R1.04
Introduction aux systèmes d'exploitation et à leur fonctionnement

Jean-François Remm

IUT Nantes département Informatique

2021–2022

Objectif (Extrait du PN)

L'objectif de cette ressource est de comprendre le rôle, les composants et le fonctionnement d'un système d'exploitation. Les savoirs de référence suivants devront être étudiés :

- Caractéristiques et types de systèmes d'exploitations
- Langage de commande (commandes de base, introduction à la programmation des scripts)
- Gestion des processus (création, destruction, suivi, etc.)
- Gestion des fichiers (types, droits, etc.)
- Gestion des utilisateurs (caractéristiques, création, suppression, etc.)
- Principes de l'installation et de la configuration d'un système : notion de noyau, de pilotes, de fichiers de configuration, boot système...

Cette ressource permettra de découvrir les principes d'un système d'exploitation, leur mode de fonctionnement et les différents types existants. Elle contribuera à comprendre comment installer un système sur une machine et à le personnaliser en développant des fonctions simples facilitant la configuration et le paramétrage.

Organisation du module

Volume horaire : 20H

- 2H40 CM + 7 * 2H40 TP

Modalité d'évaluation :

- 1 test coeff. 1
- 1 ds coeff. 1

Équipe pédagogique

- CM : Jean-François Remm
- TD : Jean-François Berdjugin, Sébastien Faucou, Erwan Helleu et Jean-François Remm

Remerciements – Contributeurs – Sources

- Loïg Jezequel DUT Info M1101 ISI
- Sébastien Faucou DUT Info M3101 PSE
- Pascal Anelli & cie : Maitriser le shell Bash (FUN MOOC)

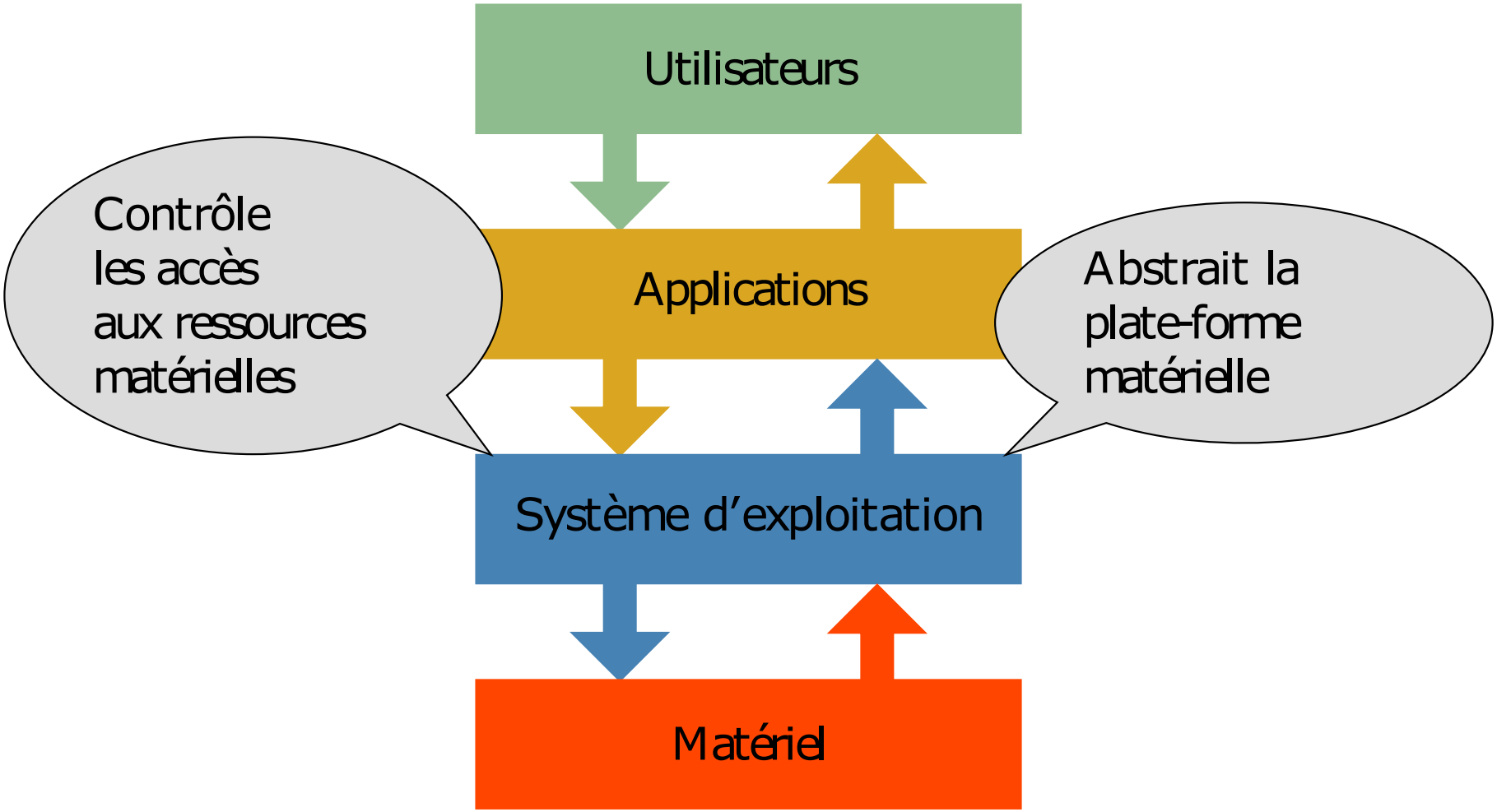
Système d'exploitation

noté SE ou OS = Operating System

- Qu'est ce qu'un système d'exploitation ?
  - premier programme chargé au démarrage
  - interface entre l'utilisateur et les ressources matérielles
- À quoi ça sert?

  abstraction présenter à l'utilisateur une machine virtuelle, plus conviviale

  gestion de ressources partage de la machine physique entre les applications


Utilisateurs
Contrôle
les accès
aux ressources
matérielles
Applications
Abstrait la
plate-forme
matérielle
Système d'exploitation
Matériel

Constituant d'un SE

SE =

- un noyau
- des services
- des programmes de base

Caractéristiques des SE

Eléments de différenciation :

- mono-utilisateur vs. multi-utilisateurs
- monotâche vs. multitâche
- centralisé vs. distribué
- ...

Type/Historique des SE

traitement par lots (*batch*) (années 50 à 60) tâches soumises jusqu'à terminaison

systèmes multi-programmés exécution simultanée de plusieurs programmes pour libérer le processeur en cas d'entrée/sortie $\rightarrow$ maximise utilisation processeur

temps partagé partage du temps de processeur entre les usagers (tranches de temps de durée fixée) $\rightarrow$ minimise temps de réponse

systèmes répartis (80 $\rightarrow$) abstraction de la localisation physique des ressources partagées au sein d'un réseau

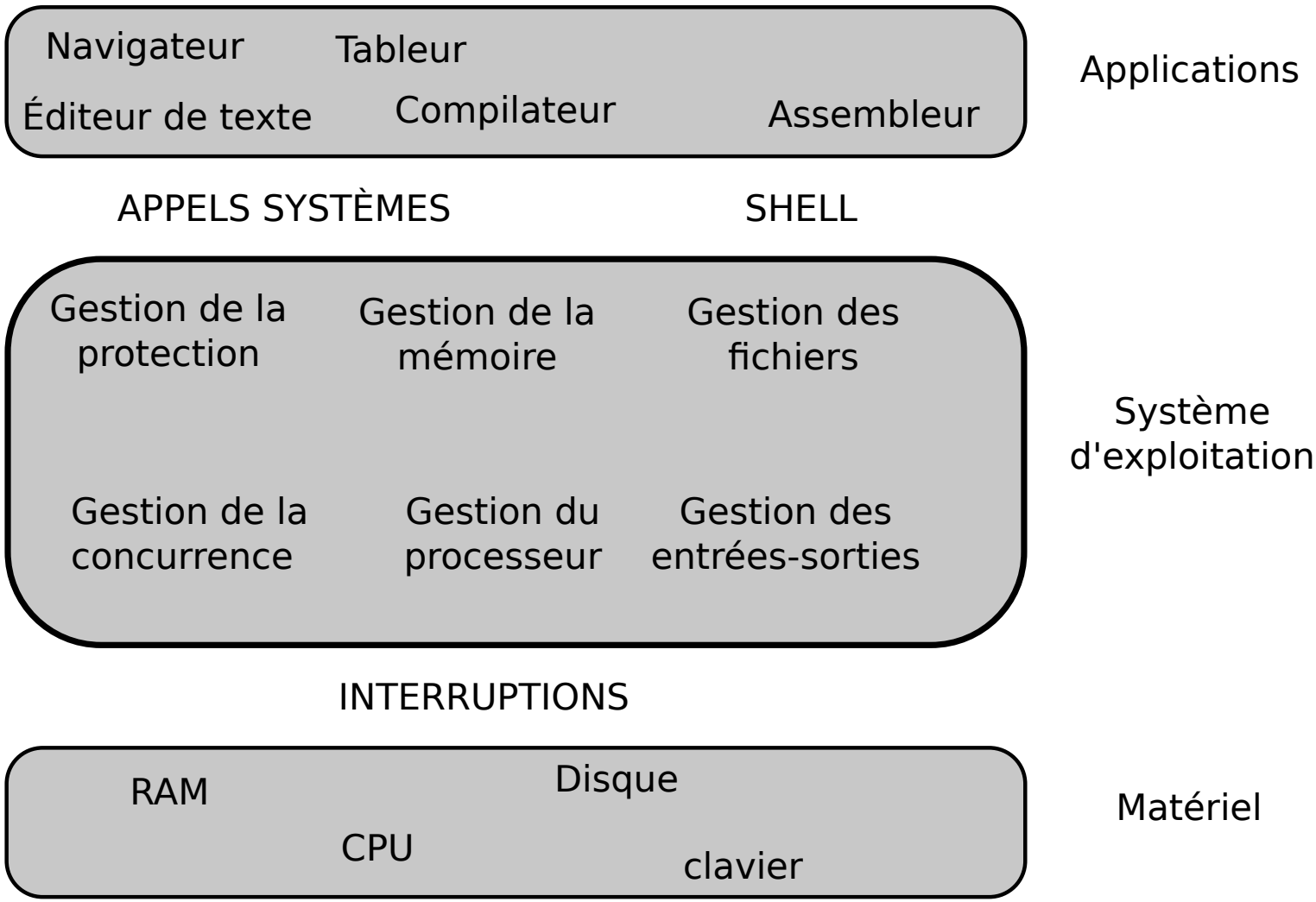
systèmes embarqués (90 $\rightarrow$) adaptés à des contraintes : énergie, temps-réel, ressources limitées, coût

Noyau

Partie du système d'exploitation constituée d'un binaire résident en mémoire (chargée au démarrage du système) d'une bibliothèque de primitives de service et de routines d'interruptions. **Il arbitre l'accès aux ressources matérielles** :

- accès au CPU (service **ordonnancement**)
- accès à la RAM (service gestion de la mémoire)
- accès aux disques (service gestion des fichiers)
- accès aux périphériques (service gestion des e/s)

Organisation de SE



Architectures des noyaux de SE

Plusieurs architectures possibles :

- **noyau monolithique** : tout est intégré en un seul binaire
- **noyau modulaire** : chargement dynamique des modules selon les besoins
- **micronoyau ou client/serveur** : la majorité des services sont implantés sous la forme de binaires indépendants appelés serveurs
- etc.

Services

- chargement et exécution de programmes
- accès aux périphériques d'E/S (interfaces réseaux, disques, écran, caméra, clavier, souris, etc.)
- gestion des utilisateurs (authentification, permission)
- synchronisation et communications entre processus
- ...

Programme de base

- interprète de commande (shell)
- outils de développement (éditeur de texte, compilateur, éditeur de liens, debugger)
- manuel
- outils de surveillance des activités du système
- services de communication par réseaux

Processus

Définition (simplifiée) : un processus est **une** instance d'exécution d'un programme sur un processeur

- le **programme** est statique,
- le **processus** est dynamique,
- vision anthropomorphique du processus : il naît, vit (évolue), et meurt

Métaphore (Tanenbaum)

- un informaticien prépare un gâteau pour sa fille
  - recette de cuisine : programme
  - ingrédients : données
  - instruments: ressources nécessaires
  - informaticien : processeur
  - confection de ce gâteau : processus

Métaphore (Tanenbaum)

- un informaticien prépare un gâteau pour sa fille
  - recette de cuisine : programme
  - ingrédients : données
  - instruments: ressources nécessaires
  - informaticien : processeur
  - confection de ce gâteau : processus
- le fils de l'informaticien se fait piquer par une abeille
  - interruption
  - traitement
  - reprise du travail de cuisinier

Métaphore (Tanenbaum)

- un informaticien prépare un gâteau pour sa fille
  - recette de cuisine : programme
  - ingrédients : données
  - instruments: ressources nécessaires
  - informaticien : processeur
  - confection de ce gâteau : processus
- le fils de l'informaticien se fait piquer par une abeille
  - interruption
  - traitement
  - reprise du travail de cuisinier
- la fille de l'informaticien annonce de nouveaux invités
  - il faut faire 2 gâteaux
  - 2 processus distincts sur le même programme
  - séparation des données propres de chacun des processus
  - partage des ressources ?
  - si une seule fiche recette : programme réentrant

Processus

Définition abstraite : ensemble des activités résultant de l'exécution d'un programme

Du point de vue du SE : objet système qui regroupe l'ensemble des données relatives à un programme en cours d'exécution

- son **espace d'adressage**,
- son **contexte d'exécution** : valeurs des registres volatiles et registres d'exécutions (compteur ordinal, pointeur de pile et de cadre, etc.)
- son **descripteur** : identifiant du processus, permissions associées, liste des ressources occupées, etc.

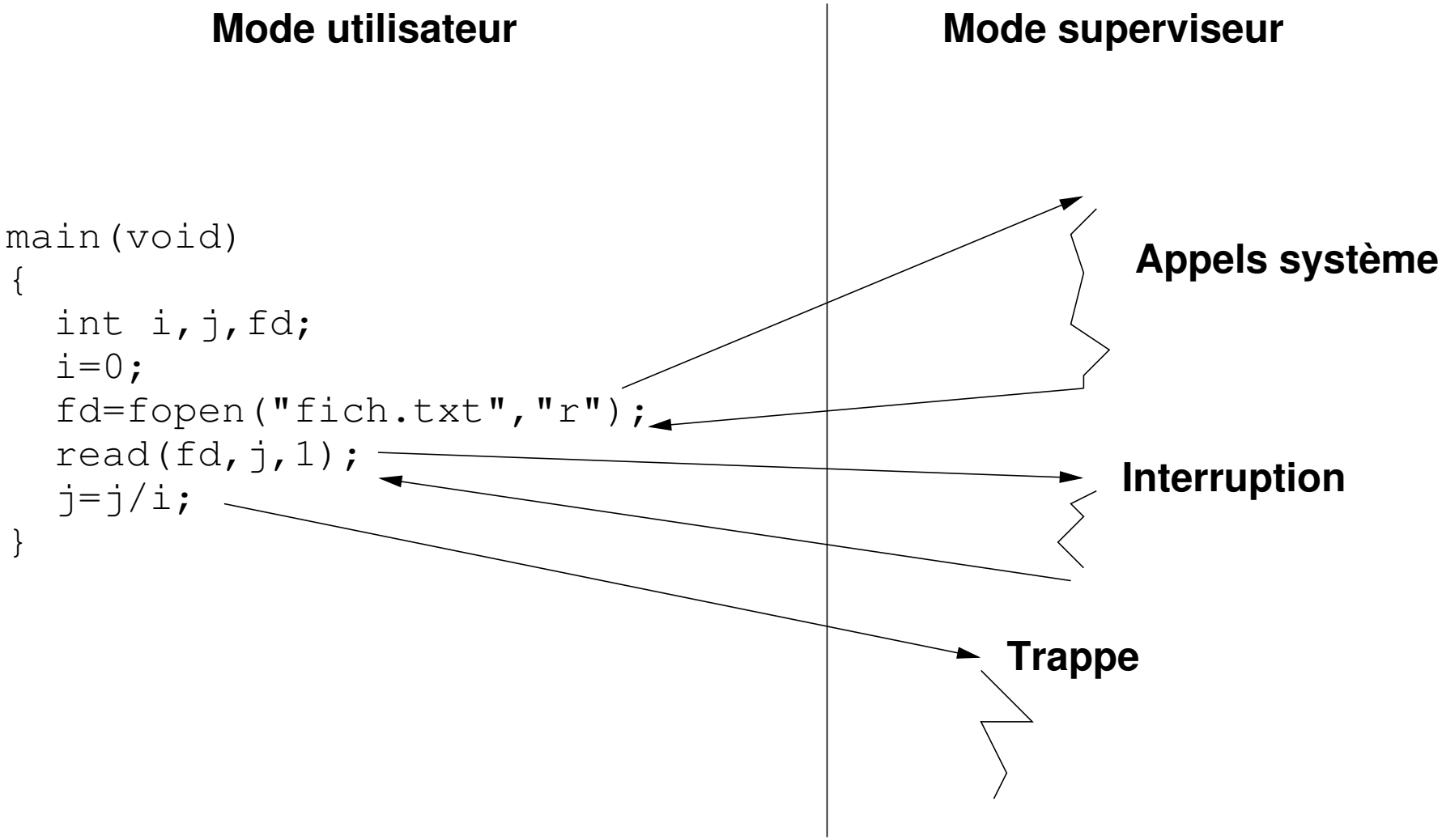
Le **contexte d'exécution** est géré par le noyau uniquement s'il est **multitâche**

Interactions processus / noyau

Un processus peut interagir avec le noyau en invocant une primitive de service, ou **appel système** :

1. l'appel système nécessite le plus souvent de basculer le processeur en **mode superviseur** (accès à toute la mémoire, à toutes les instructions machines)
2. l'exécution de l'appel système peut impliquer un appel à la procédure d'ordonnancement
   - **points de réordonnancement = appels systèmes** (éventuellement appelé par une routine d'interruption)
3. en fin d'appel système, le processeur est rebasculé en **mode utilisateur**

Traitement des primitives de service



Appel système

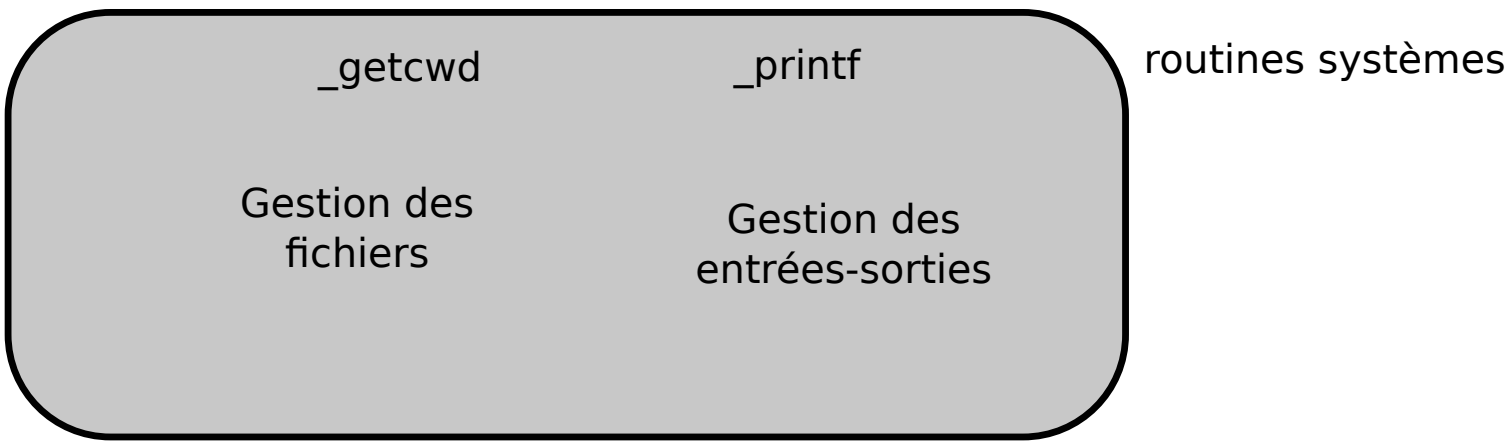
```
#include <unistd.h>
#include <stdio.h>

int main(void)
{
   char *repCour;
   repCour = getcwd(NULL,0);
   printf("%s\n",repCour);
   return(0);
}
```

APPELS SYSTÈMES

```
$ pwd
```

SHELL



routines systèmes

Noyau multitâche

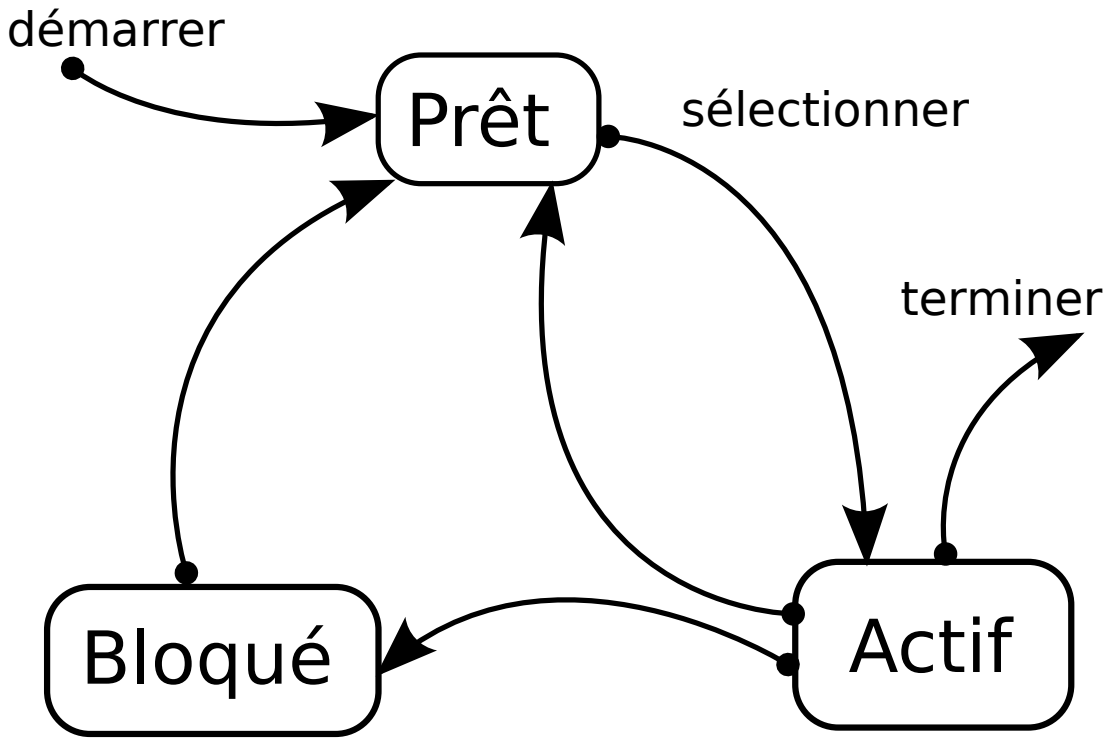
Noyau capable d'exécuter des **processus concurrents** :

- un processus, choisi par l'ordonnanceur, s'exécute
- un ensemble de **processus prêts attendent d'être choisi** par l'ordonnanceur
- d'autres processus peuvent être **en attente qu'une certaine condition se réalise** (par exemple qu'une donnée devienne disponible sur un flux d'entrée)

Politique d'ordonnancement

Politique mise en œuvre par le noyau du SE pour allouer le temps CPU.
N'a de sens que dans le cas d'un noyau **multitâche**

États d'un processus (simplifié)

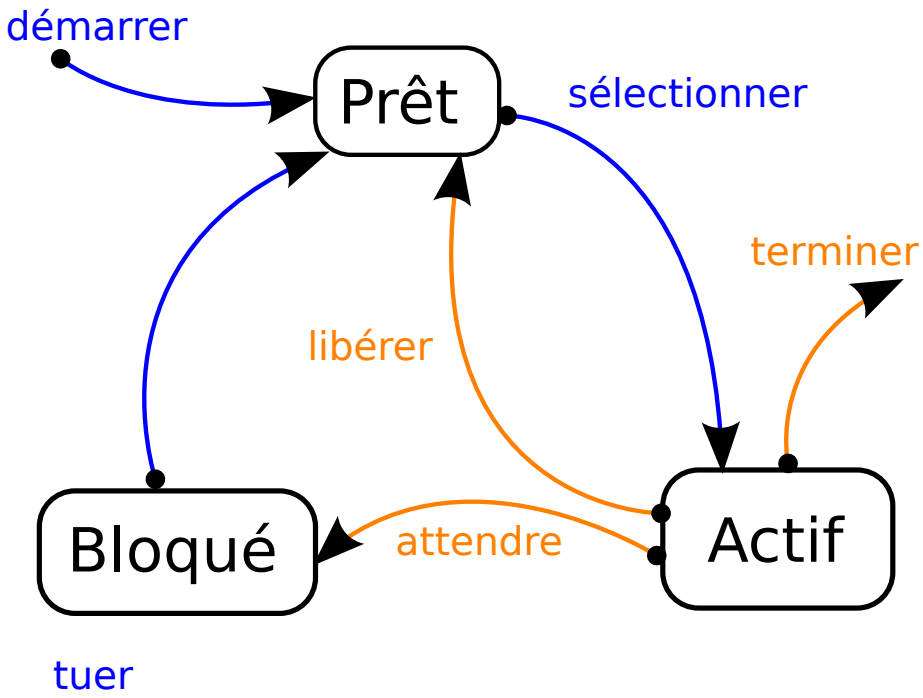


Avantages du multitâche

- **meilleure utilisation des ressources** : lorsqu'un processus est en attente de la fin d'une opération d'entrée/sortie, un autre peut utiliser le CPU
- exploitation du système par **plusieurs utilisateurs simultanément** : le noyau doit alors mettre en œuvre une politique d'ordonnancement en **temps partagé**
- **exécution simultanée de plusieurs programmes** : la politique d'ordonnancement mise en œuvre doit donner à l'utilisateur l'impression que tous les processus progressent simultanément

Multitâche coopératif

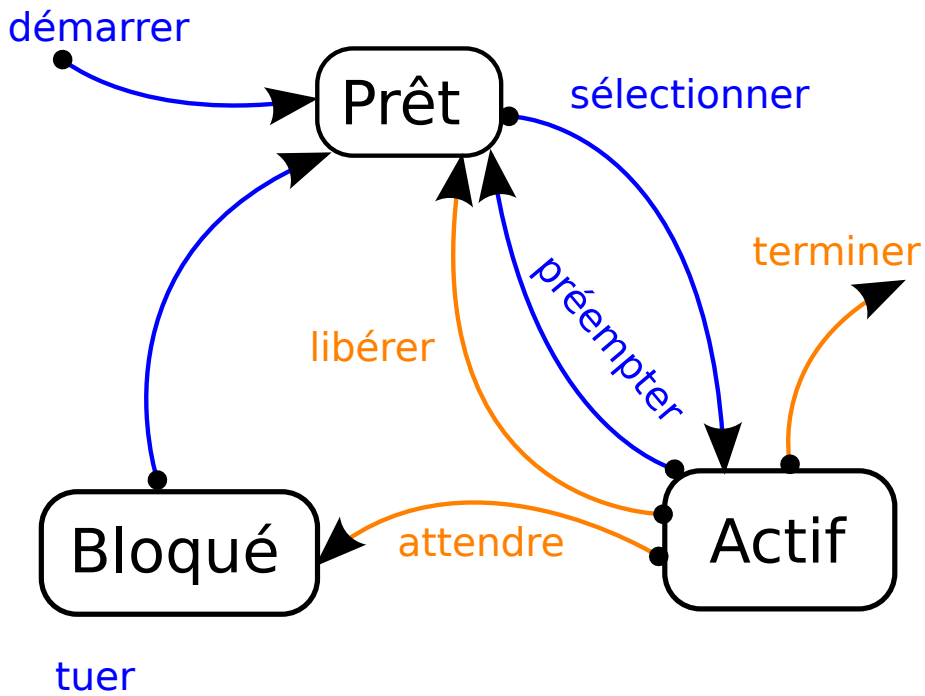
Système multitâche dans lequel le processus en cours d'exécution décide de lui-même quand il rend la main (via des primitives de service dédiées)



- transition contrôlée par un tiers (noyau, autre processus)
- transition contrôlée par le processus

Multitâche préemptif

Système multitâche dans lequel le noyau peut décider de **préempter** le processus en cours pour choisir un autre processus parmi les processus prêts



- transition contrôlée par un tiers (noyau, autre processus)
- transition contrôlée par le processus

Coopératif vs. préemptif

Coopératif

- gestion simplifiée des ressources partagées : les accès concurrents sont évités en ne rendant pas la main lors d'une section critique
  - **uniquement dans le cas monoprocesseur monocoeur**
- noyau simple : le surcoût induit par l'ordonnanceur est minimal

Préemptif

- le programmeur n'a pas à se soucier de passer la main
- plus robuste : un processus défaillant ne paralyse (normalement) pas le système
- plus réactif : si un événement urgent est signalé, il est possible de donner immédiatement la main au processus qui était en attente

Commutation de contexte

Pour mettre en œuvre un système multitâche, il faut que le noyau puisse :

- sauvegarder l'état du processus en cours d'exécution
- puis charger l'état du processus nouvellement choisi pour reprendre l'exécution **là où elle avait été arrêtée**

- L'état d'un processus est appelé **contexte**. Il est constitué :
  - du contenu des registres volatiles
  - du contenu des registres d'exécution (compteur ordinal, pointeur de pile, de cadre, registres d'état, etc.)
- Le cycle *sauvegarde du contexte courant / chargement d'un nouveau contexte* est appelé **changement de contexte**

Schéma général du service d'ordonnancement

```
procédure ordonnancer( _es  prêts : ensemble processus,
                       _es  en_cours : processus)
début
   verrouiller_ordonnanceur()
   sauver_contexte(en_cours)
   prêts := prêts + en_cours
   en_cours := choisir(prêts)
   charger_contexte(en_cours)
   déverrouiller_ordonnanceur()
fin
```

- **Coopératif** : seul le processus en cours peut appeler cette procédure
- **Préemptif** : le noyau peut appeler cette procédure suite à une évolution de l'image qu'il a du système (évolution via une interruption ou un appel système)

Objectifs d'une politique d'ordonnancement

- Attribuer une **ressource** à des **processus** en satisfaisant certains critères
- ressources : CPU, disque, mémoire,...
- critères : équité, priorité, efficacité, ...
- métriques :
  - processeur : maximiser le taux d'utilisation du processeur, minimiser le temps de réponse maximal, minimiser le temps de traitement moyen,...
  - disque : minimisation des déplacements du bras de lecture
  - mémoire : minimiser le swap
  - imprimante : garantir l'ordre sans mélange
  - ...

Objectifs de l'ordonnancement du CPU

Le choix d'une politique d'ordonnancement est **guidé par les besoins des applications**

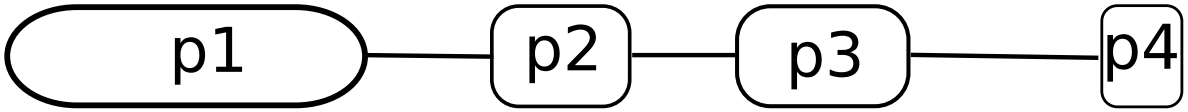
- système de traitement par lots : ordonnancement premier arrivé premier servi
- système multi-utilisateurs : partage équitable du temps entre utilisateurs
- station de travail : partage du temps pour donner l'illusion du parallélisme / permettre le confort d'utilisation en étant réactif aux interactions
- système temps réel : allocation aux traitements les plus urgents
- etc.

Dans les systèmes modernes, on trouve souvent des politiques hybrides

Métriques

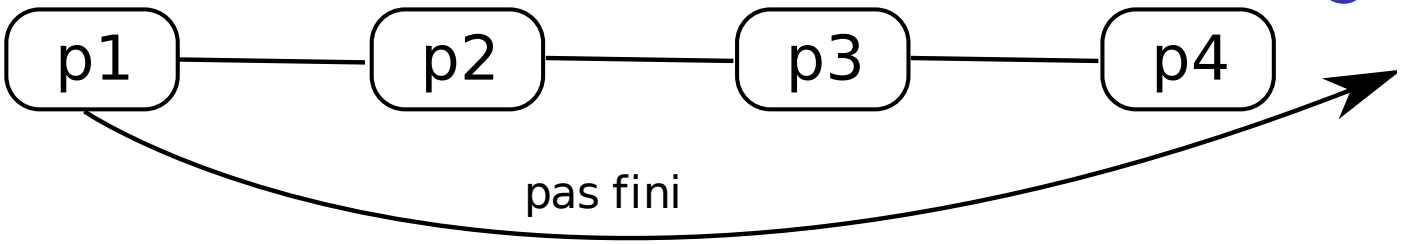
- temps d'attente : temps passé dans l'état prêt par le processus
- temps de rotation : intervalle de temps entre arrivé et fin d'une demande
- temps de réponse : intervalle de temps entre arrivé d'une demande et première réponse $\rightarrow$ périphérique de sortie

La stratégie FIFO



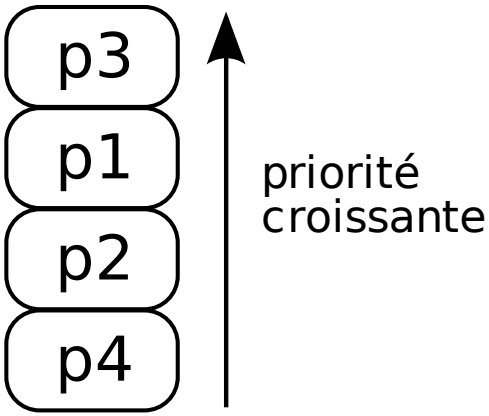
- premier arrivé, premier servi
- Objectif : simplicité, éviter les famines
- Principes
  - Les processus sont dant une file d'attente.
  - Chaque fois qu'un processus se termine, le noyau appelle la procédure d'ordonnancement
  - Chaque fois qu'un processus se met en attente, le noyau appelle la procédure d'ordonnancement
  - Difficulté : équité ?, priorité
  - Pénalise les processus à temps bref d'exécution

La stratégie RR



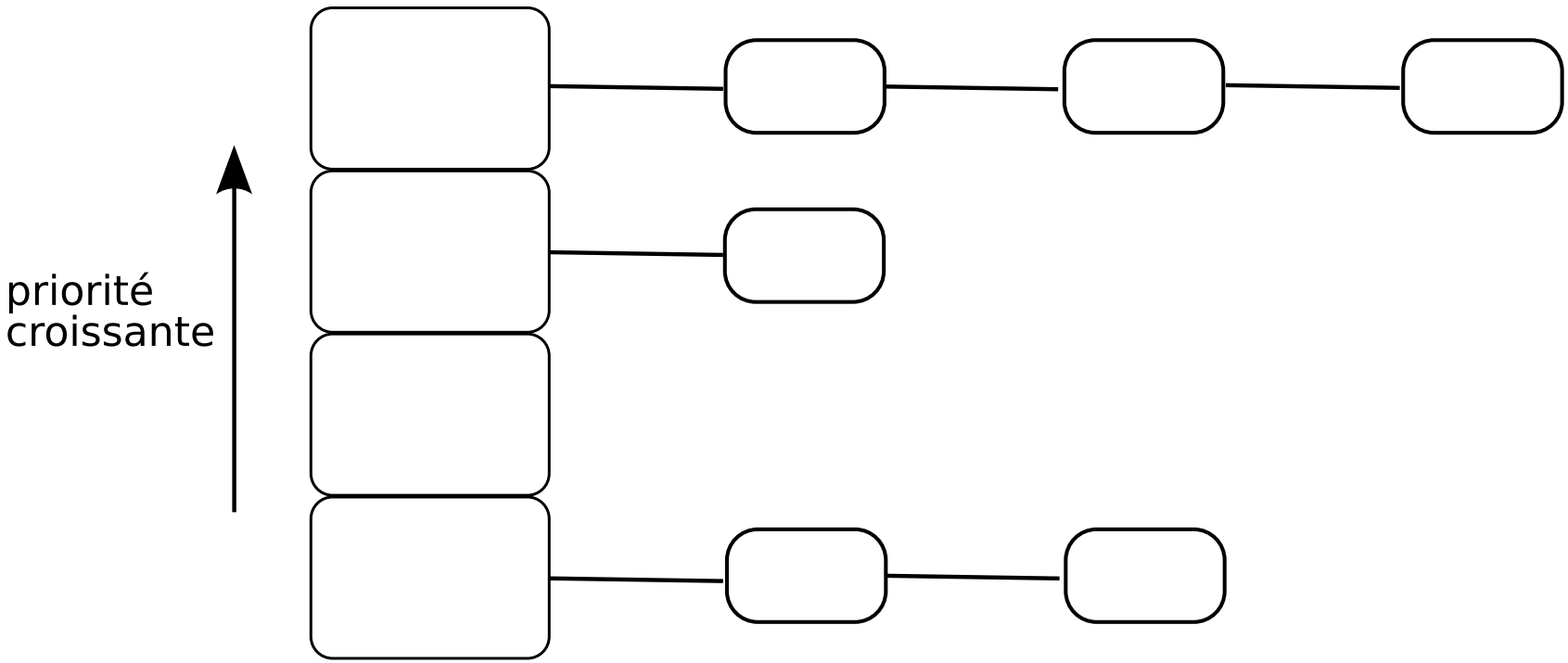
- RR = *Round-Robin* = tourniquet
- Objectif : éviter les famines
- Principes
  - on gère une file d'attente
  - un nouveau processus est placé en fin de file
  - un **timer** est programmé pour **émettre périodiquement une interruption matérielle**. Sa période est appelé **quantum**
  - à chaque fin de quantum, le noyau appelle la procédure d'ordonnancement et le processus courant est placé en fin de file
  - chaque fois qu'un processus se termine ou se met en attente, le noyau appelle la procédure d'ordonnancement
  - Difficulté : reglage du quantum
    - court : ☺interactivité ☹changement fréquent (temps passé à changer de contexte) ,
    - long : ☹temps de réponse, ☺efficacité.

La stratégie par priorité



- chaque processus a une priorité
- Objectif : favoriser les choses importantes
- Principes
  - les plus prioritaire d'abord
  - priorité dynamique ou statique
  - avec ou sans quatum

En pratique

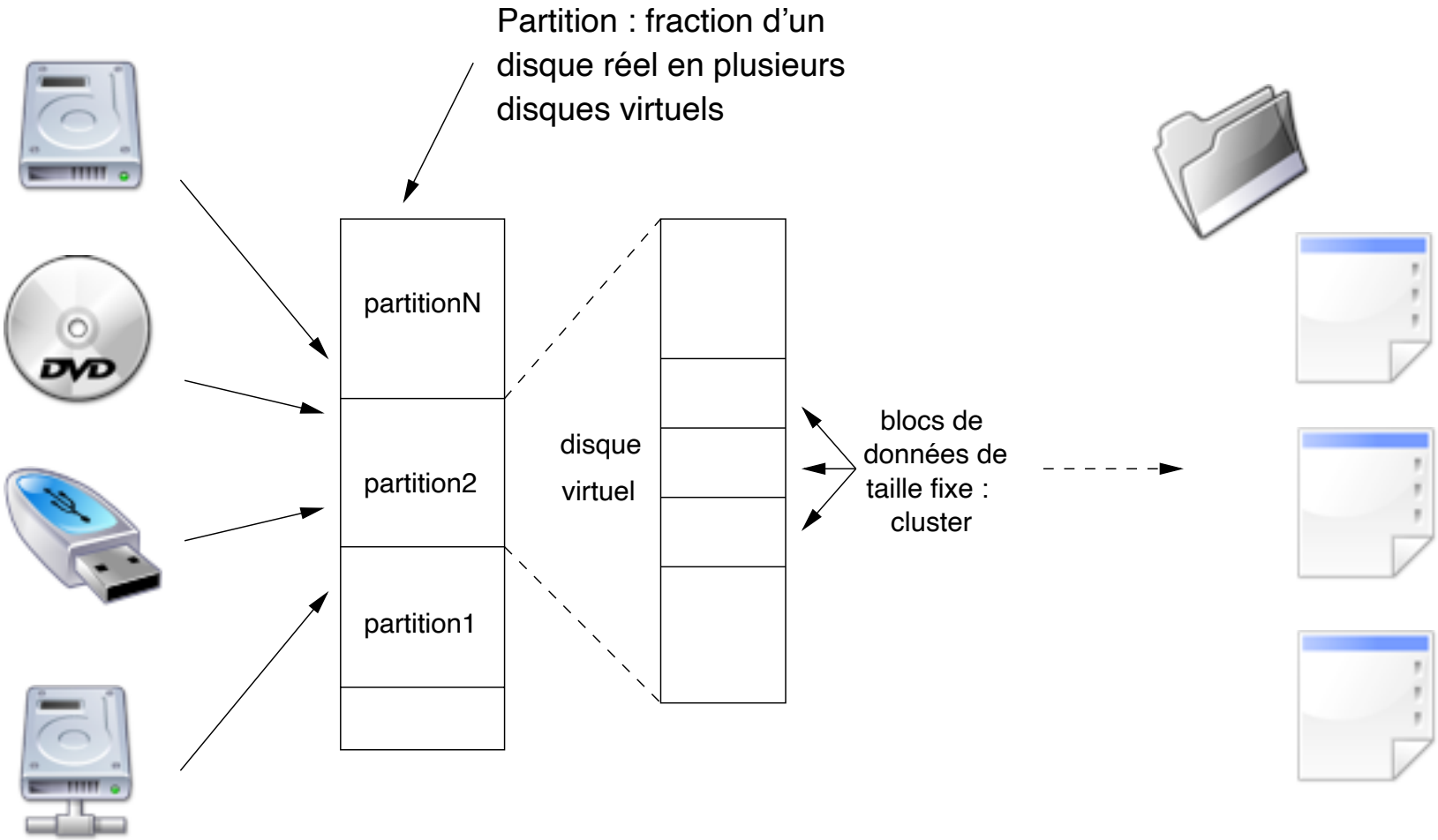


gestion par priorité et tourniquet sur chaque niveau de priorité.

Système de gestion de fichiers

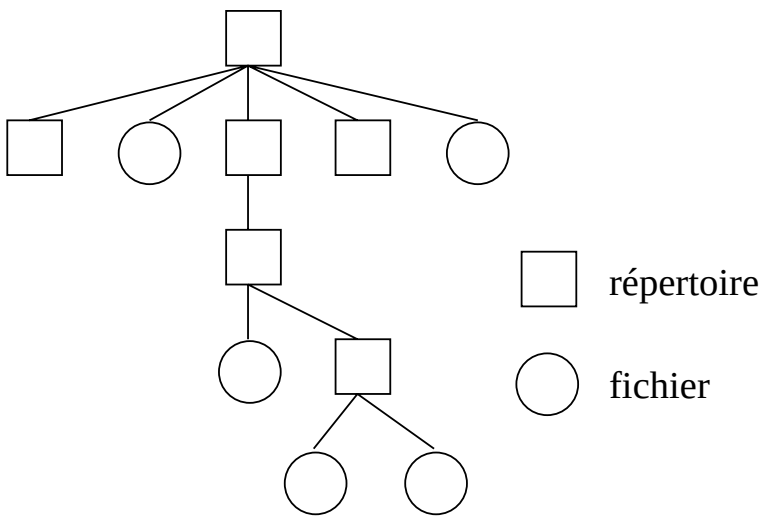
- la partie du SE qui gère accès, nommages, structures, types, ... des fichiers est le **système de gestion de fichiers**
- ce système présente deux aspects :
  1. aspect utilisateur : nommage, création, ouverture, fermeture, ... des fichiers
  2. aspect physique : faire correspondre un fichier à une implantation sur un support physique, si possible avec un niveau d'abstraction suffisant.

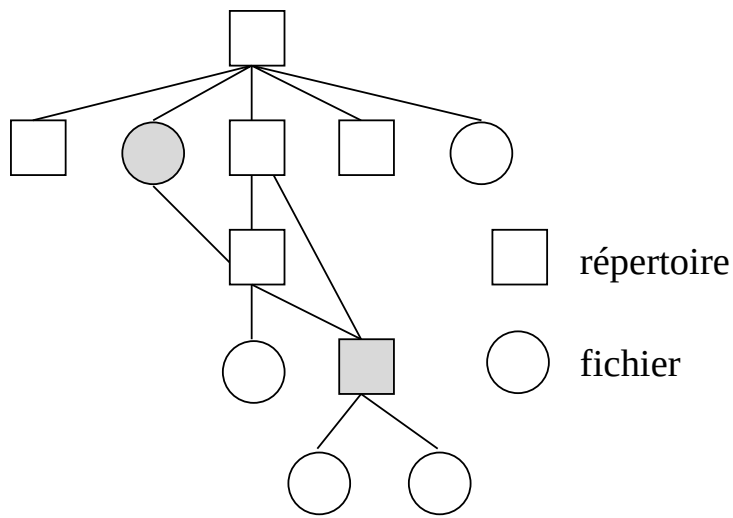
Système de gestion de fichiers



Organisation des fichiers

- structure arborescente : qui implique la notion de **répertoire** et la présence d'une (ou plusieurs) **racine**
- graphe : comme une structure arborescente, mais un objet du SGF peut-être visible à plusieurs endroits → liens unix, raccourcis windows

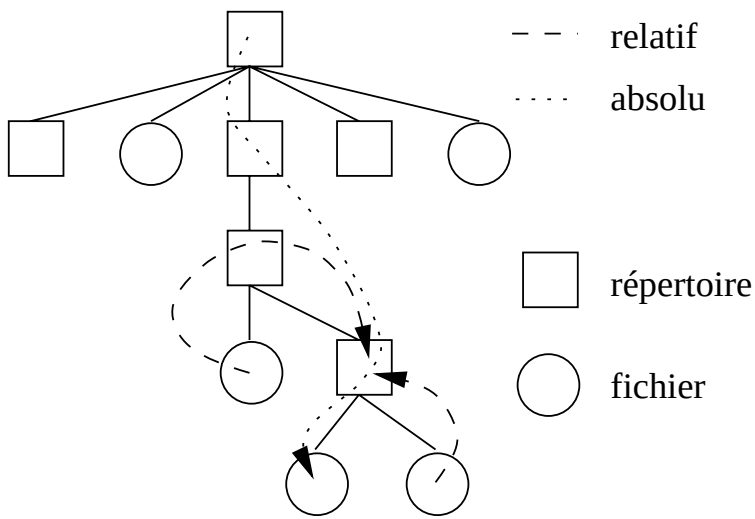




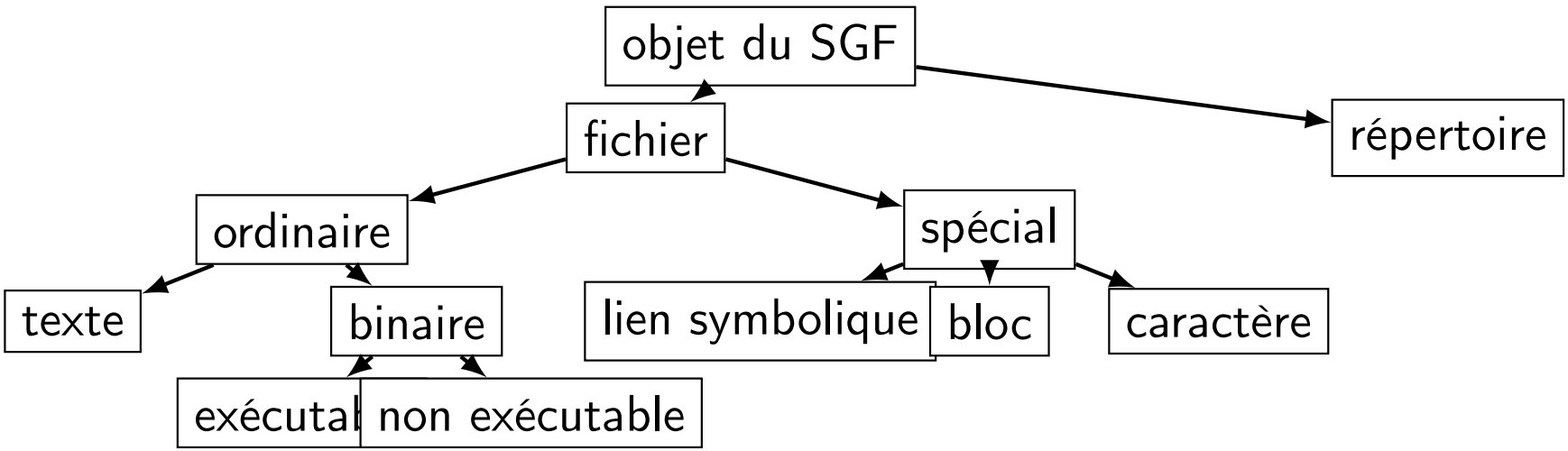
Références des fichiers

Dans le cas d'une organisation arborescente ou en graphe, la référence d'un fichier peut se faire de deux manières :

- une référence **absolue** où chaque fichier est désigné par le chemin la racine jusqu'au fichier. Ex : `/home/remm/IUT/cmSys.pdf` (unix) ou `\Program Files\notepad.exe` (windows)
- une référence **relative** – qui nécessite le concept de **répertoire courant** – où chaque fichier est désigné par le chemin depuis répertoire courant jusqu'au fichier. Ex : `../IUT/cmSys.pdf` (unix) ou `..\..\Program Files\notepad.exe` (windows)



Typologie des objets du SGF



Protection des fichiers

Comment protéger les fichiers ?

- protéger chaque fichier par mot de passe → trop lourd à mettre en oeuvre
- gérer une politique de droits par utilisateur, qu'on place dans des groupes → les utilisateurs du système sont classés en trois catégories :
  1. le **propriétaire** du fichier
  2. les **membres du groupe** du propriétaire
  3. les **autres**
- gérer plus finement des ACL (*Access Control List*): les acl sont une liste de droits pour autant d'utilisateurs et/ou de groupes souhaités.

Droits

Trois opérations élémentaires sont contrôlées par le système :

1. **lecture** (`r` pour *Read*, lire) : sur un **fichier**, autorise la **lecture** de son contenu ; sur un **répertoire**, autorise le **listage** des fichiers qui y sont contenus.
2. **écriture** (`w` pour *Write*, écrire) : sur un **fichier**, autorise la **modification** de son contenu : sur un **répertoire** autorise l'**ajout** ou la **suppression** de fichiers dans ce répertoire (même sans en être propriétaire).
3. **exécution** (`x` pour *eXecute*, exécuter) : sur un **fichier**, autorise son **exécution** ; sur un **répertoire**, autorise sa **traversée**

Utilisateurs

- les utilisateurs sont répertoriés et authentifiés
- cette authentification peut être locale ou via un service d'annuaire distant :
  - Domaine windows (*Active Directory*)
  - NIS, NIS+, LDAP (*Lightweight Directory Access Protocol*)
- il existe souvent un utilisateur privilégié avec des droits étendus

Modes d'accès aux fichiers

- Le mode d'accès à un fichier définit la manière dont les données sont rendues accesibles.
- la disponibilité des modes d'accès dépend du SE
- Deux modes d'accès classiques :
  1. accès séquentiel : parcours depuis le début du fichier pour lire l'enregistrement souhaité
  2. accès aléatoire ou direct : on accède directement à l'enregistrement souhaité, éventuellement par un décalage par rapport au début du fichier

Attributs ou métadonnées des fichiers

Un fichier, c'est au minimum un nom et des données, mais le sgf peut gérer en plus :

- protection (qui accède dans quelle mesure)
- propriétaire et groupe propriétaire
- taille du fichier
- type (binaire/exécutable, voir plus)
- dates (création, dernier accès, dernière modification...)
- ...

Opérations sur les fichiers

- create : création du fichier sans données
- delete : effacement
- open : ouverture
- close : fermeture
- read : lecture
- write : écriture
- append : ajout
- seek : modification de la position dans un fichier à accès aléatoire
- get attributes : lecture d'attributs de fichiers
- set attributes : écriture d'attributs de fichiers
- rename : changer le nom d'un fichier

Opérations fournies soit :

- depuis un langage de programmation : API système
- par l'interpréteur de commande (encore lui)

Opérations sur les répertoires

- create : création d'un répertoire
- delete : effacement d'un répertoire
- opendir : ouverture du contenu d'un répertoire
- closedir : fermeture du contenu d'un répertoire
- readdir : lecture d'une entrée du répretoire
- rename : changer le nom d'un répertoire
- link : rajouter une entrée dans un répertoire
- unlink : supprimer une entrée d'un répertoire

Opérations fournies soit : iz

- depuis un langage de programmation : API système
- par l'interpréteur de commande (encore lui)

Exemple en Go

Différents système de gestions de fichiers

- compatibles seulement avec certains SE
- diffèrent dans le manière d'organiser les répertoires, l'allocation des blocs de données, la gestion des protection, ...
- on peut lister plusieurs autres fonctionnalités :
  1. journalisation
  2. distribution
  3. compression
  4. cryptage
  5. ...

Exemples de SGF

- Sous windows :
  - FAT16, FAT32, VFAT
  - NTFS
- Unix :
  - ext2
  - ext3, ext4, ReiserFS
  - swap (cas particulier)
- Mac OS
  - HFS (Hierarchical File System)
  - HFS+, HFSX
  - APFS
- Réseau
  - SMB (Server Message Block), CIFS
  - NFS

SGF en chiffres

Nom	journa-lisé	Taille nom de fichier	gestion de droit	ACL	compres-sion	cryptage	Tai d'u
Ext2		255 car.	X	(X)		X	
Ext3	X	255 car.	X	(X)		X	
FAT16		8.3 car.					
FAT32		255 car.					
NTFS	X	255 car.	X	X	X	X	
HFS		31 car.	X		X		
HFS+/HFSX	X	255 car.	X	X		(X)	