Programmation Système
Partie 1

Xavier Bultel
Basé sur le cours de Jérémy Briffaut

INSA CVL - 3A STI

29 septembre 2021

Table des matières

Sommaire

Objectif du cours

- Comprendre ce qu'est un Système d'Exploitation
- Comprendre comment fonctionne un Système d'Exploitation
- Etudier les bases de la programmation Système

Plan du Cours

- Introduction
  - Système d'exploitation
  - Historique UNIX
  - Mode d'exécution d'un processeur
  - Norme (Unix->POSIX->SUS)
- Utilisation de la bibliothèque C standard et de l'API POSIX
  - interagir avec le système d'exploitation
  - Appel système
  - Cycle d'exécution d'un programme
  - Accès à l'environnement
  - Gestion des erreurs
  - utilisation appels système
  - implantation de la bibliothèque au dessus des appels système

Plan du Cours

- Les entrées-sorties
  - Généralités
  - Manipulation des i-noeuds
  - Primitives de base
  - Descripteurs
- Système de fichiers
  - Exemple : ext2, FAT32
- Processus (J. Briffaut)
  - création, terminaison, interruptions, ordonnancement
  - Gestion des processus
  - Attributs des processus
  - Vie des processus

Evaluation

Contrôle continu :

- 2 interrogations (cours 4, cours 8)
  Soyez là !
- 1 TD à rendre, 1 TD noté en autonomie.

Sommaire

Sommaire

Qu'est-ce qu'un système d'exploitation ?

- Qu'est-ce qu'un système d'exploitation ?

def1 Programme qui agit comme un intermédiaire entre l'utilisateur d'un ordinateur et le matériel ; il fournit un environnement dans lequel l'utilisateur peut exécuter des programmes de manière *pratique et efficace*.

def2 Le **système d'exploitation**, abrégé SE (en anglais *operating system*, abrégé OS), est l'ensemble de programmes central d'un appareil informatique qui sert d'interface entre le matériel et les logiciels applicatifs.

Qu'est-ce qu'un système d'exploitation ?

- Qu'est-ce qu'un système d'exploitation ?

def1 Programme qui agit comme un intermédiaire entre l'utilisateur d'un ordinateur et le matériel ; il fournit un environnement dans lequel l'utilisateur peut exécuter des programmes de manière *pratique et efficace*.

def2 Le **système d'exploitation**, abrégé SE (en anglais *operating system*, abrégé OS), est l'ensemble de programmes central d'un appareil informatique qui sert d'interface entre le matériel et les logiciels applicatifs.

Qu'est-ce qu'un système d'exploitation ?

- Un système d'exploitation :
  - Couche logicielle intercalée entre l'ordinateur et ses utilisateurs, lancée au démarrage ;
  - Intermédiaire entre les logiciels applicatifs et le matériel ;
  - Facilite l'exploitation des périphériques matériels dont il coordonne et optimise l'utilisation ;
  - Met à disposition des logiciels applicatifs une interface de programmation standardisée d'utilisation des matériels ;

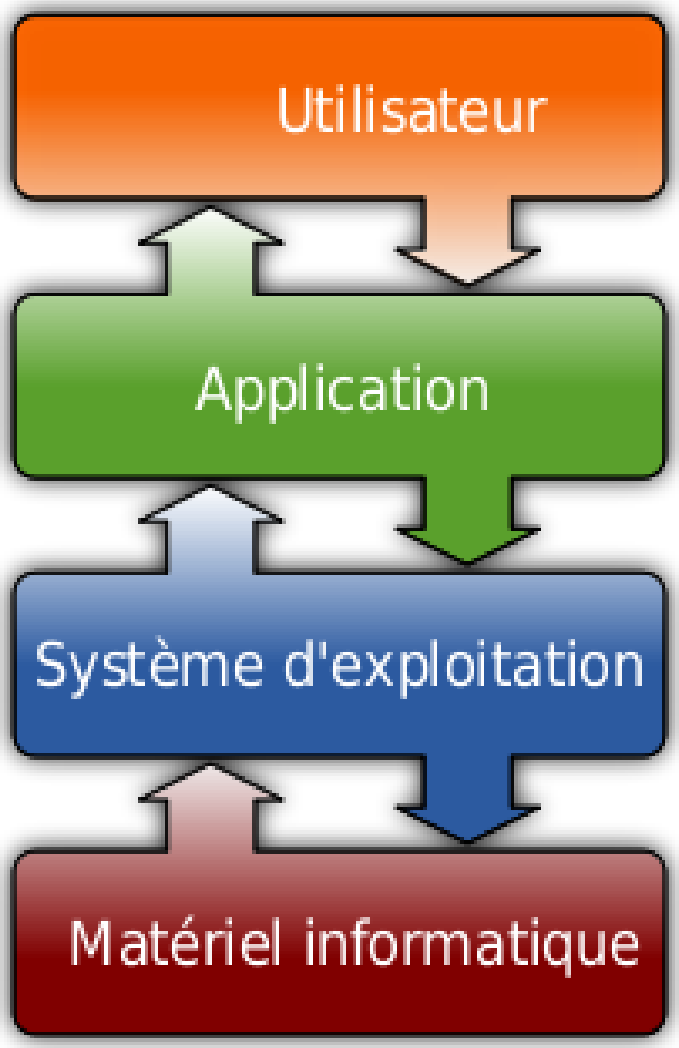
Qu'est-ce qu'un système d'exploitation ?

- Ordinateur = ensemble de ressources physiques
  - Processeur/Unité Centrale (CPU)
  - Mémoire principale
  - Mémoires secondaires
  - Périphériques d'E/S
  - Périphériques internes (horloge,...)
- Son utilisation génère des ressources logiques
  - Processus
  - Fichiers
  - Bibliothèques "Système" partagées
  - Sessions utilisateurs

Qu'est-ce qu'un système d'exploitation ?

- Un système d'exploitation :
  - Alloue les ressources aux utilisateurs
  - Controle leur bonne utilisation
  - Problèmes : efficacité, fiabilité, sécurité, équité, etc.

Place du système d'exploitation



Sommaire

Historique des systèmes informatiques

- 1936-55 : Les premiers ordinateurs
- 1955-65 : Traitement par lots
- 1965 : Tamponnement des entrées/sorties
- 1965-70 : Multiprogrammation, Temps partagé
- 1980 : Ordinateurs personnels

John Von Neumann

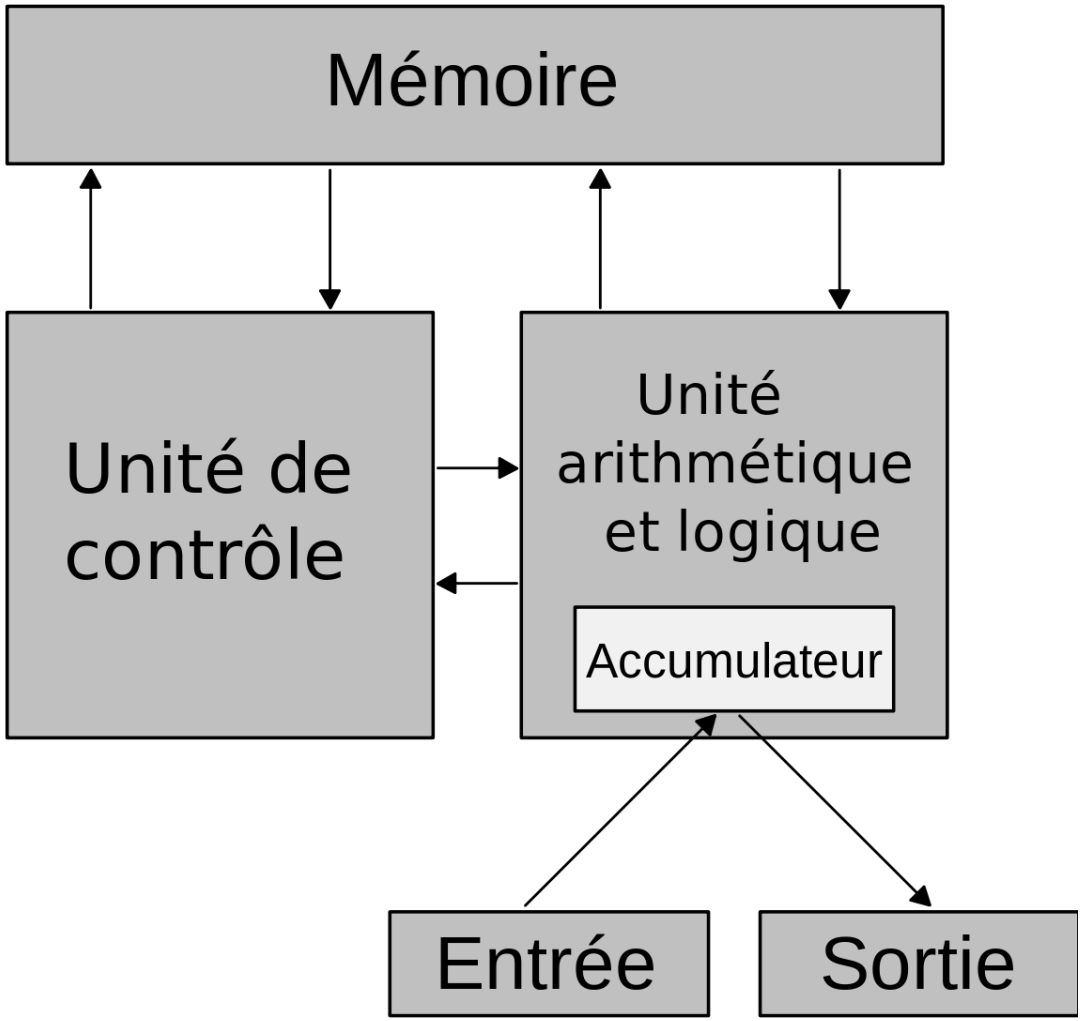
- 1946 : John Von Neumann (1903-1957) et le principe du premier ordinateur
  - Possibilité de mémorisation des résultats partiels étendue à l'enregistrement des programmes et des données
  - Exécution séquentielle des instructions enregistrées avec possibilité de branchement conditionnel

John Von Neumann

- De fait, un ordinateur doit disposer des fonctions suivantes :
  1. Moyen d'entrée (échange et recensement des informations)
  2. Moyen de mémorisation des informations
  3. Moyen de calcul (traitement de l'information)
  4. Moyen de sortie (résultats)
  5. Moyen de décision
  6. Gestion des données et des instructions (stockées dans le moyen de mémorisation sous la même forme)

- Babbage avait déjà introduit les idées 1 à 5.
- Von Neumann introduit, lui, le point 6.

Architecture de Von Neumann



Premiers Ordinateurs (1936 - 1955)

Ordinateur :

- Machine programmable
- Traite des informations *numériques* ou *discrètes* ($\neq$ *analogiques* ou *continues*)

- Apparition des premiers ordinateurs :
  - à relais et à tubes vides
  - programmés par tableaux de connecteurs, puis par cartes perforées (1950)
  - mono-utilisateur et mono-tâches
  - apparition du termes **BUG**

Premiers Ordinateurs (1936 - 1955)

Ordinateur :

- Machine programmable
- Traite des informations *numériques* ou *discrètes* ($\neq$ *analogiques* ou *continues*)

- Apparition des premiers ordinateurs :
  - à relais et à tubes vides
  - programmés par tableaux de connecteurs, puis par cartes perforées (1950)
  - mono-utilisateur et mono-tâches
  - apparition du termes **BUG**

Premiers Ordinateurs (1936 - 1955)

- 1947 : invention du transistor (laboratoires Bell)
  $\Rightarrow$ invention de la mémoire physique

- 1950 : premier ordinateur digne de ce nom, UNIVAC 1
  - Chacun est unique
  - Sans interface
  - Les utilisateurs sont multi-fonctions : fabrication, programmation, maintenance et utilisation
  - Un seul programme à la fois

Premiers Ordinateurs (1936 - 1955)

- 1947 : invention du transistor (laboratoires Bell)
  $\Rightarrow$ invention de la mémoire physique

- 1950 : premier ordinateur digne de ce nom, UNIVAC 1
  - Chacun est unique
  - Sans interface
  - Les utilisateurs sont multi-fonctions : fabrication, programmation, maintenance et utilisation
  - Un seul programme à la fois

Premiers Ordinateurs (1936 - 1955)

- Séance-type de programmation :
  - Ecriture sur cartes (programmeur)
  - Chargement des cartes compilateur (opérateur)
  - Chargement des cartes du programme
  - Création du code intermédiaire (assemblage)
  - Chargement des cartes de l'assembleur
  - Création du code en langage machine
  - Exécution du programme

Problèmes :

- Exécution des instructions d'un programme sans intervention extérieure possible
- Temps d'inactivité importants : matériel sous-employé

Premiers Ordinateurs (1936 - 1955)

- Séance-type de programmation :
  - Ecriture sur cartes (programmeur)
  - Chargement des cartes compilateur (opérateur)
  - Chargement des cartes du programme
  - Création du code intermédiaire (assemblage)
  - Chargement des cartes de l'assembleur
  - Création du code en langage machine
  - Exécution du programme

Problèmes :

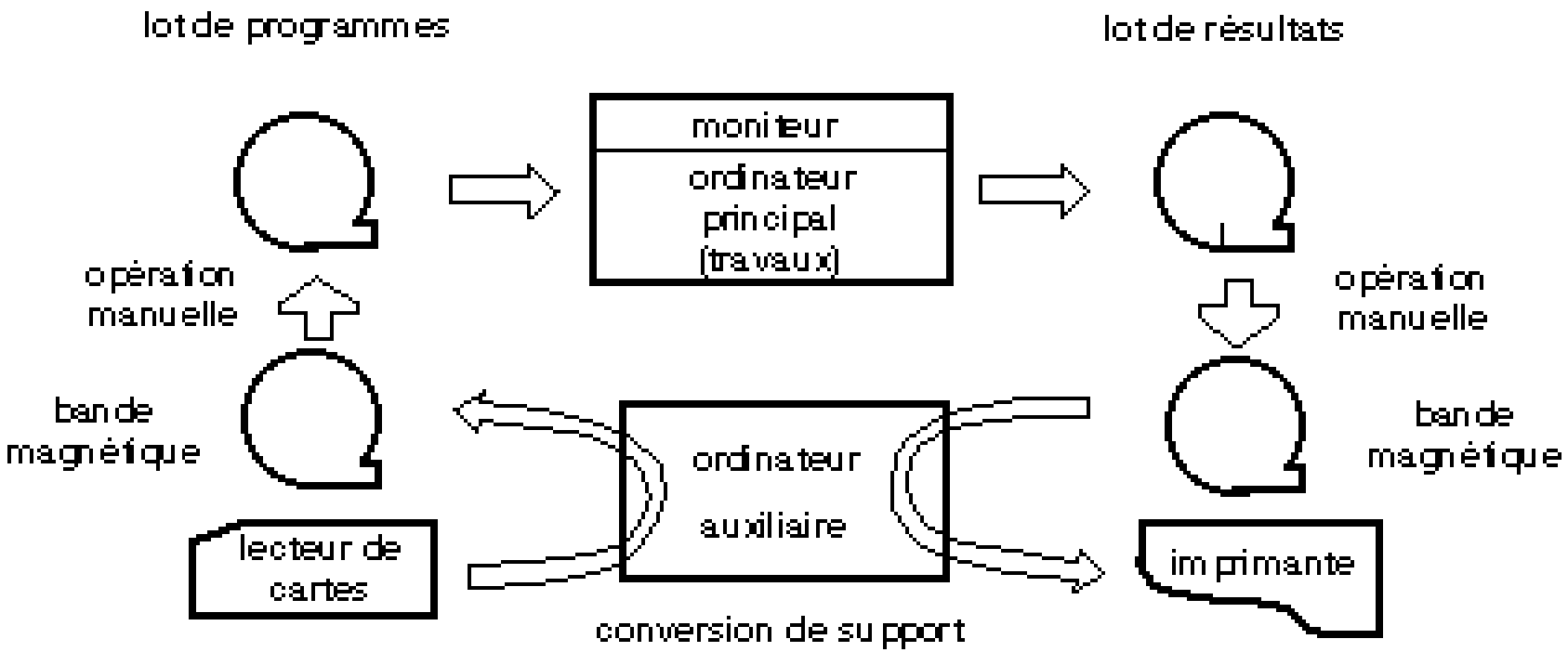
- Exécution des instructions d'un programme sans intervention extérieure possible
- Temps d'inactivité importants : matériel sous-employé

Premiers Ordinateurs (1936 - 1955)

Traitement par Lots (1955 - 1965)

- Traitement par lots :
  - Emergence des supports magnétiques
    - conservation de programmes binaires importants sur ce support
  - Améliorations :
    - Regroupement et exécution par groupe des travaux similaires (batch processing)
    - Moniteur résident : enchaine automatiquement les travaux

Traitement par Lots (1955 - 1965)



Traitement par Lots (1955 - 1965)

- Conséquences :
  - L'utilisateur n'accède plus directement à la machine
  - Enchainement automatique des programmes
  - Débit des travaux amélioré
  - Temps de réponse augmenté

Tamponnement des E/S (1965)

Problèmes :

- Différence de vitesse entre les E/S et l'UC
- De fait, temps d'inactivité entre les E/S

- 1965 : Tamponnement des E/S
  - Idée : rendre les opérations d'E/S autonomes
  - Utilisation de tampons («buffers»)
  - Stocker les données lues non encore nécessaires
  - Stocker les requêtes de sorties quand le périphérique n'est pas disponible
  - Introduction du mécanisme d'interruption

Tamponnement des E/S (1965)

Problèmes :

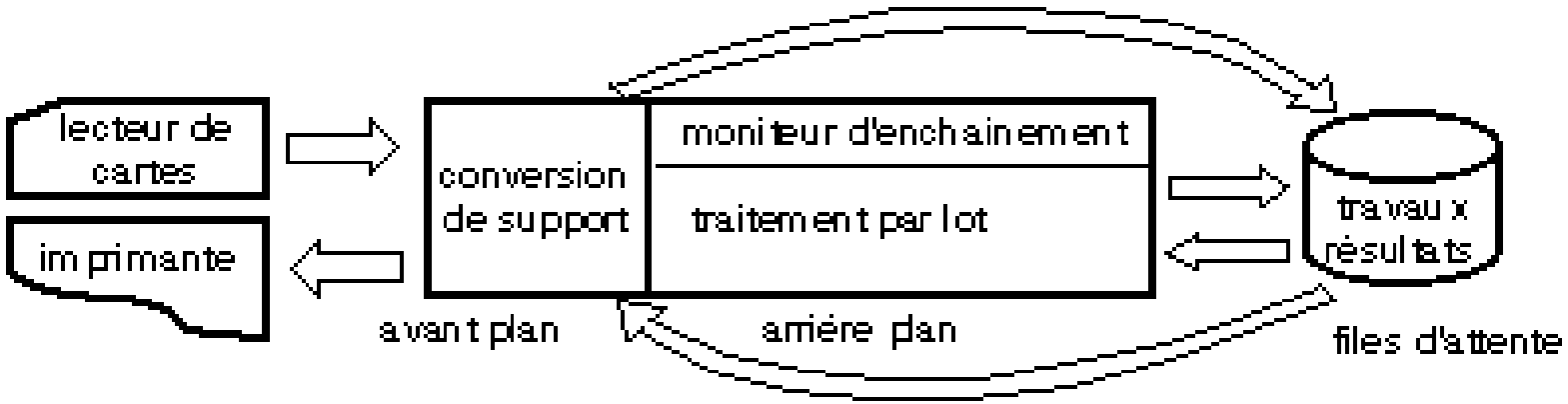
- Différence de vitesse entre les E/S et l'UC
- De fait, temps d'inactivité entre les E/S

- 1965 : Tamponnement des E/S
  - Idée : rendre les opérations d'E/S autonomes
  - Utilisation de tampons («buffers»)
  - Stocker les données lues non encore nécessaires
  - Stocker les requêtes de sorties quand le périphérique n'est pas disponible
  - Introduction du mécanisme d'interruption

Tamponnement des E/S (1960)

Avantages :

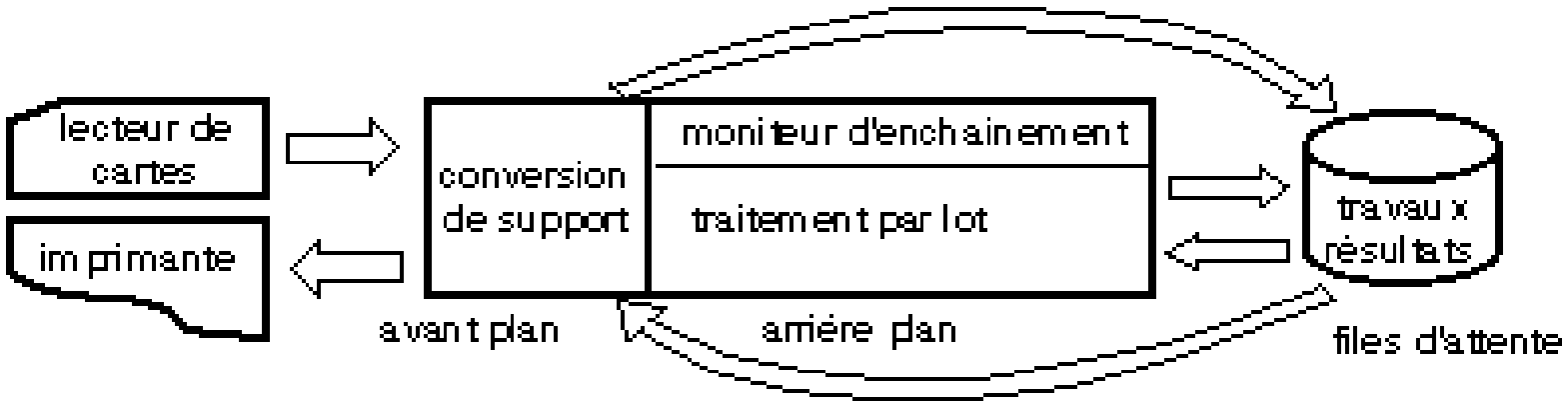
- Diminution coût/performance
- Plus d'ordinateurs annexes/manipulation de bandes



Tamponnement des E/S (1960)

Avantages :

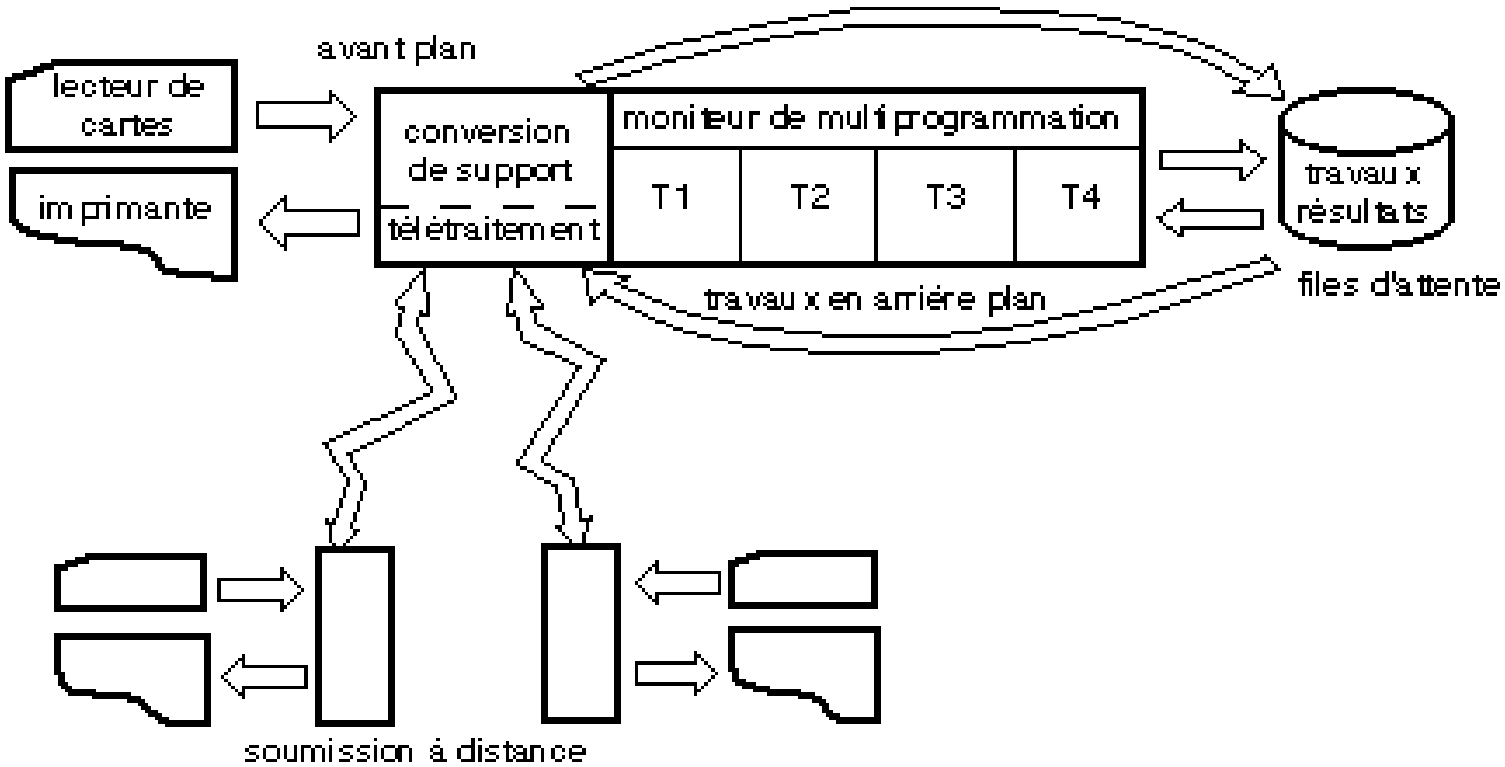
- Diminution coût/performance
- Plus d'ordinateurs annexes/manipulation de bandes



Multiprogrammation (1965)

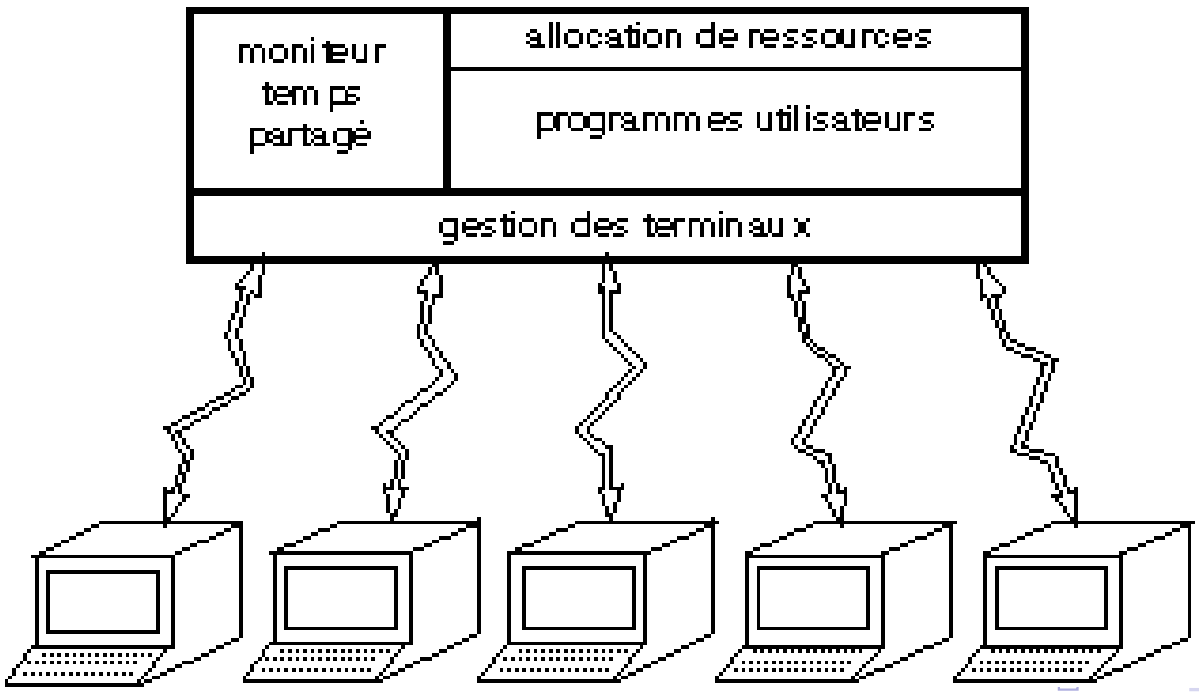
- 1965-70 : La multiprogrammation :
  - Augmentation de la taille de la mémoire principale –> multiprogrammation
    - Charger plusieurs travaux en mémoire simultanément
    - Faire un autre travail au lieu d'attendre
  - Amélioration : exécution parallèle d'un ensemble de programmes
  - Les E/S sont effectuées de façon asynchrone avec des calculs sur l'UC

Multiprogrammation (1965)



Temps Partagé (1965 - 1980)

- 1970 : le temps partagé :
  - Améliorations :
    - Partage du temps en quanta via une horloge temps réel
    - Affectation du processeur à un programme durant un quantum
    - Prise en compte rapide des nouveaux programmes



Ordinateur Individuel (1980 -)

- Avènement de l'ordinateur individuel
- Développement des micro-ordinateurs
  - 1981 : IBM lance le PC
  - Poste de travail conçu de façon autonome, pour une utilisation par une seule personne
  - Fin des années 80 : X-Window
  - Année 90 : essor d'Internet
  - Réseaux et systèmes répartis

Sommaire

Sommaire

Types de système d'exploitation

- Il n'existe pas de système d'exploitation efficace dans tous les contextes d'utilisation.
- On a donc différentes catégories/familles de systèmes :
  - Mono-utilisateur
  - Controle de processus (Ex : machines outils)
  - Serveurs de fichiers
  - Transactionnel
  - Général

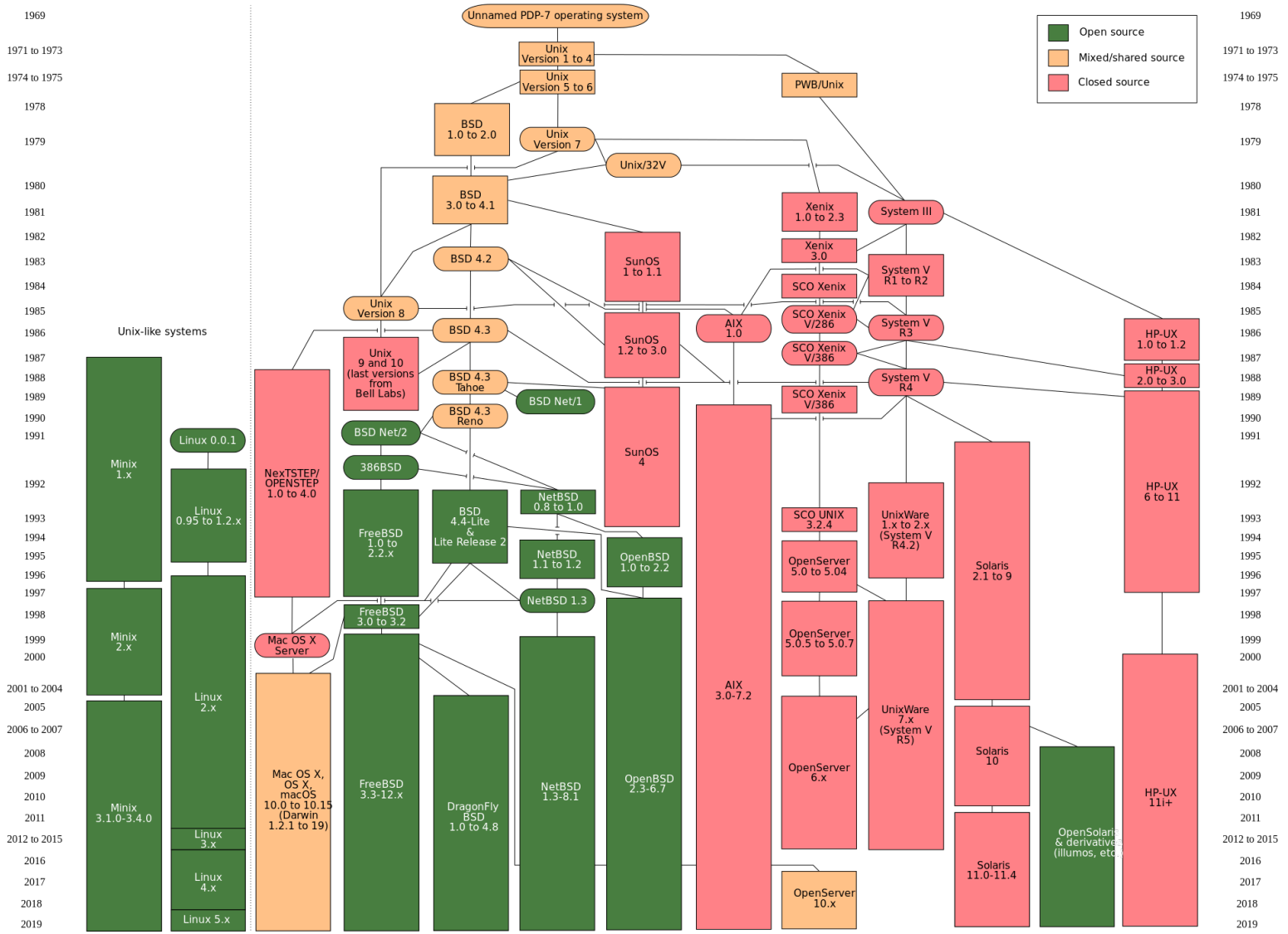
Sommaire

Unix

- Système d'exploitation crée en 1969 sous l'impulsion de Kenneth Thompson et Dennis Ritchie
- Noyau en langage C
- Au coeur du développement informatique depuis son apparition
- Multi-utilisateurs
- Multi-tâches
- Temps partagé

Linux

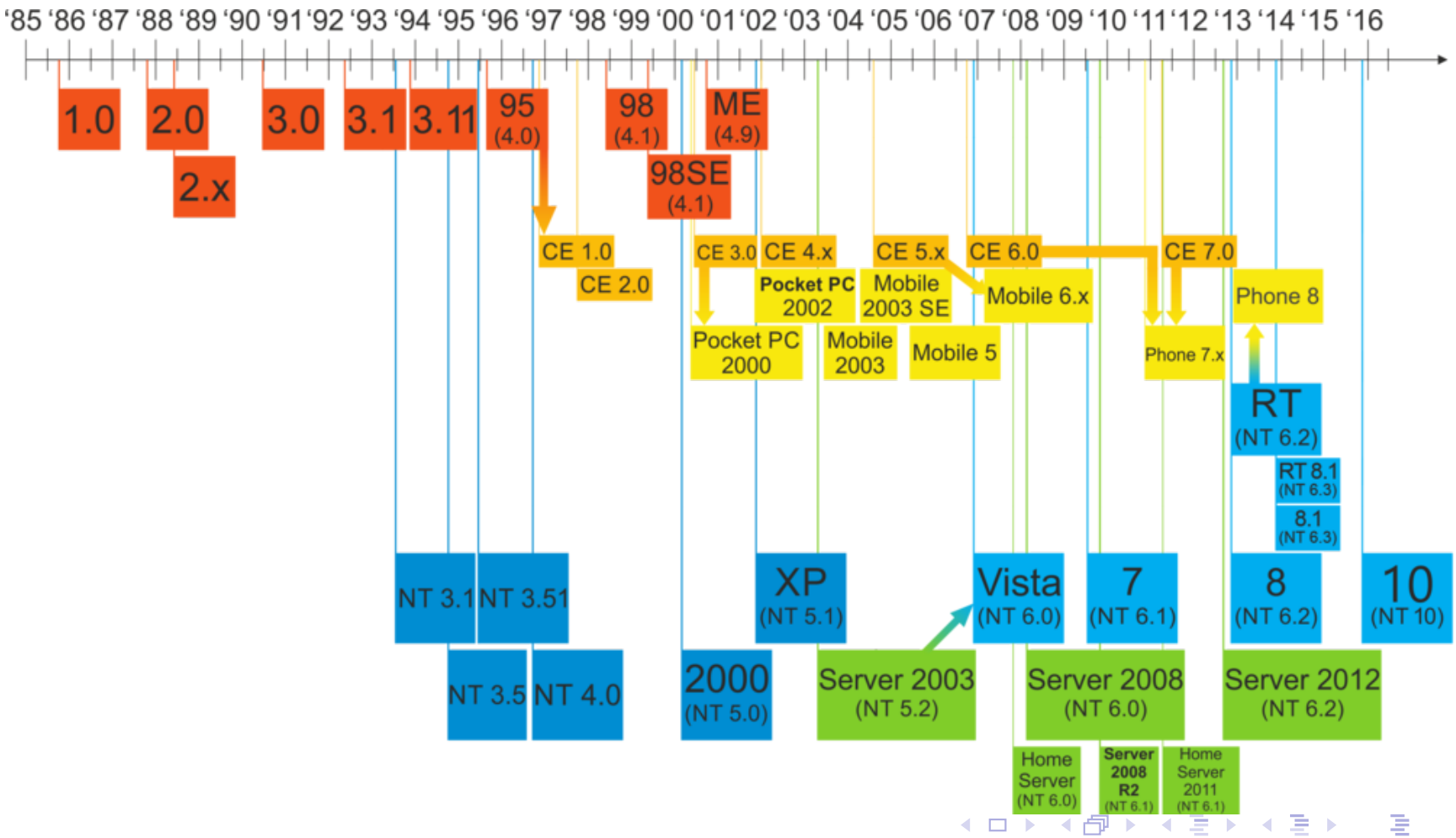
- Basé sur Unix
- 1984 : lancement du projet GNU par Richard M. Stallman
- 1991 : Linus Torvalds propose la première version d'un noyau baptisé Linux
- 1992 : système d'exploitation GNU/Linux
- Disponible sous la forme de multiples distributions


Open source
Mixed/shared source
Closed source
Unix-like systems
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2015
2016
2017
2018
2019
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
PWB/Unix
BSD 1.0 to 2.0
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
SunOS 1 to 1.1
System V R1 to R2
SCO Xenix
Unix Version 8
SCO Xenix V/286
AIX 1.0
System V R3
BSD 4.3
SCO Xenix V/386
Unix 9 and 10 (last versions from Bell Labs)
HP-UX 1.0 to 1.2
HP-UX 2.0 to 3.0
BSD 4.3 Tahoe
BSD Net/1
System V R4
SCO Xenix V/386
BSD 4.3 Reno
BSD Net/2
SunOS 1.2 to 3.0
386BSD
SunOS 4
Minix 1.x
Linux 0.0.1
NexTSTEP/ OPENSTEP 1.0 to 4.0
Linux 0.95 to 1.2.x
NetBSD 0.8 to 1.0
HP-UX 6 to 11
SCO UNIX 3.2.4
UnixWare 1.x to 2.x (System V R4.2)
FreeBSD 1.0 to 2.2.x
BSD 4.4-Lite & Lite Release 2
Solaris 2.1 to 9
OpenServer 5.0 to 5.04
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
Minix 2.x
NetBSD 1.3
FreeBSD 3.0 to 3.2
OpenServer 5.0.5 to 5.0.7
Mac OS X Server
AIX 3.0-7.2
Linux 2.x
UnixWare 7.x (System V R5)
Solaris 10
HP-UX 11i+
Mac OS X, OS X, macOS 10.0 to 10.15 (Darwin 1.2.1 to 19)
FreeBSD 3.3-12.x
DragonFly BSD 1.0 to 4.8
NetBSD 1.3-8.1
OpenBSD 2.3-6.7
OpenServer 6.x
Minix 3.1.0-3.4.0
Linux 3.x
Linux 4.x
Linux 5.x
Solaris 11.0-11.4
OpenSolaris & derivatives (illumos, etc.)
OpenServer 10.x

MS-DOS

- Lancé en 1981, en même temps que le PC d'IBM
- Mono-utilisateur
- Mono-tâche
- Couche graphique : Invite de commande, puis Windows jusqu'à Windows 2000
- Désormais, l'invite de commande est un émulateur intégré.

Evolution de Microsoft Windows



Sommaire

Architecture Générale

- Ordinateur :
  - Processeur (traitements)
  - Mémoire principale (rangement des données/résultats)
  - Périphériques (échange d'informations avec l'extérieur)
  - Bus (liaison entre les constituants)

Structures internes des systèmes d'exploitation généraux

- Principaux types de systèmes :
  - Noyau monolithiques
  - Noyau monolithiques modulaires
  - Micro-noyaux
  - Hybride

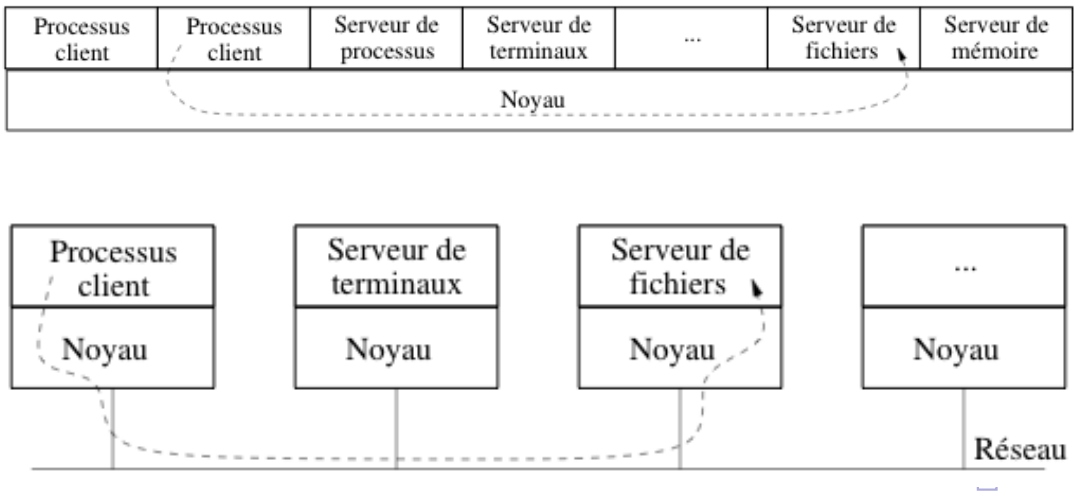
Noyau monolithique

- Caractérisé par l'absence de structure interne : un seul bloc de code
- Le système est une collection de procédure, chacune visible de tous les autres, et pouvant appeler toute autre procédure qui lui est utile
- La seule barrière est la protection entre le monde utilisateur et le monde noyau
- Code difficile à maintenir, mémoire partagée ($\neq$ sécurité)
- mais une efficacité excellente
- Noyaux monolithiques modulaires : code fondamentale du système séparé de quelques fonctions (pilotes)

Micro-noyaux

- Basé sur une approche horizontale
- SE vu comme un système distribuée (système "client-serveur")
  - Notion de micro-noyaux
  - Chaque procédure système est séparée
  - Chacune a son propre espace d'adressage

=> Sécurité et lisibilité du code, mais problèmes de latences



Sommaire

Sommaire

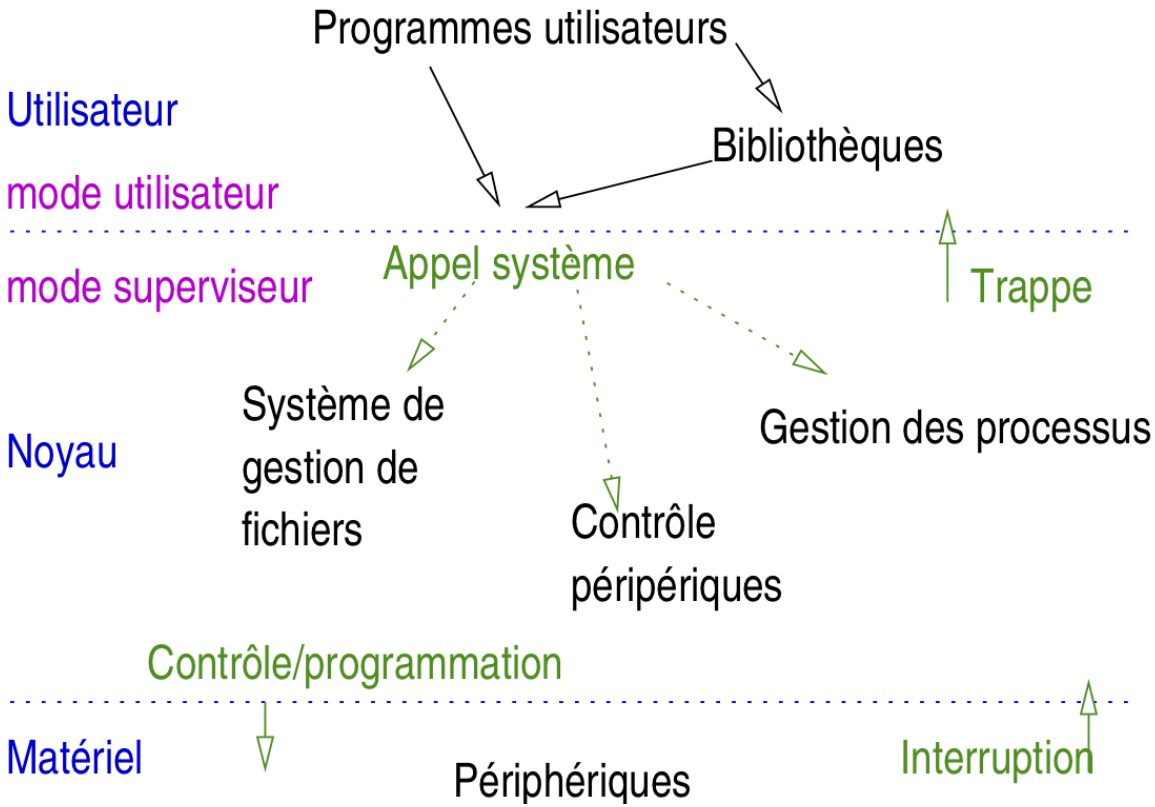
Rappel : Modes d'exécution d'un processeur

- Différencier les modes d'exécution
  - Mode Utilisateur
  - Mode Superviseur (ou mode moniteur/ système/ privilégié)
- Fonctionnement :
  - A l'initialisation du système, le matériel démarre en mode superviseur.
  - L'OS est chargé et démarre des processus utilisateur en mode utilisateur.
  - Lors d'un déroutement/interruption, la matériel passe en mode superviseur.
  - Le système revient toujours en mode utilisateur avant de passer le controle à un programme utilisateur.

Mode d'exécution

- Utilisateur
- Superviseur

- Passage mode superviseur : appels système, trappes, IT



Interruptions

- Fonctions généralement communes aux mécanismes d'interruption :
  - Appel de la routine de traitement via une table de pointeurs
  - Sauvegarde de l'adresse de l'instruction interrompue
  - Après traitement de l'interruption, chargement de l'adresse de retour dans le compteur ordinal

- Les systèmes d'exploitation modernes sont dirigés par les interruptions.

Structure en couche d'un Unix

- Niveau utilisateur
  - applications utilisateurs
  - logiciel de base
  - bibliothèques système
  - appels de fonctions
- Niveau Noyau
  - gestion des processus
  - système de fichiers
  - gestion de la mémoire
- Niveau matériel
- Système d'exploitation
  - bibliothèques système
  - noyau

Sommaire

Normalisation de l'interface

- Unix
  - système d'exploitation Ken THOMPSON et Dennis RITCHIE, Bell Labs, 1969
  - distribution du code source
  - multiples versions (branches BSD, System III...)
- POSIX
  - Portable Open System Interface
  - X for Unix
  - standard IEEE, 1985
  - interface standardisée des services fournis par le système
- Single Unix Specification, SUS
  - X/Open reprend les activités de normalisation POSIX, 1999

Normalisation de l'interface

Norme	Symbole	Valeur
POSIX.1-1988	_POSIX_SOURCE	
POSIX.1-1990 (1003.1)	_POSIX_C_SOURCE	1
POSIX.1b-1993	_POSIX_C_SOURCE	199309L
POSIX.1c-1996	_POSIX_C_SOURCE	199506L
POSIX.1-2001	_POSIX_C_SOURCE	200112L
POSIX.1-2008	_POSIX_C_SOURCE	200809L
XPG3	_XOPEN_SOURCE	1
XPG4	_XOPEN_SOURCE	4
XPG4	_XOPEN_VERSION	4
SUS	_XOPEN_SOURCE	1
SUS	_XOPEN_SOURCE_EXTENDED	4
SUSv2	_XOPEN_SOURCE	500

Pour compiler de manière conforme à POSIX.1 :

```
gcc -D_POSIX_C_SOURCE=1 -c source.c
```

Normalisation de l'interface

- POSIX.1 :
  - Processus
  - erreur de segmentation, instructions illégales
  - bibliothèque standard
  - entrées-sorties
- POSIX.1b, temps réel :
  - ordonnancement, signaux
  - sémaphores
  - entrées-sorties synchrones/asynchrones
  - verrouillage de la mémoire
- POSIX.1c, processus légers (threads) :
  - création, utilisation des threads
  - ordonnancement, synchronisation ...

Fourniture de l'interface POSIX

- Norme POSIX = interface d'utilisation du système
  - description des fonctions d'appel des services système fournis par le noyau
  - portabilité des applications
  - ne définit pas la construction du système d'exploitation, noyau
- POSIX et l'interface d'un système Unix
  - interface POSIX = l'interface du système !
  - interface native
  - une fonction POSIX = un appel système Unix
- POSIX et l'interface de systèmes propriétaires
  - Windows, VMS, Mach...
  - interface POSIX $\neq$ interface du système
  - bibliothèque niveau utilisateur au dessus des appels système
  - une fonction POSIX = un appel fonction bibliothèque utilisateur = un / multiples appels système

Interface POSIX I

- Interface POSIX : direct avec le noyau.
- Fonctions : appels systèmes
  - Basculement du *user mode* au *kernel mode*
  - Exécution en mode noyau
  - Interruption du processus courant (sauvegarde)
- Exécute des opérations "dangereuses"
- Liste des appels systèmes : syscall.h

Raccourci : numéro de l'appel suivi des paramètres :

```
#include <sys/syscall.h>
int syscall (int numero...)
```

Interface POSIX II

Exemple d'appel système :

```
#include <sys/syscall.h>
#include <unistd.h>
int main (){
    (void)syscall(SYS_write,STOUT_FILENO, "hello\n",6);
}
```

Performances : éviter les appels systèmes multiples !

Interface POSIX III

- sys/types.h : types de base

Type	Description
dev_t	Numéro de périphérique
uid_t	Identifiant de l'utilisateur
gid_t	Identifiant de groupe d'utilisateurs
ino_t	Identifiant de fichier (numéro de série)
mode_t	Droits d'accès et types de fichiers (masque)
nlink_t	Compteur de liens
off_t	Taille de fichier et déplacement
pid_t	Identifiant de processus
fsid_t	Identifiant de système de fichiers
size_t	Taille
ssize_t	Taille signée

Sommaire

Programmation système en C

- Langage C pour programmer le système
  - sémantique claire
  - efficacité
  - accès à toutes les structures de la machine (registres, bits...)
  - allocation mémoire explicite
  - autres approches possibles : langage dédié
- Langage C interface naturelle avec le système
  - bibliothèques écrites en C
  - utilisation de la bibliothèque depuis le C
  - autres approches possibles : Java, OCaml

Bibliothèque C standard et POSIX

- Bibliothèque C
  - normalisation ISO du langage C
  - section 3 de man
  - *man 3 malloc*
- POSIX
  - normalisation Single Unix
  - section 2 de man
  - *man 2 sbrk*

Bibliothèque et appel système I

- **Utilisation :** appel système semblable à une fonction de bibliothèque standard
  - comme des appels de fonctions C
- **Fonctionnement :** Appel système différent d'une fonction de bibliothèque standard
  - appel système :
    - pas d'édition de liens
    - exécution de code système
  - bibliothèque standard :
    - abstraction de plus haut niveau
    - édition de liens avec la bibliothèque

Bibliothèque et appel système II

- Appels système
  - Manipulation du système de fichiers et entrées/sorties
  - Gestion des processus
    - processus = exécution d'un programme
    - allocation de ressources pour les processus (mémoire...)
    - lancement, arrêt, ordonnancement des processus
  - Communications entre processus

Terminaison d'un appel système

- Sémantique POSIX :
  - comportement
  - y compris en cas d'erreur
  - liste des erreurs pouvant être retournées
  - voir le manuel man

Bibliothèque et appel système III

Gestion des erreurs :

- Code de retour d'erreur : -1
- Utilisation de errno :

#include <errno.h>
extern int errno;
void perror (**const char** *message); *// affiche le message*
char * strerror (**int** symboleErreur); *// retourne le message*

Erreurs répertoriées par POSIX (extrait) :

Symbole	Signification
EPERM	Opération non autorisée
ENOENT	Fichier ou répertoire inexistant
ESRCH	Processus inexistant
EBADF	Descripteur d'E/S non valide

Bibliothèque et appel système IV

- Valeur de retour d'un appel système
  - retourne -1 en cas d'erreur
  - positionne la variable globale errno
  - perror() produit une message décrivant la dernière erreur (appel système ou fonction bibliothèque)
- Exemple typique

```
if(syscall(SYS_write, 100, "hello\n", 6) == -1){
        perror("Avec perror");
        printf("Avec strerror : %s\n",strerror(errno));
}
```

- Test systématique des retours des fonctions (laborieux...)

Bibliothèque et appel système V

Bibliothèques standard :

- Nombreuses bibliothèques
  - entrées/sorties formatées & bufferisées
  - fonctions mathématiques
  - allocation mémoire dynamique
  - etc.
- Abstraction de plus haut niveau
- Performance
- nombre appels système réduits
- exemple : allocation mémoire malloc()/sbrk()

Sommaire

Environnement des processus I

Environnement lié à la ligne de commande
Variables d'environnement :

- L'environnement est accessible via : **extern char** **environ;
- Ou par les fonctions POSIX :

```
#include <stdlib.h>
char *getenv(const char *nomDeVariable);
int putenv(const char *coupleNomValeur);
```

Environnement des processus II

- Accès aux variables d'environnement (*$PATH, $USER...*)
  - variable globale environ
  - Tableau de chaînes de carractères terminé par NULL

```
#include <stdio.h> #include <stdlib.h>
extern char **environ;
int main (int argc, char *argv[]){
        char **envp = environ;
        while (*envp) printf("%s\n", *envp++);
        exit(EXIT_SUCCESS);
}
```

Environnement des processus III

- Fonction POSIX getenv()

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main (int argc, char *argv[]){
        char *username;
        username = getenv("USER");
        assert(username != NULL);
        printf("Hello %s\n", username);
        exit(EXIT_SUCCESS);
}
```

Interface avec le programme appelant

- Programme = nouvelle commande
  - appel depuis une autre commande
  - depuis un shell
  - *mprogram [arguments]...*
- Accès aux arguments de la commande

---

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

argc : nombre d'arguments

arguments : argv[1] a argv[argc]

nom commande : argv[0]

Terminaison d'un processus I

- Terminaison d'un processus :
  - de lui-même : exit ou _exit ou **return**
  - d'un signal ***kill***

Etapes de terminaison :

- Fermeture de tous les descripteurs ouverts.
- Libération de toutes les ressources allouées.
- Envoi du signal SIGHUP à tous les processus du groupe si le processus est leader.
- Rattachement de tous les fils au processus de pid 1.
- Réveil du père attendant via wait ou waitpid.

Terminaison d'un processus II

- Un programme termine à la fin de main()
- Retourne une valeur à l'environnement
  - Retour : EXIT_SUCCESS/ EXIT_FAILURE
  - fonction exit() de la bibliothèque C, fait appel à fonction POSIX _exit()
- Enregistrement de fonctions de terminaison par atexit()

```
void bye(void) { printf("A la semaine prochaine!\n"); }
int main (int argc, char *argv[]) {
        atexit(bye);
        printf("Hello...\n");
        exit(EXIT_SUCCESS);
}
```

Exécution de commandes shell

Commande system :

```
#include <stdlib.h>
int system(const char * commande);
```

Exemple d'appel système :

```
#include <stdlib.h>
int main(int argc, char ** argv) {
  int rep;
  rep = system(argv[1]);
  printf("Valeur renvoyee: %d", rep);
}
```

Sommaire

Constantes de configuration POSIX

```
#include <unistd.h>
long sysconf(int name);
long pathconf(const char *reference, int symbole);
long fpathconf(int numeroDescripteur, int symbole);
```

- sysconf : Retrouver la valeur associée au symbole POSIX.

Nom	Symbole	Explication
ARG_MAX	_SC_ARG_MAX	Long. Max des arguments passes à exec
CHILD_MAX	_SC_CHILD_MAX	Nb max de processus par utilisateur
CLK_TCK	_SC_CLK_TCK	Nb de ticks d'horloge par seconde
NGROUP_MAX	_SC_NGROUP_MAX	Nb max de groupes de processus par processus
OPEN_MAX	_SC_OPEN_MAX	Nb max de fichiers ouverts par processus
PASS_MAX	_SC_PASS_MAX	Nb Max de caractères dans un mot de passe
POSIX_VERSION	_SC_POSIX_VERSION	Indique la version POSIX supportée
NAME_MAX	_PC_NAME_MAX	Nb max de caractères dans un nom de fichiers
PATH_MAX	_PC_PATH_MAX	Nb max de caractères dans un nom de chemin relatif
PIPE_BUF	_PC_PIPE_BUF	Nb max de caractères écrits de faÃ§on atomique dans un tu

Accès à la bases de données systèmes

```
#include <pwd.h>
struct passwd *getpwuid(uid_t numero);
struct passwd *getpwnam(const char *nom);
struct passwd {
   char *pw_name;
   char *pw_passwd;
   uid_t pw_uid;
   gid_t pw_gid;
   char pw_gecos;
   char *pw_dir;
   char *pw_shell; };
```