Architecture et Système 2

Partie 3 : Les processus
Ordonnancement des processus

Jalil BOUKHOBZA

UBO / Lab-STICC

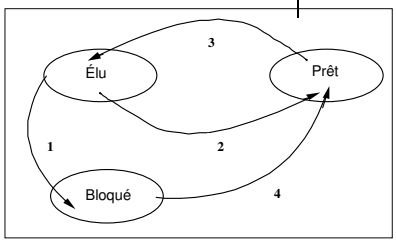
Email : boukhobza@univ-brest.fr

Partie 3 : Ordonnancement des processus

1. Ordonnancement et état d'un processus
2. Les métriques et objectifs de l'ordonnancement
3. Quelques algorithmes d'ordonnancement

Diagramme simplifié des états et des transitions

- **Trois états** possibles :
  - Actif en mémoire centrale (ELU)
  - Suspendu en attente d'exécution (PRÊT).
  - Bloqué en attente de ressource (BLOQUE).



- **Quatre transitions**
  1. Le processus se bloque en attente de données.
  2. L'ordonnanceur interrompt le processus courant.
  3. L'ordonnanceur choisit un nouveau processus.
  4. Les données deviennent disponibles.

Tâches de l'ordonnanceur

- L'ordonnanceur est le composant de l'OS qui détermine l'ordre et la durée des tâches qui s'exécutent sur le CPU.
- L'ordonnanceur dicte l'état dans lequel doivent se trouver les tâches.
- Il charge et décharge le bloc de contrôle de la tâche
- Certains OS utilisent un processus séparé pour allouer le CPU au processus nouvellement sélectionné, il s'appelle le **dispatcheur**.

Commutation des processus

- C'est le rôle de **l'ordonnanceur** contenu de piloter la commutation des processus:
  - Sauvegarde du contexte de l'unité centrale (ou mot d'état) du processus courant dans le contexte du processus en cours.
  - Sauvegarde du contexte du processus courant.
  - Mise en place d'un nouveau contexte de l'unité centrale permettant le traitement de l'interruption au travers du chargement du mot d'état correspondant.
  - Traitement de l'interruption.
  - Appel de l'ordonnanceur pour élire le processus à activer.
  - Restauration du mot d'état et du contexte du processus élu.

Rôle de l'ordonnanceur (1)

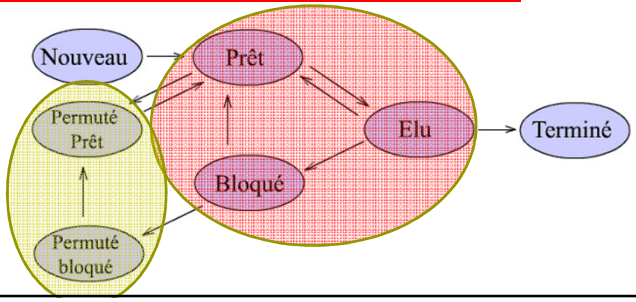
- L'ordonnanceur ne fournit pas seulement un mécanisme, mais prend des décisions (une politique).
- Un bon ordonnanceur (pour les systèmes d'exploitation généralistes / temps partagés) se doit :
  - de **maximiser** l'utilisation du processeur,
  - d'être **équitable** envers des processus équivalents,
  - de présenter un **temps de réponse** acceptable,
  - d'avoir un bon **rendement/débit**,
  - d'assurer certaines **priorités**.

Rôle de l'ordonnanceur (2)

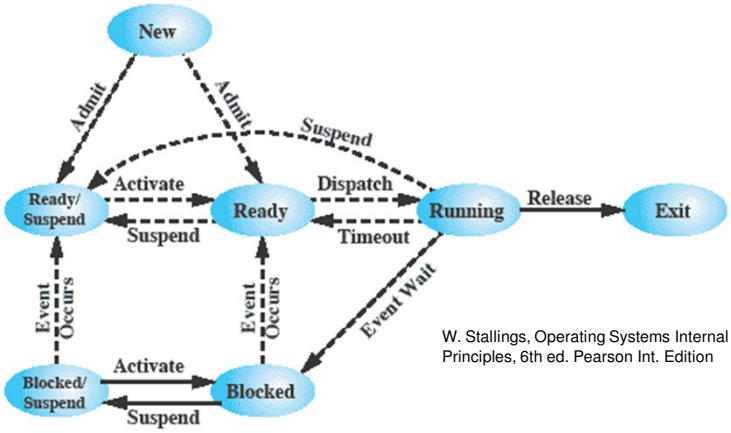
- Un bon algorithme d'ordonnancement doit donc être capable de :
  - s'assurer que chaque processus reçoit sa part de temps processeur (sinon pb de famine),
  - utiliser le temps processeur à 100 %,
  - minimiser les temps de réponse pour les utilisateurs en mode interactif,
  - minimiser l'attente des utilisateurs qui travaillent en batch,
  - maximiser le nombre de travaux effectués en une heure.
- Ces objectifs sont bien évidemment parfois **contradictoires**.

Ordonnancement à 3 niveaux

- **Ordonnanceur d'admission / long terme**: degré de multiprogrammation
  - Quels processus admettre dans le système
- **Ordonnancement de mémoire / moyen terme**
  - Quels processus placer en mémoire et lesquels en swap (voir TD mémoire)
- **→ Ordonnancement processeur / court term**



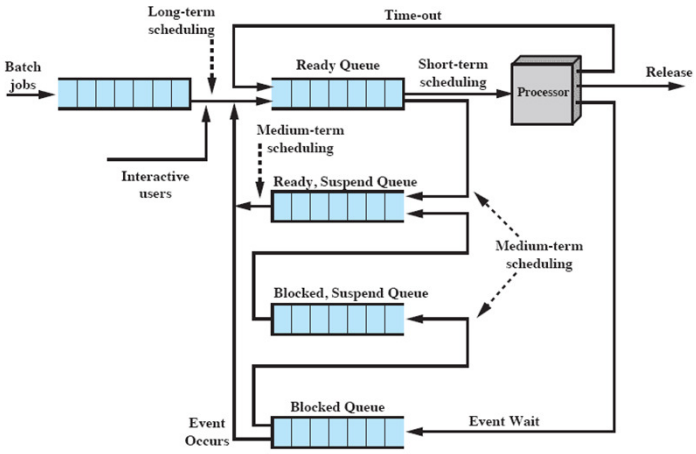
Ordonnancement à 3 niveaux (2)



(b) With Two Suspend States

W. Stallings, Operating Systems Internal Design and Principles, 6th ed. Pearson Int. Edition

Ordonnancement à 3 niveaux (3)



W. Stallings, Operating Systems Internal Design and Principles, 6th ed. Pearson Int. Edition

Figure 9.3 Queuing Diagram for Scheduling

Quand ordonnancer ?

Les décisions d'ordonnancement sont prises dans plusieurs circonstances:

- **Fork**
  - Qui choisir entre le père et le fils ?
- **Fin d'un processus**
  - En trouver un autre
  - Si aucun, un processus spécial est exécuté (*idle* ou *Processus inactif du système*)
- **Quand un processus devient bloqué**
  - Attente d'une E/S ou d'une condition
  - Peut influer sur le prochain à exécuter
- **Interruption E/S**
  - Si indique fin de traitement, réordonnancer le processus qui attendait

Ordonnancement non préemptif

Un processus est choisi et détient le CPU jusqu'à ce qu'il bloque ou se termine

Ordonnancement préemptif

Un processus n'a le CPU que pour une durée donnée (*timeslice*)

La notion d'espace de priorités

- **2 possibilités**:
  - Certains OS laissent toutes les tâches (système et utilisateur) « vivre » dans **un même espace de priorités**.
  - D'autres **isolent l'espace des priorités** des processus système et utilisateurs (WindowsNT, Linux, UNIX)
    - Sous Unix par ex, les tâches utilisateurs voulant augmenter leur priorités peuvent utiliser l'appel à `nice()`
    - … mais sont tous préemptées par une tâche système.
- **L'espace noyau a 3 types de *« tâches » (en termes de priorités)*:**
  1. Interruptions: c'est une interruption hw (timer, clavier, réseau, etc). Ce type de tâche est appelé ISR (Interrupt Service Routine). Ce n'est pas vraiment une tâche mais une fonction exécutée ***indépendamment de l'ordonnanceur*** à chaque interruption. L'OS n'est pas impliqué.
  2. Fonctions *tasklets (minitâches)* et routines de service différées: fonctions qui peuvent être activées par n'importe quelle tâche du noyau (pour la mini tâche) ou par une fonction d'interruption (pour les DSR). Les interruptions sont actives lors de l'exécution de ces fonctions ***et l'OS est impliqué*** pour déterminer l'ordre d'exécution.
  3. Toutes les autres tâches du noyau: le niveau le moins prioritaire du noyau, préempte toutes les tâches utilisateurs.

Métriques/critères clés de l'ordonnancement

- Orientés utilisateur / Performance
  - Temps de rotation / turnaround: temps entre soumission et complétion (exécution + attente de ressources) → batch
  - Temps de réponse: temps entre soumission et début de réponse → interactif (point de vue utilisateur)
  - Deadlines: si deadlines connus → maximiser le nbr de deadlines respectées
- Orientés utilisateur / autre
  - Prédictibilité: même temps et même coût d'exécution indépendamment de la charge du système

Métriques/critères clés de l'ordonnancement (2)

- Orientés système / performance
  - Débit: maximiser le nbr de processus par unité de temps
  - Utilisation processeur: meilleure utilisation de la ressource CPU (temps partagé)
- Orientés système / autre
  - Equité: processus équivalent / temps d'exécution équivalent
  - Priorités: respect des priorités (si système à priorité)
  - Utilisation/équilibre des ressources: garder les ressources utilisée → complémentarité des processus par rapport aux ressources

Différents besoins/différents ordonnancements

- Des systèmes ont des propriétés et besoins différents
- Il faut des ordonnanceurs adaptés
- **Batch**
  - Pas d'utilisateurs devant des écrans
  - Ordonnanceurs non préemptifs ou presque…
  - Changements de processus réduits
  - Ex: compilateur de langage, recherche dans des BDs, calcul scientifique, etc.
- **Interactif**
  - Préemption pour maintenir la réactivité
  - Ex: éditeur de texte, shell, etc.
- **Temps réel**
  - Préemption
  - Ex: Application vidéo, audio, collecte de données à partir d'un capteur.

Objectifs de l'ordonnancement

- Les objectifs des algorithmes d'ordonnancement dépendent du système considéré
  - Mais certaines propriétés sont communes
- **Ce qui est commun**
  - Respect des règles : permettre à certains processus d'avoir un ordonnancement particulier
  - Équilibre : Maximiser l'utilisation du système (alterner des E/S et du CPU)
  - Équité : chaque processus doit avoir accès au CPU de manière équitable (si processus équivalent)
- **Batch**
  - Débit : maximiser le nombre de jobs par heure
  - Délai de rotation de l'appli: minimiser le temps entre la soumission et la complétion
  - CPU : maximiser l'utilisation du CPU

Objectifs (2)

- **Interactif**
  - Réactivité: répondre rapidement aux demandes
  - Proportionnalité
    - Les utilisateurs ont souvent une idée du temps que va prendre une opération (click == rapide…)
    - L'ordonnanceur doit choisir les processus pour être conforme à leurs attentes
- **Temps réel**
  - Respecter les contraintes temporelles
  - Prédictibilité : l'ordonnanceur doit être prévisible
- Certains de ces objectifs peuvent ne pas être maximisés en même temps (voire contradictoires)
  - Débit et temps de réponse par exemple…

Ordonnancement avec réquisition/préemption

- Lorsque l'ordonnanceur lance un processus, il ne sait pas à l'avance combien de temps ce processus s'exécutera avant de se bloquer sur une E/S, un sémaphore ou pour une autre raison.
- Pour s'assurer qu'aucun processus ne s'exécutera pendant trop de temps, le système d'exploitation reprend la main (à chaque **interruption d'horloge**) et décide :
  - Si le processus courant doit poursuivre son exécution.
  - S'il a consommé le temps processeur qui lui était imparti.
  - Dans ce dernier cas, le processus est suspendu et le processeur est alloué à un autre processus.

Ordonnancement avec réquisition (2)

- Cette stratégie qui permet de suspendre des processus prêts est appelée ordonnancement avec **réquisition/préemption** (preemptive scheduling).
- Le fait qu'un processus peut être suspendu à n'importe quel instant (sauf s'il s'exécute en mode noyau) peut conduire à des conflits d'accès qu'il faut éliminer par l'usage des primitives de communication et d'exclusion inter-processus :
  - Sémaphores
  - Mémoire partagée
  - Files de messages
  - etc.

Le partage de l'unité centrale (1)

- Ce partage doit être fait
  - non seulement entre les processus utilisateurs
  - mais aussi entre les différentes tâches du système : ordonnanceur, entrées-sorties, ...
- Par rapport à l'unité centrale le but de l'ordonnancement est de maximiser le débit et taux d'utilisation de l'unité centrale:
  - Le ***débit*** est le nombre moyen de processus exécutés en un temps donné.
  - Le ***taux utile*** est la proportion de temps réellement utilisée pour exécuter des processus utilisateurs.
- L'algorithme d'ordonnancement doit également prémunir les tâches du problème de famine.

Le partage de l'unité centrale (2)

- La conception d'un algorithme d'ordonnancement se base en générale sur des remarques statistiques concernant le comportement des processus:
  - Le couple UC/ES (cpu/io) est important car les processus ont tendance à basculer constamment entre des phases d'**entrées-sorties** et des phases de **calcul** sur l'unité centrale.
  - Les processus consommant de longues périodes d'UC sont proportionnellement rares.

Stratégie de l'ordonnancement à court ou à long terme

- Les ordonnancements à court terme doivent être très rapides
  - Le processus élu ne va utiliser l'unité centrale que pendant un très court laps de temps (10 milli-secondes par exemple). Si on utilise trop de temps (1 milli-seconde) pour sélectionner cet élu, le taux utile décroît très rapidement (ici on perd 10% du temps d'unité centrale).
- L'ordonnancement à long terme peut être plus long car il a lieu moins souvent
  - Toutes les secondes par exemple.

Quelques algorithmes d'ordonnancement

- Ordonnancement Batch
- Ordonnancement Interactif

Ordonnancement Batch

Politique FCFS / exécution jusqu'à terminaison

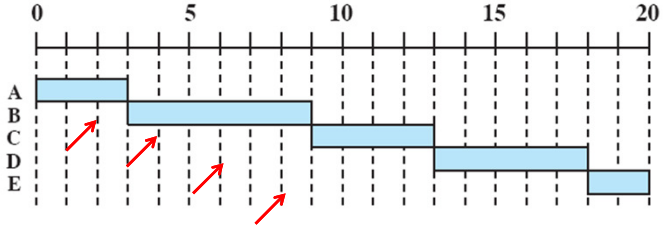
- First Come First Served (premier arrivé, premier servi).
- Ordonnancement utilisé pour les batchs
- Approche non préemptive:
  - Pas de file d'attente de tâches bloquées
- Temps de rotation lent
  - Surtout s'il existe des tâches longues
  - Dans ce cas: pb d'équité (les tâches courtes « attendent » proportionnellement plus que les tâches longues)
- Famine: impossible.

FCFS

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

First-Come-First Served (FCFS)



Le plus court d'abord (SPN/Shortest Process Next)

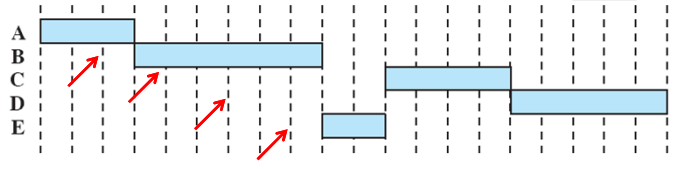
- On suppose que le temps d'exécution est connu à l'avance
- La tâche la plus courte de l'ensemble des processus s'exécute en premier.
- Temps de rotation plus court que le précédent
- Famine possible
- Temps de calcul (de l'ordonnanceur) peut être important: pour savoir quel processus doit s'exécuter
- Version préemptive:
  - Le job dont le temps restant est le plus court

SPN

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Shortest Process Next (SPN)



SPN exemple

- Délai de rotation/turnaround: 8, 12, 14, 18, moyenne de 13.5
- Délai de rotation/turnaround: 2, 6, 10, 18, moyenne: 9
- Présuppose la disponibilité de toutes les tâches au moment de l'ordonnancement.
- Si 5 jobs: a, b, c, d, e avec temps d'exécution: 2, 4, 1, 1, 1 et temps d'arrivée de 0, 0, 0, 3, 3
  - SPN: a, b, c, d, e
  - Si tous dispos: b,c, d, e, a.

Ordonnancement interactif

Tourniquet

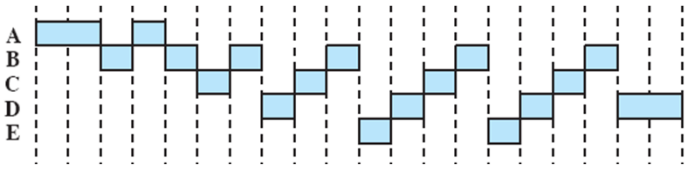
- Cet algorithme est l'un des plus utilisés et l'un des plus fiables :
  - Chaque processus PRÊT dispose d'un **quantum de temps** pendant lequel il s'exécute.
  - Lorsqu'il a épuisé ce temps ou qu'il se bloque, par exemple sur une entrée-sortie, le processus suivant de la file d'attente est élu et le remplace.
  - Le processus suspendu est mis en queue du tourniquet.
- Le seul paramètre important à régler, pour le tourniquet, est la durée du quantum
  - La part de gestion du système correspond au rapport de la durée de commutation sur la durée du quantum.
  - Plus le quantum est long plus cette part est faible, mais plus les utilisateurs attendent longtemps leur tour.

Tourniquet /Round Robin

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Round-Robin (RR), $q = 1$



Ordonnancement prioritaire

- Dans l'algorithme du tourniquet, un même quantum pour tout le monde rend les différents processus égaux.
  - Il est parfois nécessaire de privilégier certains processus par rapport à d'autres.
  - L'algorithme de priorité choisit le processus prêt de plus haute priorité.
- Ces priorités peuvent être **statiques** ou **dynamiques** :
  - Les processus du système auront des priorités statiques (non-modifiables) fortes.
  - Les processus des utilisateurs verront leurs priorités modifiées, au cours de leur exécution, par l'ordonnanceur.
  - Ainsi un processus qui vient de s'exécuter verra sa priorité baisser par exemple.

Le tourniquet avec priorités

- Très souvent sur les systèmes, une combinaison des deux techniques précédentes est utilisée.
  - À chaque niveau de priorité (file d'attente séparée) correspond un tourniquet.
  - L'ordonnanceur choisit le tourniquet non vide de priorité la plus forte et l'exécute.
  - Pour que tous les processus puissent s'exécuter, il est nécessaire d'ajuster périodiquement les différentes priorités.

Variantes ...

- SPN pour processus interactifs
  - Estimer le temps d'exécution d'un processus selon l'historique:
    - Estimé = T0, estimation suivante = T1
    - Actualisation de l'estimation par pondération:
      - a*T0+(1-a)*T1
      - Ex: a= ½
      - T0, ½*T0+½*T1, ¼ *T0+ ¼ *T1 + ½ *T2, 1/8*T0+1/8*T1+1/4 T2 + ½*T3

Variantes (2)

- Feedback: suppose que l'on ignore le temps d'exécution global, on se base sur le temps passé à s'exécuter.
  - Utilisation de plusieurs files d'attente / priorités différentes
  - FCFS par file

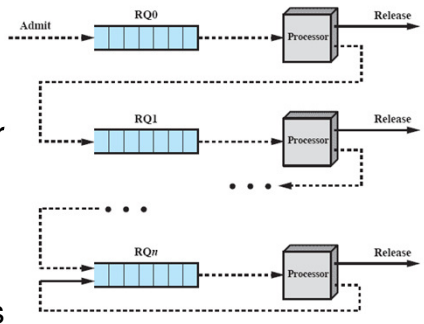


Figure 9.10 Feedback Scheduling

Variantes (3)

- Ordonnancement garanti:
  - Si n processus → 1/n temps CPU
  - Suivi des temps processeurs par processus
  - Calcul du temps processeur par processus: temps écoulé depuis création / nbr de processus
  - Calcul du ratio du temps processeur consommé Vs réellement autorisé
    - Ratio de 0,25: le processus n'a exécuté que le quart du temps dont il a droit
  - → exécuter le processus avec ratio le plus faible

Variantes (4)

- Ordonnancement équitable:
  - Utilisateur 1 lance 9 processus
  - Utilisateur 2 en lance 1
  - 90% CPU pout le 1er et 10% pour le second.
  - Prise en compte de l'utilisateur lors de l'ordonnancement
- U1: 4 processus (a, b, c, d), U2: 1 processus (z)
  - Si 50% de CPU chacun: a, z, b, z, c, z, d, z, a, z …
  - Si 75% u1, et 25% u2: a, b, c, z, d, a, b, z, ….
  - Rajouter des priorités par utilisateur …

Ordonnancement processus et thread

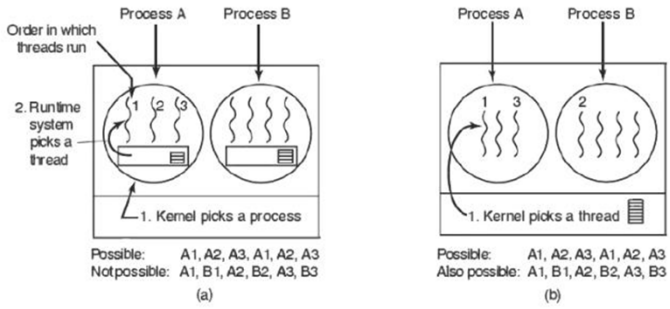


Figure 1. (a) Possible scheduling of user-level threads with a 50-msec process quantum and threads that run 5 msec per CPU burst. (b) Possible scheduling of kernel-level threads with the same characteristics as (a).

Sous Linux, threads et processus sont mis à plat

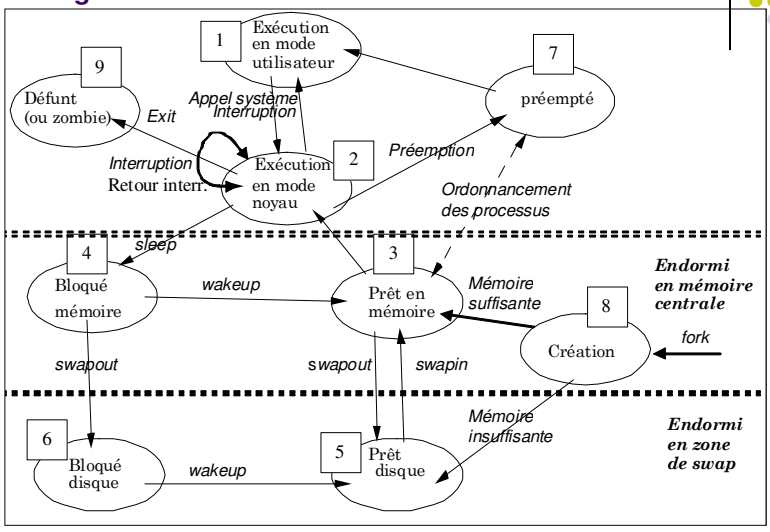
Diagramme des états et transitions sous UNIX

Présentation du diagramme d'état sous UNIX
Diagramme des états et transitions sous UNIX
Description des états et transitions sous UNIX
Un exemple de transition d 'état

1.3.1 Présentation du diagramme d'état UNIX

- Un diagramme plus complexe pour 3 raisons :
  - Sous Unix, l'exécution d'un processus se fait sous deux modes :
    - Le mode noyau qui correspond aux appels système. Un processus en mode noyau ne pouvant être suspendu par l'ordonnanceur, il passe dans un état appelé " préempté " à la fin de son exécution.
    - Le mode utilisateur qui correspond à l'exécution des autres instructions.
  - Le code et les données du noyau résident en permanence dans le système et tous les processus se les partagent par opposition à l'espace d'adressage du processus.
    - Lorsque la mémoire centrale ne peut contenir tous les processus prêts, certains sont déplacés sur le disque (zone de swap).
  - Les processus terminés ne sont pas immédiatement éliminés de la table des processus (état zombie).

1.3.2 Diagramme des états et transitions UNIX



1.3.3 Description des états et transitions UNIX (1)

- Etat 1 : Exécution en mode utilisateur
  - Le processus accède à ses données et exécute ses instructions contenues dans la structure interne propre au processus.
- Etat 2 : Exécution en mode noyau pour tous les appels système.
  - Le processus accède à des données du système et exécute le code système qui résident en permanence dans le système.
- Etat 3 : Prêt en mémoire
  - Le processus ne s'exécute pas mais il est éligible. Il est prêt à s'exécuter.

1.3.3 Description des états et transitions UNIX (2)

- Etat 4 : Bloqué en mémoire
  - Quand un processus exécute un appel système, il passe du mode utilisateur au mode noyau. Si le processus doit attendre une ressource (Ex. : un sémaphore) ou le résultat de son appel système (Ex. : une entrée/sortie) il est mis en sommeil.
- Etat 5 : Prêt sur disque
  - Le processus est prêt à s'exécuter mais le swapeur doit le transférer en mémoire centrale pour le rendre éligible.
- Etat 6 : Bloqué sur disque
  - Le processus est endormi en zone de swap (sur disque) en attente de ressources.

1.3.3 Description des états et transitions UNIX (3)

- Etat 7 : Préempté
  - Un processus en mode noyau ne peut pas être suspendu par l'ordonnanceur. Au retour de son appel système, le processus peut néanmoins être préempté c'est-à-dire qu'il reste prêt à s'exécuter mais c'est un autre processus qui est élu.
- Etat 8 : Création
  - C'est l'état qui correspond à la naissance d'un processus après l'appel système *fork* .
- Etat 8 : Zombie
  - L'état défunt ou zombie correspond à l'état du processus qui vient de se terminer. Il est conservé dans la table des processus, le temps pour son processus père de récupérer certaines informations.

1.3.4 Exemple de transition d'état

- Soit la situation suivante :
  - L'ensemble de la mémoire est occupé par des processus, mais le processus le plus prioritaire est dans l'état 5 " Prêt sur disque ".
  - Pour pouvoir exécuter ce processus, il faut le placer dans l'état 3 " Prêt en mémoire ".
  - Pour cela le système doit au préalable libérer de la mémoire en faisant passer un ou plusieurs processus des états 3 " Prêt en mémoire " ou 4 " Bloqué en mémoire " dans la zone de swap, donc les faire passer dans les états 5 " Prêt sur disque " ou 6 " Bloqué sur disque ".
- C'est le **swapeur** qui réalise ces opérations :
  - Sélection de processus pour un transfert sur disque (swapout).
  - Réalisation du transfert de la mémoire centrale vers l'espace de swap.
  - Chargement en mémoire (swapin) du processus prioritaire.

Appels système relatifs à l'ordonnancement

- Conventionnels
  - **`nice( ):`** changement de la valeur de « gentillesse ». Maintenu pour des raisons de compatibilité et remplacé par `setpriority`()
    - Valeurs négatives: augmentation de priorité statique (superutilisateur)
  - **`getpriority( )`** **et** **`setpriority( ):`** concerne tout le groupe de processus (priorité statique max).
  - **`sched_getaffinity( )`** **et** **`sched_setaffinity( ):`** quels sont les CPUs qui peuvent exécuter ce processus.
- Temps réel
  - **`sched_getscheduler( )`** **et** **`sched_setscheduler( ):`** politique d'ordonnancement
  - **`sched_ getparam( )`** **et** **`sched_ setparam( ):`** retourne les paramètres de la politique d'ordonnancement (priorité temps réel).
  - **`sched_yield( ):`** relâche le CPU.
  - **`sched_get_priority_min( )`** **et** **`sched_get_priority_max( ):`** retourne la priorité min et max utilisable par le processus.
  - **`sched_rr_get_interval( ):`** retourne la taille du quantum de temps de la politique du tourniquet.