## Légende

SE : Système d’exploitation. E/S : Exception système. DMA : Direct Memory Access. CPU : Central Processing Unit. PC : Program Counter. Pg : Programme. Pu : Processus. PCB : Pu control block. ORD : Ordonnanceur. PID : Pu Id. IPC : Inter-mediary Pu Communications.

*AVs* : Avantages – *DAVs* : Désavantages – Mots surlignés : Données manquantes.

Introduction

## Interruptions

Un SE est guidé par les interruptions (*trap* ou E/S) – Détermine quel type d’interruption par l’usage de polling ou de vecteurs d’interruption.

Après une E/S, le système retourne seulement quand la E/S est complétée : boucle d’attente active, une seule E/S à la fois – Retourne au Pg sans attendre la fin de l’E/S (requêtes au SE pour la suite, attend le **OK** du SE).

Gestion de la mémoire : stockage temporaire des données, mise en cache, utilisation spool.

## Stockage

Mémoire centrale : seul stockage auquel le CPU peut accéder directement (accès aléatoires, volatile) – Mémoire de masse : extension de la mémoire centrale, grande capacité, pas volatile – SSD – Caching : Accès aux données récurrentes rapidement en les stockant dans un espace dédié, coût moindre, complexité gestion – Registres.

## Architecture des systèmes

Un CPU – Multiprocesseurs (multi-core): plus haute performance, moindre coût, meilleure fiabilité. Deux types : asymétrique (un CPU principal gère plus les autres), symétrique (le plus populaire) – Systèmes en groupes (clusters) : comme multiprocesseurs, mais là, nous utilisons une architecture multisystèmes. Plus efficace, tolérance aux pannes, etc.

## Structure des SE

Multiprogrammation : organise le travail que le CPU ait toujours quelque chose à faire, sous-ensemble des tâches gardé en mémoire, choix d’exécution des tâches, parallélisme. Multitasking : plusieurs tâches prêtes en même temps. Interblocage, mémoire virtuelle (tâches partiellement en mémoire), swapping.

## Transition au mode noyau

Si E/S, passe en mode noyau. Utilisé afin d’éviter les boucles infinies. Repasse en mode utilisateur lorsque l’E/S a été gérée. Fonctions noyau : clear memory, turn off interrups, switch from user to kernel, set value of timer...

## Gestion des processus (intro)

Un programme est une entité passive. Un processus est une entité active.

Single-thread : PC qui indique l’adresse de la prochaine instruction. Multi-threads : Plusieurs PC.

SE est responsable de créer, détruire, suspendre, réveiller des processus et doit fournir des mécanismes de synchronisation et de communication entre les processus.

## Gestion de la mémoire (et +)

SE doit optimiser l’usage du CPU et le temps de réponse (gestion des données efficace) – SE monitore les accès mémoire (trace, allouer et désallouer les espaces mémoire) – SE fournit une vue abstraite des données (répertoires, fichiers, etc.) + gestion de cette vue. SE gère la sécurité (permissions).

## Services offerts par le SE

Exécution d’un Pg – I/O Opérations – Manipulation du système de fichiers – Communications entre différents Pu– Détection d’erreurs – Allocation de ressources – Comptabilité (garder une trace des opéra-tions effectuées) – Protection et sécurité.

## Types d’interfaces

CLI (command line) – GUI (graphical user interface) – Appel système (CLI) : Pro-grammes avec paramètres disponibles à travers une API. Beaucoup de types de programmes – Programme système : Création, affichage, modification et suppre-ssion de fichiers (mkdir, vim, etc.). Compila-teurs, assembleurs, interprètes. Services background. Applications.

## Différentes structures du SE

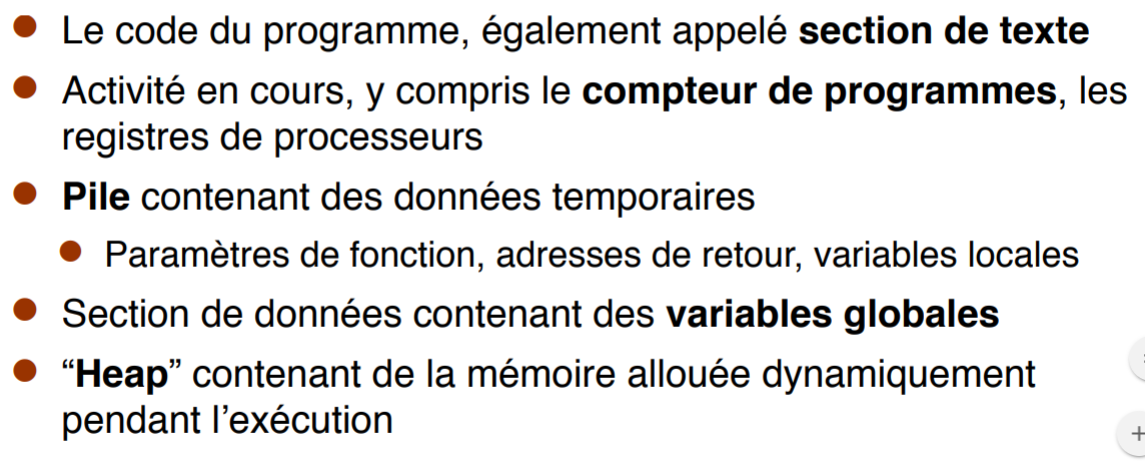
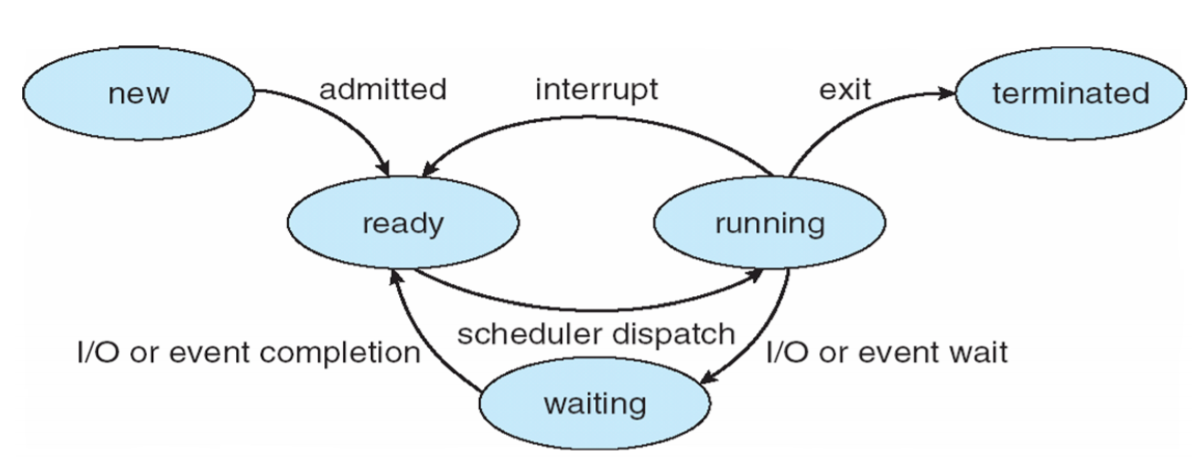
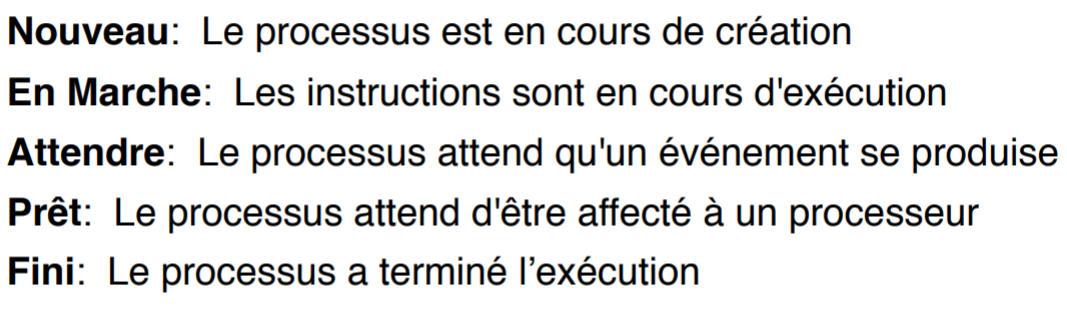
MS-DOS : Simple, non divisé en modules, fonctionnalités pas bien séparées – Unix : deux parties [Programmes systèmes, Noyaux : entre système et matériel physique, fournit le système de fichier, la gestion de la mémoire, un grand nombre de fonctions, très peu de frais généraux dans l’interface d’appel système].

Approche en couches : Couche 0 (matériel), couche N (UI). Toutes les couches utilisent que des fonctionnalités inférieures – Microkernel: Commu-nications au niveau du noyau ET des modules de l’utilisateur. Mac OS X. *AVs* : Plus facile d’étendre un micro-noyau, plus facile de porter le SE à de nouvelles machines, plus fiable (moins de code s’exécute en mode noyau), plus sécurisé. *DAVs* : Espace utilisateur doit être plus performant pour permettre la communication entre modules – Modules : Chaque composant est séparé et peut être chargé (au besoin) dans le noyau (Linux, solaris, etc) – Systèmes Hybrides : SE modernes ne sont pas « purs », combine plusieurs approches.

## Débogage du système exploitation

Les SE génèrent des fichiers journaux contenant des informations d'erreur. Kernighan’s Law : Le débogage est 2x plus difficile que coder, alors si tu codes à ton max, tu n’es pas assez intelligent pour deboger ton code

Processus

Pu Job tâche. Pg (passif), Pu (actif). Pg devient Pu lorsque chargé en mémoire.  
   
  
  
Contenu du PCB : État du Pu, PC, Registres du Pu, Informations Pu (priorité, mémoire allouée, temps écoulé, fichiers ouverts, CPU utilisé, etc)

## Ordonnanceurs

Les ORD choisissent quand exécuter quel Pu sur quel CPU. Utilisent des queues : job queue (tous les Pu), ready queue (les Pu), device queues (les Pu qui attendent un périphérique)

Type de ORD : Long terme – Sélectionne quels Pu doivent être placés dans la file d’attente, invoqué rarement (lent), contrôle le nombre de Pu en mémoire. Court terme – Gère les Pu, planificateur des tâches, invoqué fréquemment. Moyen terme – Contrôle le degré de multiprog (swapping).

Types de Pu : I/O-bound – Beaucoup de rafales du CPU (plus de E/S que des calculs). CPU-bound – Quelques rafales (beaucoup de calculs).

## Gestion des Pu

Changement de Contexte : SE sauvegarde l’état du Pu courant et remplace celui en chargeant l’état du nouveau Pu. Contexte = PCB. Le temps de charge-ment doit être minimal – Création des Pu : Les Pu parents créent des Pu enfants. Chaque Pu a un PID. Options de partage de ressources : Parents et enfants partagent les mêmes ressources, enfants partagent un sous-ensemble de ressources. Aucun partage. Options d’exécution : Le parent et les enfants s’exécutent simultanément, parent attend que les enfants se terminent. Possibilités : L’en-fant est une copie du parent ou un nouveau Pu. – Terminaison des Pu : 4 possibilités – *(1)* Le Pu enfant termine via **exit()** et le parent reçoit les données via **wait()**. *(2)* Le parent termine les Pu enfants avec **abort()**. *(3)* Le Pu enfant termine pendant qu’aucun parent n’attend : **Zombie**. *(4)* Le parent termine, alors les enfants deviennent des Pu **orphelins**.

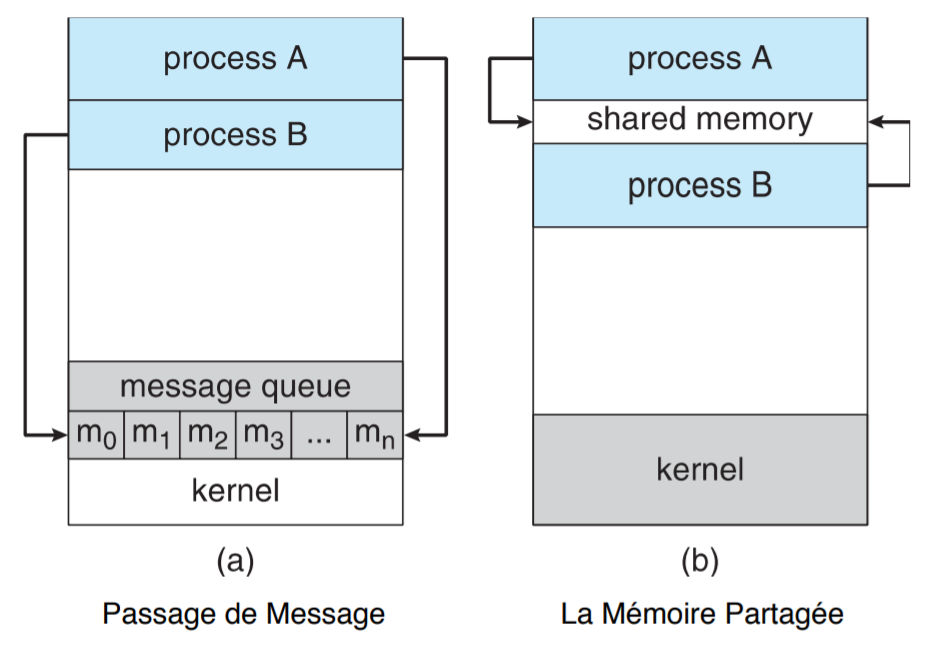
Processus (suite)

## Méthodes

fork() : Duplique le processus courant. Cette méthode retourne *0* si le Pu courant est l’enfant, *-1* s’il y a une erreur et un *chiffre quelconque* si c’est le parent (Pu d’origine).

exec\*() : Exécute un programme (avec paramètres)  
wait() : Attend après un Pu enfant.

## Intermediary Pu Communications (IPC)

Shared memory: unbounded-buffer (illimité), bounded-buffer (taille fixe). Message passing: send(TargetPu msg), receive(targetPu, msg). Un lien par pairs de Pu. Il est aussi possible de communiquer indirectement en transmettant des messages à des ports utilisés par des Pu. Les messages peuvent être (a)synchrones.

## Communications client-serveur

Socket : Le socket u.v.x.y:z fait référence au port z sur la machine u.v.x.y. UDP/TCP/Multiclast – Remote Procedure Calls (RPC) : Communication à distance. Les messages doivent être livrés exactement une fois (solution : timestamp, client attend accusé de réception). Comment le client peut savoir les numéros des ports? (solution : Le SE emploi un matchmaker) – Pipes (tuyaux): Permet la communication entre 2 Pu. (Uni/Bi)directionnel (write-end, read-end). Les tuyaux ordinaires sont unidirectionel, il faut donc 2 tuyaux pour aller dans les 2 sens.

## [\*\*Exercices TP\*\*]

Client-serveur VS P2P : P2P, chaque station peut être le client ou le serveur – Que partagent les Pu lors d’un fork? Pile et tas (readonly), puis mémoire partagée.

Threads

as