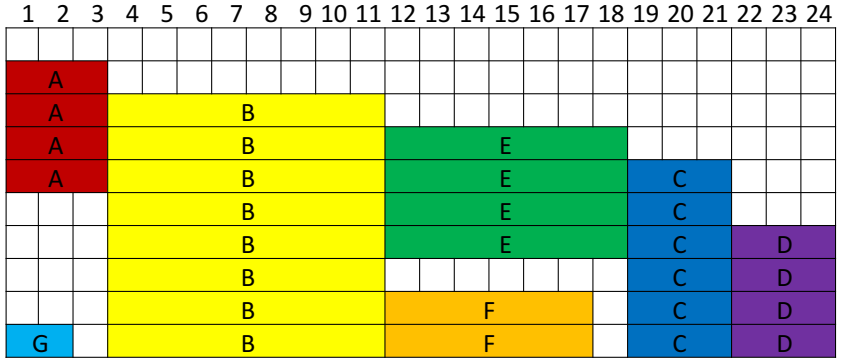
1. Chargement A, 3ua
2. Chargement B, 8ua
3. Chargement E, 7ua
4. Chargement C, 3ua
5. Déchargement A
6. Chargement D, 3ua
7. Déchargement E
8. Chargement F, 6ua
9. Chargement G, 2ua

En considérant une zone mémoire de 24 ua

First Fit



Next Fit



Autres stratégies, même constat
=> Aucune solution idéale !

Même problématique sur la gestion des espaces libres sur disque

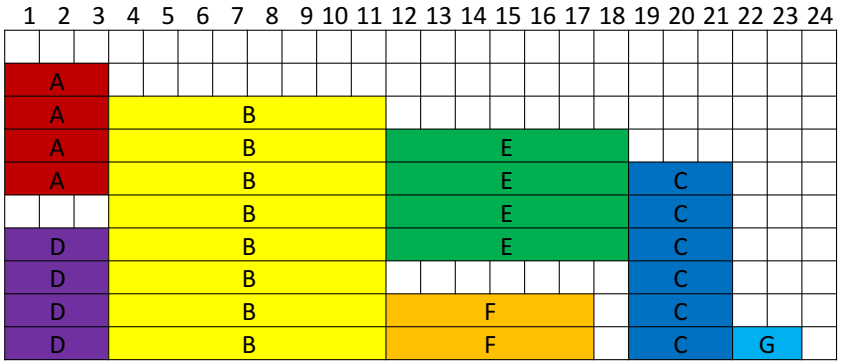
Le Va et Vient

Soit le scénario :

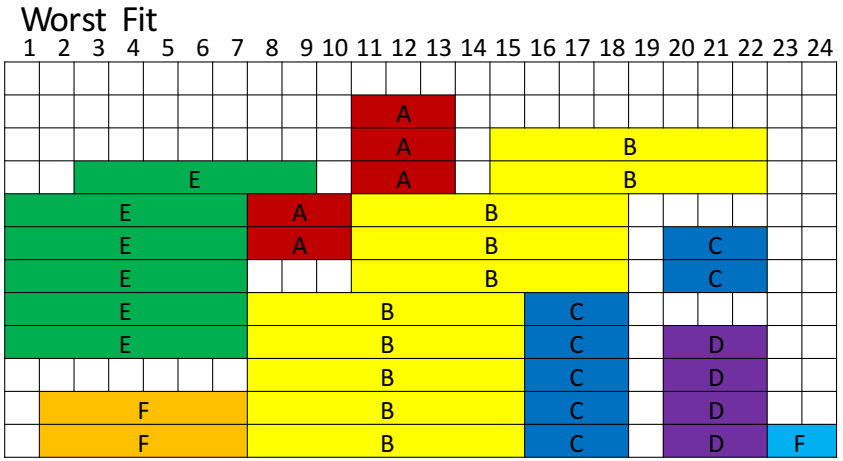
1. Chargement A, 3ua
2. Chargement B, 8ua
3. Chargement E, 7ua
4. Chargement C, 3ua
5. Déchargement A
6. Chargement D, 3ua
7. Déchargement E
8. Chargement F, 6ua
9. Chargement G, 2ua

En considérant une zone mémoire de 24 ua

Best Fit



Worst Fit

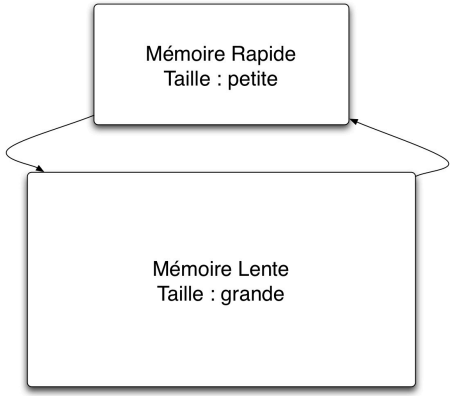


Autres stratégies, même constat
=> Aucune solution idéale !

Même problématique sur la gestion des espaces libres sur disque

La Mémoire Virtuelle

- La taille d'un processus peut être supérieur à la capacité mémoire physique.
- Les processus ne sont plus considérés dans leur intégralité : l'espace mémoire du processus est découpé en plusieurs morceaux (les pages)
- Un processus peut s'exécuter s'il est partiellement en mémoire.
- Technique : La Pagination



La Pagination

- Le défaut de page se produit lorsque le système a besoin d'une page non présente en mémoire. Dans ce cas :
  - Le système d'exploitation choisit une page en mémoire rapide, l'écrit sur la mémoire lente
  - Et charge la page qui faisait défaut.

- Choix de la page à évincer :
  - Algorithme de Belady
  - Algorithme First In First Out
  - Algorithme Last In First Out
  - Algorithme Least Recently Used
  - Algorithme Flush When Full

La Pagination

Soit la séquence S=A,B,C,B,D,A,E,C,D,E,F,C,F,G,D,A,E,B,C,F,G,E,D,F et une zone de mémoire rapide pouvant contenir 4 pages.

Belady

A	B	C	B	D	A	E	C	D	E	F	C	F	G	D	A	E	B	C	F	G	E	D	F
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	E	E	E	E	E	E	D	D
	B	B	B	B	B	E	E	E	E	F	F	F	G	G	G	G	G	G	G	G	G	G	G
		C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	F	F	F	F	F
				D	D	D	D	D	D	D	D	D	D	D	D	D	B	B	B	B	B	B	B

FIFO

A	B	C	B	D	A	E	C	D	E	F	C	F	G	D	A	E	B	C	F	G	E	D	F
A	A	A	A	A	A	E	E	E	E	E	E	E	E	E	E	E	B	B	B	B	B	D	D
	B	B	B	B	B	B	B	B	B	F	F	F	F	F	F	F	F	C	C	C	C	C	C
		C	C	C	C	C	C	C	C	C	C	C	G	G	G	G	G	G	F	F	F	F	F
				D	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A	E	E	E

La Pagination

LIFO

A	B	C	B	D	A	E	C	D	E	F	C	F	G	D	A	E	B	C	F	G	E	D	F
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
				D	D	E	E	D	E	F	F	F	G	D	D	E	E	E	F	G	E	D	F

LRU

A	B	C	B	D	A	E	C	D	E	F	C	F	G	D	A	E	B	C	F	G	E	D	F
A	A	A	A	A	A	A	A	A	A	F	F	F	F	F	F	E	E	E	E	G	G	G	G
	B	B	B	B	B	B	C	C	C	C	C	C	C	C	A	A	A	A	F	F	F	F	F
		C	C	C	C	E	E	E	E	E	E	E	E	D	D	D	D	C	C	C	C	D	D
				D	D	D	D	D	D	D	D	D	G	G	G	G	B	B	B	B	E	E	E

FWF

A	B	C	B	D	A	E	C	D	E	F	C	F	G	D	A	E	B	C	F	G	E	D	F
A	A	A	A	A	A	E	E	E	E	E	E	E	G	G	G	G	B	B	B	B	E	E	E
	B	B	B	B	B		C	C	C	C	C	C		D	D	D		C	C	C		D	D
		C	C	C	C			D	D	D	D	D			A	A			F	F			F
				D	D					F	F	F				E				G			

La Concurrence

Le problème de concurrence survient lorsque plusieurs processus souhaitent accéder à la même ressource.

Exemple : Soit une variable partagée A, et deux processus exécutant la portion de code

```
x ⇐ lire(A)
x⇐ x+1
Ecrire(A,x)
```

Que vaut A après l'exécution des deux processus ?

La Concurrence

La portion de code qui accède à A doit être réalisée en excluant tout autre portion de code qui accède à A.

Problème d'exclusion mutuelle :

- Sureté : A un instant donné, un seul processus exécute sa portion de code dite *de section critique*
- Vivacité : Un processus demandant l'exécution de son code en *section critique* doit finir par l'obtenir

Solutions au problème de Concurrence

- Désactivation des interruptions $\Rightarrow$ Inenvisageable !
- Alternance stricte : Fastidieux !
- Algorithme de Peterson

```
while vrai do
        Actions avant la S .C.
        D_i ⇐ vrai
        Tour ⇐ i
        while (D_((i+1)%2)) \ (Tour = i) do
                Rien
        end while
        Actions de la S.C.
        D_i ⇐ faux
Actions suivant la S.C.
end while
```

- Les Sémaphores

Les sémaphores

- Reprend l'idée de verrou :
  - Un sémaphore est un entier
  - 2 opérations P() et V()
  - P() effectue un test et une décrémentation réalisés de manière **atomique**
  - V() effectue une incrémentation
  - Si le sémaphore S a une valeur nulle, l'opération P() bloque jusqu'à ce que la valeur de S devienne strictement positive.

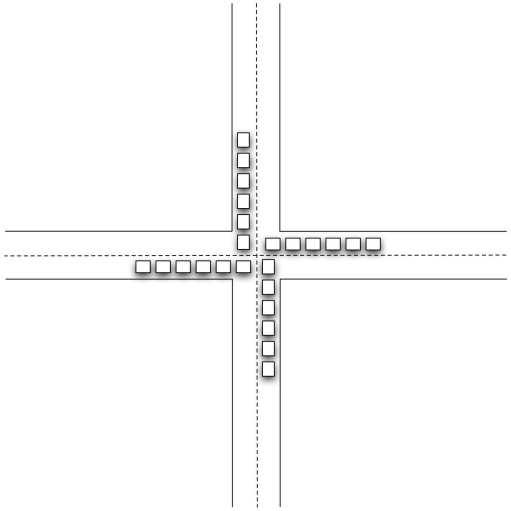
Le problème d'interblocage

Survient lorsque plusieurs processus souhaitent accéder à plusieurs ressources et que les demandes d'accès ne se font pas dans le même ordre pour chacun des processus.

Exemple :

Soit 2 processus P1 et P2, 2 sémaphores S=1 et S'=1 et l'exécution suivante

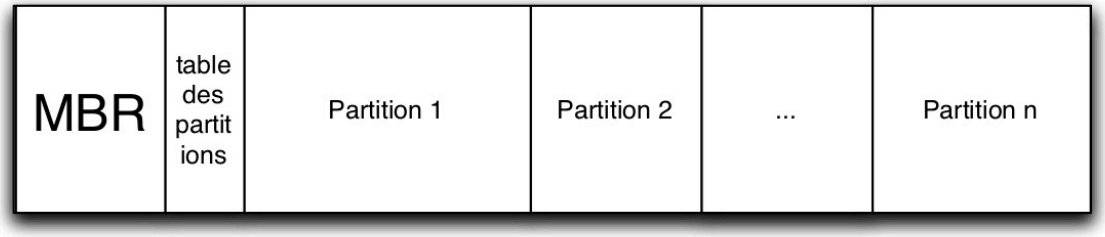
P1: P(S)
P2: P(S')
P1: P(S')
P2: P(S)
P1 Section critique
P2 Section critique
P1: V(S)
P1: V(S')
P2: V(S)
P2: V(S')



Exemple courant :
La priorité à droite !

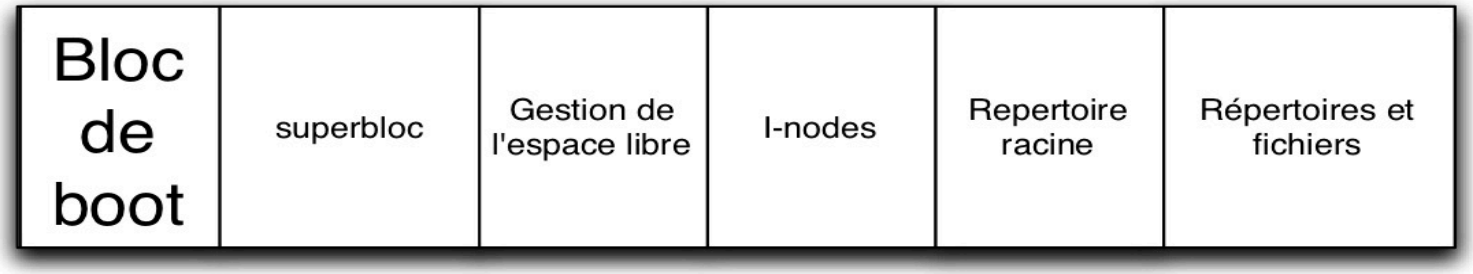
Système de fichiers

Organisation d’un Disque



Partition (exemple Unix)

Partition type Unix



Bloc de Boot : Contient les instructions nécessaires à l'initialisation du système.

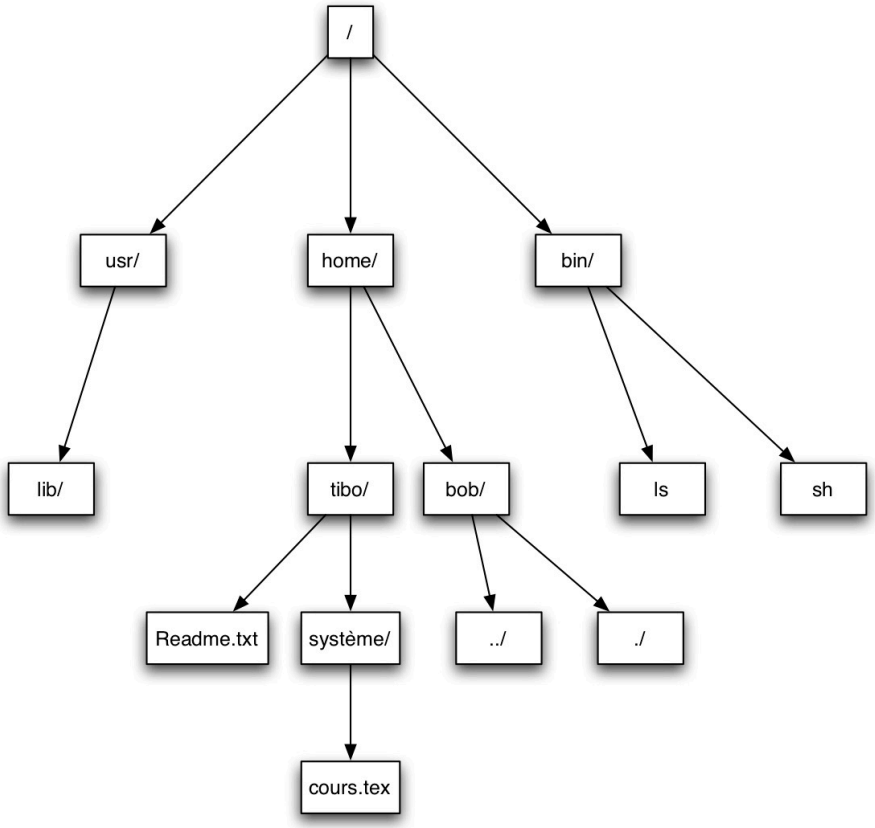
Superbloc : contient la structure du système de fichier (nb de blocs, nb d'i-nodes).

Gestion de l'espace libre : contient le début de la liste des blocs libres.

I-nodes : contient des informations comptables permettant de retrouver le contenu des fichiers.

Répertoires / Fichiers : contenu des fichiers et répertoires, grâce a un mécanisme de référencement astucieux la taille d'un fichier est théoriquement infinie.

Arborescence Unix



Mécanismes système Unix

« Small is beautiful »

- Manipulation de répertoires
  - mkdir() : crée un répertoire
  - rmdir() : supprime un répertoire
  - chdir() : change le répertoire de travail
  - opendir() : ouvre un répertoire
  - closedir() : ferme un répertoire
  - readdir() : lit une entrée sur un répertoire
  - rewinddir() : repositionne la lecture du répertoire au début

- Manipulation de fichiers
  - creat() : crée un fichier
  - open() : ouvre un fichier
  - close() : ferme un fichier
  - read() : lit des octets sur un fichier
  - write() : écrit des octets sur un fichier
  - lseek() : positionne le pointeur du fichier
  - stat() : récupère le statut du fichier
  - fcntl() : verrouille le fichier

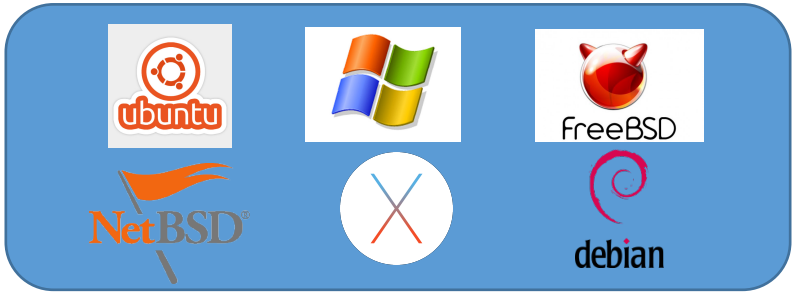
Quelques types de partition

- EXT2 : partition type linux, minimise l'espace physique entre blocs utilisés.
- EXT3 : partition type linux, inclut une journalisation.
- NTFS : partition type Windows, successeur des partitions FAT 32, améliore les performances et la fiabilité et offre un mécanisme de journalisation.
- APFS : partition type OSX, offre un mécanisme de journalisation, et une possibilité de chiffrement.

Virtualisation

- **Virtualisation système :** exécution sur une machine hôte, dans un environnement isolé, des systèmes d'exploitation.
- **Virtualisation applicative :** exécution sur une machine hôte, dans un environnement isolé, d'applications.
- Exécution d'un système au sein d'un autre système… Paradoxe ?
- Permet une modularité à souhait et donc une répartition de charge.

⇒

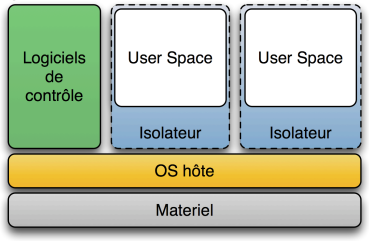


Intérêts

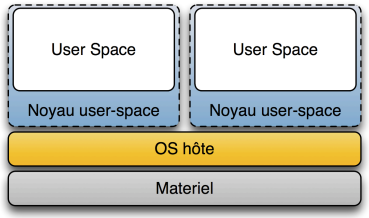
- Utilisation optimale des ressources
- Installation, déploiement et migration facile des machines virtuelles d'une machine physique à une autre
- Economie sur le matériel par mutualisation
- Tests
- Isolation
- Allocation dynamique

Types de Virtualisation

- **Isolateur :** Isolation du contexte d'exécution d'un processus (pas vraiment de la virtualisation)

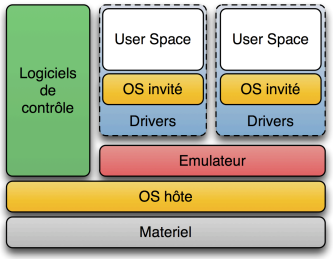


- **Noyau en espace utilisateur :** Un noyau tourne comme une application en espace utilisateur de l'OS hôte. Peu performant : empilement !

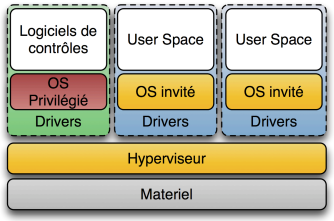


Types de Virtualisation

- **Hyperviseur de type 2 :** permet de lancer des OS invités. La machine virtualise ou/et émule le matériel pour les OS invités. Isolation complète !



- **Hyperviseur de type 1 :** est un noyau système léger qui gére les accès des noyaux d'OS invités vers le matériel. Le plus performant !



Quelques solutions Logicielles

- Isolation de ressources : Docker, BSD Jail,…
- Hyperviseur Type 2 : logiciels Vmware, VirtualBox, …
- Hyperviseur Type 1 : VMware vSphere

Travaux pratiques

Les TP auront lieu sur un environnement virtualisé (Virtual Box / AWS).

- Apprentissage des commandes de bases :
  - Manipulation de répertoires et de fichiers
  - Manipulation des processus
- Notions d'administration Système :
  - Gestion utilisateurs
  - Installer des applications (paquetages / sources)
- Introduction aux scripts

Environnement de travail

- Durant les TP, vous serez sur un système virtualisé Linux / Ubuntu avec les droits administrateurs.

- L'interaction entre l'utilisateur et le système s'effectue principalement à l'aide d'une console ou d'un terminal.

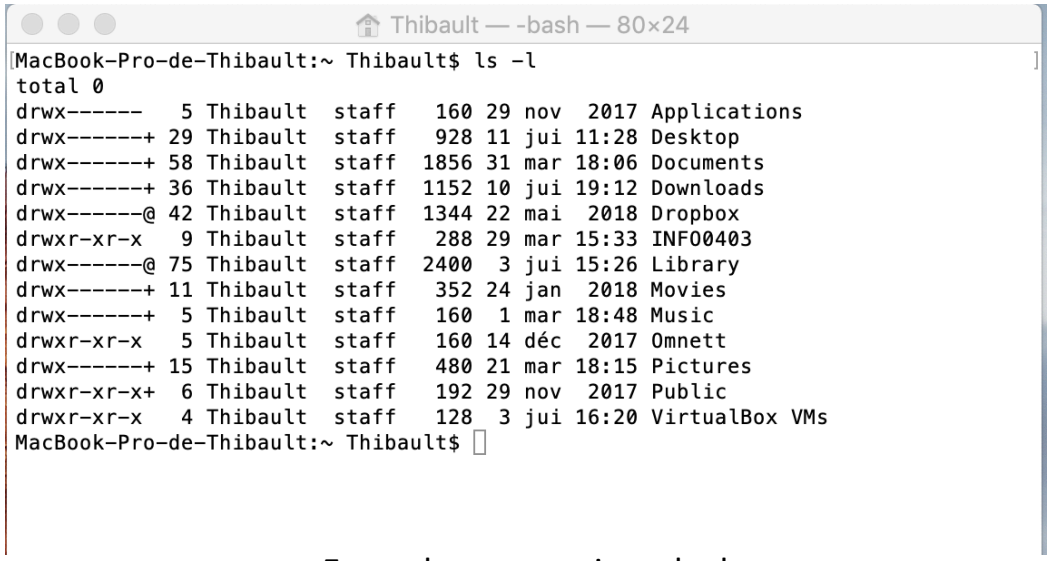
- Lorsque vous serez connecté, vous serez dans votre répertoire courant de travail.

Quelques commandes (basiques)

ls : liste les fichiers du répertoire courant
cd : permet de vous déplacer dans l'arborescence des répertoires
pwd : affiche le répertoire courant
touch : crée un fichier
cp : copie un fichier
rm : supprime un fichier
mv : renomme un fichier
mkdir : crée un répertoire
rmdir : supprime un répertoire
ln : crée un lien vers un fichier
cat : affiche le contenu d'un fichier

Options et paramètres

- La plupart des commandes nécessite des paramètres et peuvent utiliser des options



Exemple avec option : ls -l

Options et paramètres

- La plupart des commandes nécessite des paramètres et peuvent utiliser des options

```
MacBook-Pro-de-Thibault:~ Thibault$ touch toto.txt
MacBook-Pro-de-Thibault:~ Thibault$ ls
Applications    Downloads       Library         Omnett          VirtualBox VMs
Desktop         Dropbox         Movies          Pictures        toto.txt
Documents       INFO0403        Music           Public
MacBook-Pro-de-Thibault:~ Thibault$ mv toto.txt titi.txt
MacBook-Pro-de-Thibault:~ Thibault$ ls
Applications    Downloads       Library         Omnett          VirtualBox VMs
Desktop         Dropbox         Movies          Pictures        titi.txt
Documents       INFO0403        Music           Public
MacBook-Pro-de-Thibault:~ Thibault$
```

Exemple avec paramètres : touch toto.txt
mv toto.txt titi.txt

Petites choses pratiques à savoir

- Auto-completion : « TAB » permet « d'auto-completer » votre commande. Si cela ne suffit pas, appuyez 2 fois sur « TAB », l'interpréteur de commande vous affichera les différentes possibilités.

- Historique : les touches « ↑ » et « ↓ » vous permettent de vous déplacez dans votre historique.

Commande de manuel

Vous ne pourrez pas connaître toutes les commandes, il faut :

- Beaucoup de pratique !

- S'aider à l'aide des ressources à votre disposition :
  - Ressources internet
  - Aide accessible via l'interpréteur de commande

Aide

- Page de Manuel : accessible avec la commande man

  Pour en savoir plus : man man …

Utilisateurs et groupes

- Une personne connectée sur une machine est un utilisateur :
  - Elle a un quota d'espace disque
  - Elle a un identifiant
  - Elle est propriétaire d'un certain nombre de fichiers
  - Elle est affectée à un ou plusieurs groupes
- L'utilisateur « root » est l'utilisateur qui a les droits d'administration du système.
  - su : change l'utilisateur courant pour l'administrateur
  - sudo : exécute une commande en mode « root » si vous êtes dans le groupe « sudoer ».

Utilisateurs et groupes

- useradd : créer un nouvel utilisateur
- groupadd : créer un groupe d’utilisateur
- userdel : supprime un utilisateur
- groupdel : supprime un groupe
- usermod : modifier un compte utilisateur
- groupmod : modifier un groupe

Droits

Droits d'un fichier

```
drwxr-xr-x   5 Thibault  staff   160 14 déc  2017 Omnett
drwx------+ 15 Thibault  staff   480 21 mar 18:15 Pictures
drwxr-xr-x+  6 Thibault  staff   192 29 nov  2017 Public
drwxr-xr-x   4 Thibault  staff   128 12 jui 17:13 Scripts
drwxr-xr-x   4 Thibault  staff   128 12 jui 18:32 VirtualBox VMs
-rw-r--r--   1 Thibault  staff     0 11 jui 13:25 titi.txt
MBP-de-Thibault:~ Thibault$
```

Ils sont donnés par une séquence (-rw-r--r-- pour le fichier titi.txt)

- Le premier caractère indique si le fichier est spéciale (- pour un fichier normal, d pour un répertoire, l pour un lien symbolique,…)
- Les 3 prochains caractères représentent les droits pour le propriétaire (r droit de lecture, w droit d'écriture, x droit d'exécution, - pour l'absence du droit)
- Les 2 dernières séquences sont les droits pour les utilisateurs du même groupe (rwx), et les droits pour les autres utilisateurs.

- chmod : commande pour changer les droits sur un fichier
- chown : commande pour changer le propriétaire d'un fichier

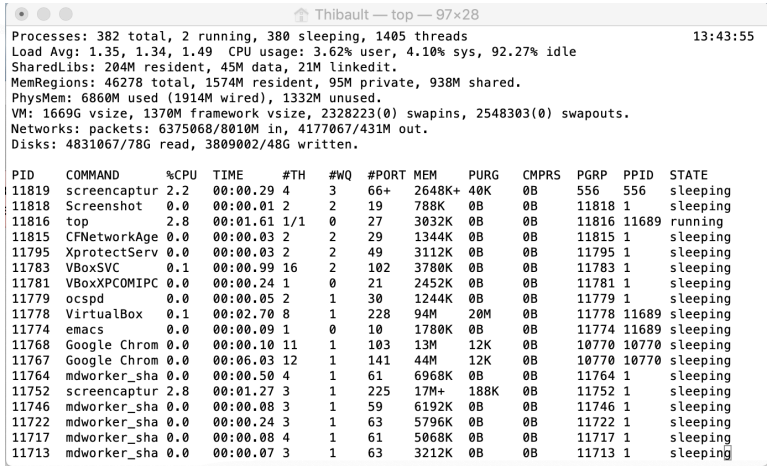
Gestion des processus

Les processus ont tous un identifiant unique.

- La commande ps affiche les processus liés au terminal

```
MacBook-Pro-de-Thibault:~ Thibault$ ps
  PID TTY           TIME CMD
11689 ttys000    0:00.08 -bash
11774 ttys000    0:00.10 emacs
11778 ttys000    0:02.00 /Applications/VirtualBox.app/Contents/MacOS/VirtualBox
MacBook-Pro-de-Thibault:~ Thibault$
```

- La commande top affiche les processus et leur consommation ressource



Gestion des processus

- La commande kill envoie un signal à un processus passé en paramètre

```
MacBook-Pro-de-Thibault:~ Thibault$ ps
  PID TTY           TIME CMD
11689 ttys000    0:00.09 -bash
11774 ttys000    0:00.10 emacs
11778 ttys000    0:02.95 /Applications/VirtualBox.app/Contents/MacOS/VirtualBox
MacBook-Pro-de-Thibault:~ Thibault$ kill -9 11774
[1]+  Killed: 9               emacs
MacBook-Pro-de-Thibault:~ Thibault$ ps
  PID TTY           TIME CMD
11689 ttys000    0:00.09 -bash
11778 ttys000    0:02.98 /Applications/VirtualBox.app/Contents/MacOS/VirtualBox
MacBook-Pro-de-Thibault:~ Thibault$
```

- bg : passe un processus suspendu en arrière plan.
- fg : ramène un processus au premier plan.
- jobs : affiche la liste des processus suspendus
- CTRL ^Z suspend l'exécution du processus courant (envoie un signal SIG_STOP)
- CTRL ^C termine l'exécution du processus (envoie un signal SIG_INT)
- nice : modifie la priorité du processus dans l'ordonnanceur

Redirection des Entrées/Sorties

L'interpréteur de commande permet de rediriger les E/S

- > : redirige la sortie standard du programme appelé vers le fichier donné (>> effectue la même chose mais en concaténation).
- < : redirige l'entrée standard du programme appelé depuis le fichier donné.
- | : réalise un « pipeline » entre deux programmes : la sortie du premier est redirigée vers l'entrée du second.

Ecriture de scripts

Un script shell est un fichier texte. Il est exécutable mais ce n'est pas un fichier binaire : il doit être interprété. De plus le fichier doit avoir les droits d'exécution.

- Commentaire : une ligne commençant par # est ignoré par l'interpréteur de commande.

- Variables
  - Variables système ($PATH, $LOGNAME, $PWD, $HOME…)
  - Variables paramètres ($0,$1,$2,…, $#, $$)
  - Variables utilisateur : elles se définissent par un nom, on y fait référence dans la suite en plaçant « $ » devant.

Ecriture d'un script

- Instructions =, read, echo
  - = affecte une valeur à une variable
  - echo permet d'afficher une chaine de caractères, variables

```
MacBook-Pro-de-Thibault:~ Thibault$ echo $LOGNAME
Thibault
MacBook-Pro-de-Thibault:~ Thibault$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin
MacBook-Pro-de-Thibault:~ Thibault$ echo $$
15165
```

  - read permet d'affecter une saisie au clavier à une variable.

```
MacBook-Pro-de-Thibault:~ Thibault$ read toto
bob
MacBook-Pro-de-Thibault:~ Thibault$ echo $toto
bob
MacBook-Pro-de-Thibault:~ Thibault$
```

Ecrire un script

- Conditionnelle
  - Si

```
#!/bin/bash
if [ $1 -eq 1 ]
then
    echo reçu 1;
elif [ $1 -eq 2 ]; then
    echo reçu 2;
elif test $1 -eq 3 ; then
    echo recu 3;
else
    echo recu autre...;
fi
```

```
MBP-de-Thibault:Scripts Thibault$ ./scif.sh 3
recu 3
MBP-de-Thibault:Scripts Thibault$
```

Ecrire un script

- Conditionnelle
  - Case

```
#!/bin/bash
case  $1 in
1 ) echo recu 1;;
2 ) echo recu 2;;
3 ) echo recu 3;;
* ) echo recu autre...;;
esac
```

```
MBP-de-Thibault:Scripts Thibault$ emacs sccase.sh
MBP-de-Thibault:Scripts Thibault$ ./sccase.sh 2
recu 2
MBP-de-Thibault:Scripts Thibault$ ./sccase.sh 4
recu autre...
MBP-de-Thibault:Scripts Thibault$
```

Ecrire un script

- Boucle
  - Pour

```
#/bin/bash

for i in *  ; do
    echo $i
done
```

```
MBP-de-Thibault:Scripts Thibault$ ./scforf.sh
#scif.sh#
sc1.sh
sc1.sh~
sccase.sh
sccase.sh~
scfor.sh
scfor.sh~
scforf.sh
scforf.sh~
scif.sh
scif.sh~
```

```
#!/bin/bash

for ((i=0;10-$i;i++))  ; do
    echo $i
done
```

```
MBP-de-Thibault:Scripts Thibault$ ./scfor.sh
0
1
2
3
4
5
6
7
8
9
MBP-de-Thibault:Scripts Thibault$
```

Ecrire un script

- Boucle
  - Tant que

```
MBP-de-Thibault:Scripts Thibault$ ./scwhile.sh
entrez un nombre entre 1 et 20
10
trop grand, entre 1 et 20
5
trop grand, entre 1 et 20
3
trop petit, entre 1 et 20
4
bravo, le nombre etait 4
MBP-de-Thibault:Scripts Thibault$
```

```
#!/bin/bash

nb=$(($RANDOM % 20) + 1))

echo "entrez un nombre entre 1 et 20"
read i
while [ $i -ne $nb ]; do
        if [ $i -lt $nb ]; then
                echo "trop petit, entre 1 et 20"
        else
                echo "trop grand, entre 1 et 20"
        fi
        read i
done
echo "bravo, le nombre etait $nb"
```

Ecrire un script

- Comparaison de chaine de caractères :
  - S : vrai si la chaine n'est pas vide
  - S1 = S2 : vrai si S1 et S2 sont identiques
  - S1 !=S2 : vrai si S1 et S2 sont différentes

- Comparaison arithmétique :
  - Ex1 –eq Ex2 : vrai si Ex1 = Ex2
  - Ex1 –ne Ex2 : vrai si Ex1 ≠ Ex2
  - Ex1 –gt Ex2 : vrai si Ex1 > Ex2
  - Ex1 –ge Ex2 : vrai si Ex1 ≥ Ex2
  - Ex1 –lt Ex2 : vrai si Ex1 < Ex2
  - Ex1 –le Ex2 : vrai si Ex1 ≤ Ex2

- Test sur un fichier
  - -d fic : vrai si le fichier fic est un répertoire
  - -e fic : vrai si le fichier fic existe
  - -f fic : vrai si le fichier fic est régulier
  - -s fic : vrai si le fichier fic a une taille non nulle

Ecrire un script

Calcul d'un PGCD par la méthode d'Euclide

```
#!/bin/bash
nb1=$1
nb2=$2

reste=$(($nb1 % $nb2))

while [ $reste -ne 0 ] ;do
    nb1=$nb2
    nb2=$reste
    reste=$(($nb1 % $nb2))
done
echo $nb2
```

```
50
MBP-de-Thibault:Scripts Thibault$ ./scpgcd.sh 450 150
150
MBP-de-Thibault:Scripts Thibault$ emacs scpgcd.sh
MBP-de-Thibault:Scripts Thibault$ ./scpgcd.sh 450 151
1
MBP-de-Thibault:Scripts Thibault$ ./scpgcd.sh 450 153
9
MBP-de-Thibault:Scripts Thibault$
```

Ecrire un script

```
#!/bin/bash

if [ -e $1 ] ; then
    rm $1
fi

for i in * ; do
    cat $i >> $1
    echo "\n" >> $1
done

echo `wc -c $1`
```

```
pc10:Scripts Thibault$ emacs scfile.sh
pc10:Scripts Thibault$ ./scfile.sh Toto
1149 Toto
pc10:Scripts Thibault$
```

Ecrire un script

Il convient de pratiquer ! Un rapide résumé est disponible dans le chapitre dédié :

https://repo.zenk-security.com/Linux%20et%20systemes%20d.exploitations/Linux-in-a-Nutshell-6th-Edition.pdf

Références

- Systèmes d'Exploitation, Andrew Tanenbaum, Pearson, 2008
- Advanced Programming in Unix Environment, Richard Stevens, Stephen A. Rago, Addison-Wesley, 2013
- The Design and Implementation of the 4.4 BSD Operating System, Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quaterman, Addison-Wesley, 1996
- Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Wiley, 2002
- Linux in a Nutshell, Ellen Siever, Stephen Figgins, Robert Love, Arnold Robbins,O'Reilly, 2009