Les systèmes d'exploitation

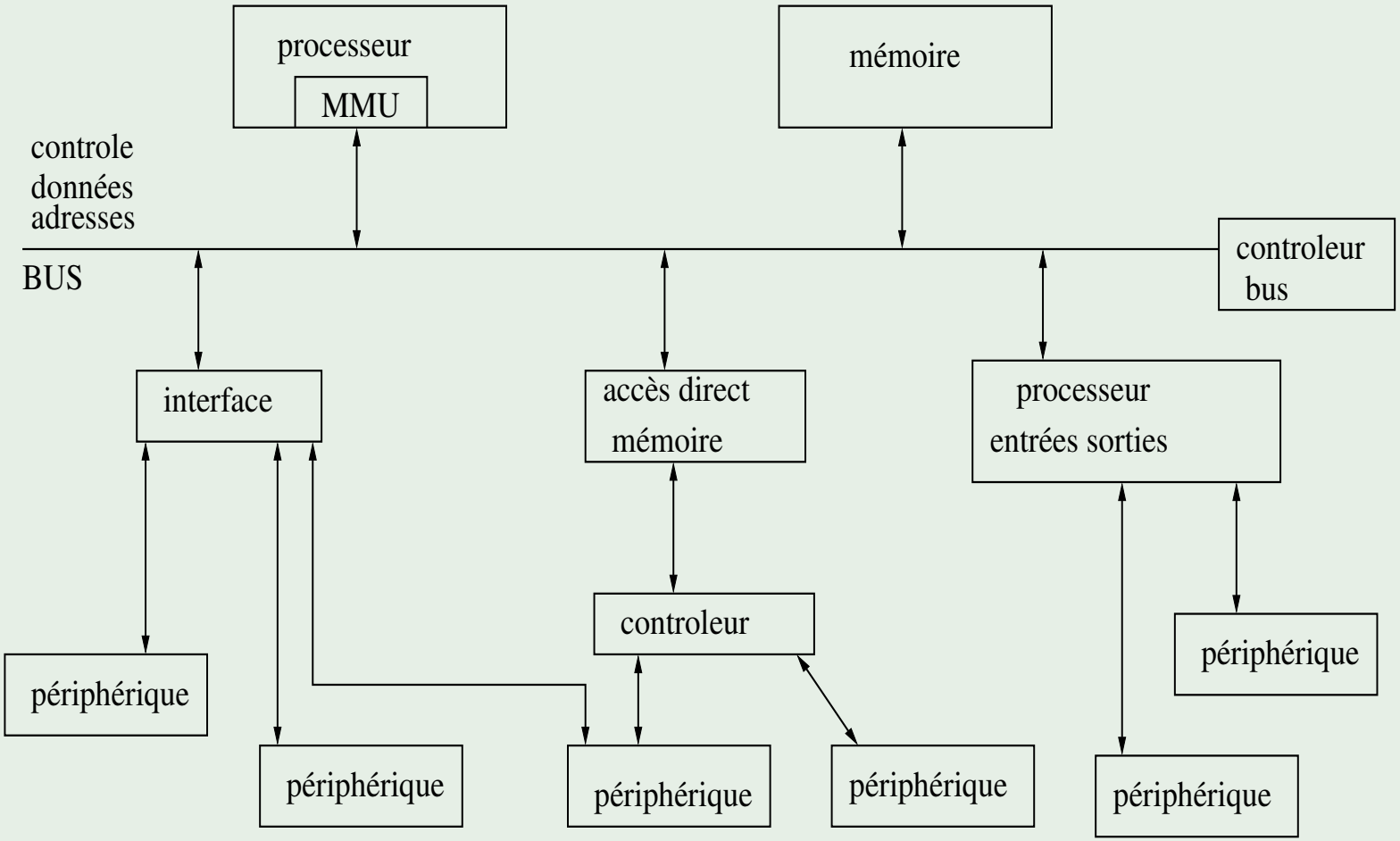
Wadoud Bousdira[1]
wadoud.bousdira@univ-orleans.fr

[1]LIFO, University of Orléans
Orléans, France

Orléans, 2023

Plan

Architecture (générique) d'une machine



Tâches du SE pour

exécuter des programmes en mémoire ?

- coordonner des activités simultanées
  - plusieurs programmes utilisateurs
    chaque programme a l'illusion qu'il dispose de son propre processeur et de sa propre mémoire
  - les programmes du SE
  - les opérations d'E/S. Elles sont sous le contrôle exclusif du SE
- distribuer les ressources pour que chaque processus en reçoive une allocation *"suffisante"* pour s'exécuter *"normalement"*
- réagir à des événements extérieurs
- assurer une protection étanche entre les domaines des différents processus et son propre domaine
- contrôler le temps : base de temps unique pour tous les processus + services de gestion du temps aux processus qui le demandent

Introduction

Événements extérieurs

Ex. clavier, souris, buffer réseau etc. . .

- attente active : technique de programmation utilisée par un processus en vérifiant de façon répétée si une condition est vraie, comme l'attente d'une entrée (clavier ou autre) ou encore la libération d'un verrou $\rightsquigarrow$ gaspillage de temps CPU
- nécessité d'un mécanisme pour réagir à un signal (asynchrone)

Protéger la mémoire

Instructions exécutables de 2 types :

1. les instructions privilégiées (Ex. inst. d'E/S)
2. les instructions utilisateur

2 modes de fonctionnement :

1. mode maître (noyau, superviseur) : toutes les instructions
2. mode esclave (user) : seules certaines instructions sont autorisées

Introduction

- à un instant t, un seul contexte peut correspondre à l'état effectif du système
- l'illusion de la présence de plusieurs processeurs repose sur la possibilité de sauvegarder/restaurer le contexte courant du processeur

$\rightsquigarrow$ pseudo-parallélisme

Nécessité de mécanismes rapides pour :

- passer de l'exécution d'un processus à un autre
- réagir à un événement extérieur
- passer du mode maître au mode esclave

Un seul mécanisme : la commutation de contexte

Représentation interne d'un processus

Un processus est

l'abstraction d'un programme en cours d'exécution
Entité dynamique.

- ensemble d'instructions pour le processeur
- des données stockées en mémoire
- un contexte si le processus utilise un seul thread d'exécution, plusieurs contextes sinon

Contexte d'un processus

formé de deux parties distinctes :

1. le contexte du processus
2. le contexte en mémoire (le segment procédure et le(s) segment(s) de données)

Contexte d'un processus

constitué de l'ensemble des informations qui définissent son état à un instant donné :

- les registres généraux. Adressables et modifiables par l'utilisateur (par le programme)
- les registres spécialisés. Inaccessibles directement à l'utilisateur et regroupés dans le mot d'état du programme (Program Status Word - PSW)

Contexte d'un processus

Le PSW

ses bits servent d'indicateurs sur l'état du processeur

1. l'état d'exécution du processeur (actif ou en attente)
   - passage "actif" $\rightarrow$ "en attente" : réalisé
     - soit par le processeur lui-même
     - soit par l'exécution d'une instruction spécialisée
     - soit par (auto)modification du PSW
   - passage "en attente" $\rightarrow$ "actif" : obtenu par un événement extérieur (ue interruption)
   - le mode maître ou esclave
   - les masques d'interruption
2. les informations sur le déroulement de l'activité en cours : le code condition, le compteur ordinal...

Contexte d'un processus

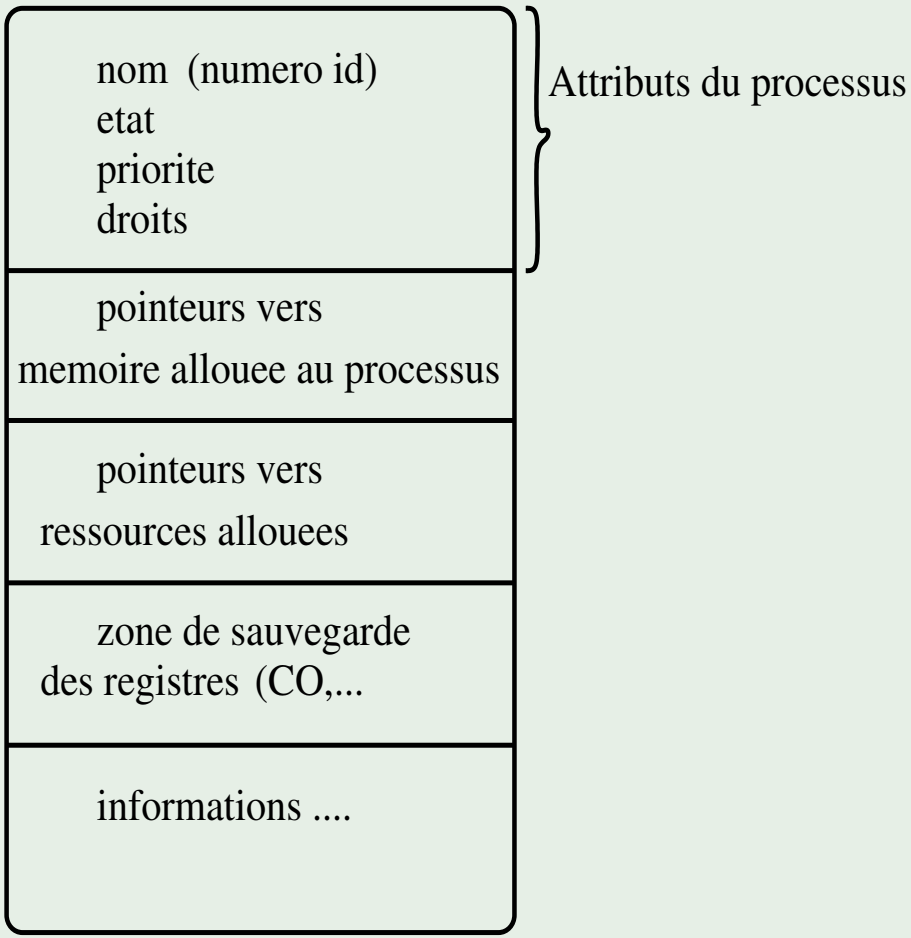
Le contexte en mémoire du processus

comprend la table des segments et les indicateurs de protection

- segments de code réentrants (exécutés par plusieurs processeurs) sans duplication de code
- les segments de données peuvent appartenir à une seule procédure, ou à un processus, ou être partagés entre plusieurs processus

Bloc de contrôle d'un processus

Un processus est représenté par une structure appelée bloc de contrôle



Le bloc de contrôle du processus

il comprend :

- l'état du processus,
- le numéro qui l'identifie,
- PC (compteur ordinal) = @ prochaine instruction exécutable,
- les valeurs des registres concernés par l'exécution
  valeurs sauvegardées lorsque le processus quitte l'état en exécution (ou élu) et rechargées quand il le retrouve
- des informations
  - sur la quantité de mémoire allouée au processus et des pointeurs vers cette mémoire
  - quantitatives : temps d'UC utilisé, temps restant autorisé
  - concernant les opérations d'E/S du processus : périphériques alloués, opérations en attente de complétion, fichiers ouverts etc...
  - concernant les files d'attente où figure le processus : numéro de priorité, pointeurs vers ces files etc...

Le bloc de contrôle du processus

- rangé dans la zone mémoire du système
- la plupart des machines ont un registre spécial qui pointe vers le bloc de contrôle du processus courant $\rightsquigarrow$ manipulation plus rapide
- souvent, des instructions câblées effectuent la sauvegarde et le chargement des blocs de contrôle

Représentation interne d'un processus

Files d'attente de blocs de contrôle

L'ensemble des blocs de contrôle des processus sont maintenus en mémoire par le SE pour réaliser l'ordonnancement

- file d'attente des processus prêts
- file d'attente des processus en attente d'un événement (E/S, signaux,...)

Les processus passent d'une file d'attente à l'autre au cours de leur exécution

- la partie du SE qui gère ces files est l'ordonnanceur (scheduler)
- ce dernier prend la main suite à une interruption, il doit alors sélectionner parmi les processus prêts celui qui va être acrivé $\rightsquigarrow$ déplace un processus d'une file à l'autre

Représentation interne d'un processus

États d'un processus

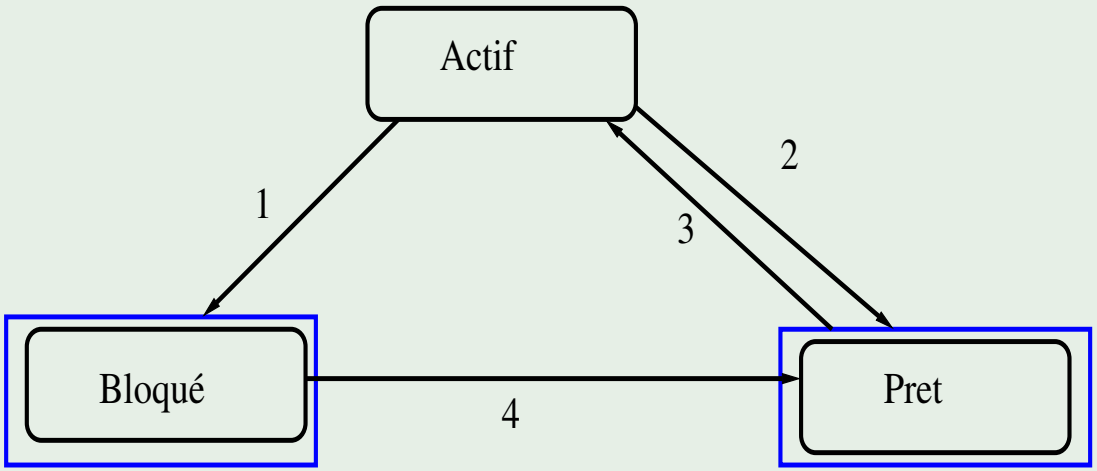
Un processus peut être

1. actif (ou élu) : utilise le processeur
2. prêt : éligible pour exécution mais ne dispose pas du processeur
3. bloqué : en attente d'un événement (E/S, ...)

Du point de vue processus, (1) et (2) sont analogues. Ils indiquent que le processus est prêt à s'exécuter, mais dans (2), aucun processeur n'est disponible

Représentation interne d'un processus

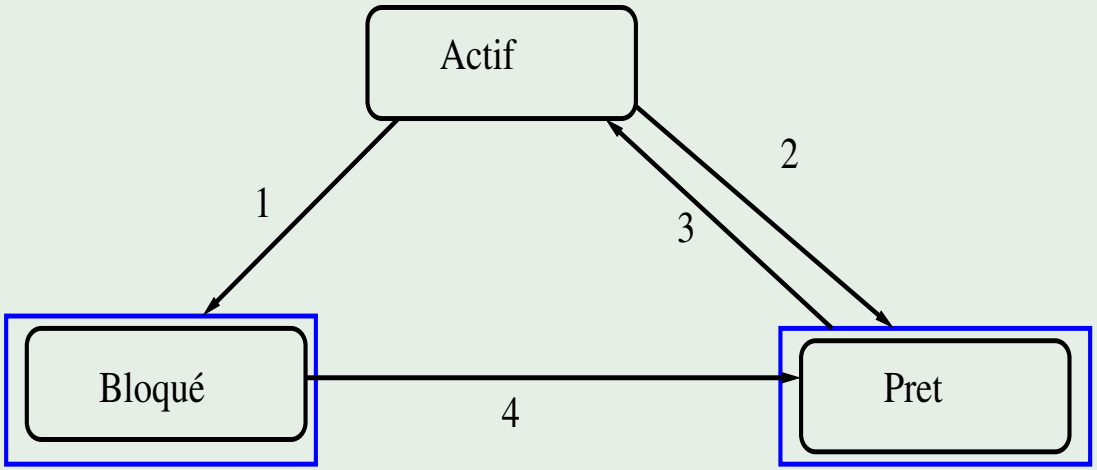
Transitions entre états d'un processus



1. bloqué en attente d'un événement
2. élection d'un autre processus
3. élection de ce processus
4. l'événement se produit

Représentation interne d'un processus

Transitions entre états d'un processus



1. la transition 1 est provoquée par le processus (ex. demande d'E/S)
2. les transitions 2 et 3 sont provoquées par le SE
3. la transition 4 est provoquée par un événement (ex. réponse E/S,...)

Représentation interne d'un processus

Opérations sur les processus

Réalisées par le SE.

- Créer : 4 événements peuvent créer un processus
  1. initialisation du système
     processus de 1er plan, processus d'arrière-plan
  2. exécution d'un appel système de création d'un processus par un processus en cours d'exécution (`fork` en UNIX)
  3. requête utilisateur sollicitant la création d'un nouveau processus. Ex. clique icône
  4. lancement d'un travail en traitement par lots

Opérations sur les processus

Créer

Tout processus est créé par un processus père

Sous UNIX, `fork` : seul appel système qui crée un nouveau processus

- le processus père et le processus fils ont la même image mémoire et les mêmes fichiers ouverts
- le processus fils dispose d'une copie de l'espace d'adressage de son père

Hiérarchie qui permet aux processus de partager des événements

la distinction entre les deux processus se fait par leurs identifiants :

- `getpid` : retourne l'identifiant du processus
- `getppid` : retourne l'identifiant du processus père

Opérations sur les processus

`fork`

L'appel de fork peut être suivi dans le processus fils d'un appel à une fonction de la famille `exec` qui permet de remplacer l'image mémoire du processus par celle d'un nouveau programme

Opérations sur les processus

Détruire

Motifs de terminaison :

- arrêt volontaire : fin d'exécution du programme.
- Erreur fatale (Ex. accès mémoire illégal, division par 0, etc...).
- à la demande :
  - un processus peut effectuer un appel système pour demander l'arrêt d'un autre processus (`kill`, ou `exit, abort` si auto-destruction)
  - le SE peut décider d'arrêter un processus pour libérer de la mémoire comme c'est le cas pour Android
    (trop de processus en mémoire $\Rightarrow$ performances réduites)

les ressources sont libérées, le bloc de contrôle est effacé et le processus disparaît des tables et des listes

Opérations sur les processus

Changer la priorité

pourquoi ?

- on modifie l'information correspondante dans son bloc de contrôle
- l'algorithme d'ordonnancement utilise les priorités des processus comme paramètres. Le SE, par souci d'équité, augmente périodiquement les priorités des processus qui ne sont pas choisis

Contexte mémoire du processus

Appel et retour de procédure :

[a] L'appelant p appelle l'appelé q :

1. p transfère la valeur de ses arguments à q
2. le SE sauvegarde le contexte d'exécution de p
3. le SE met en place le contexte de q
   q doit connaître
   - l'adresse à laquelle transférer le contrôle quand il aura fini (adresse de retour)
   - l'adresse où déposer le résultat pour p

Retour :

1. q transfère ses résultats à p
2. le SE restaure le contexte d'exécution de p

Gestion par pile

---

a. fonction, sous-programme

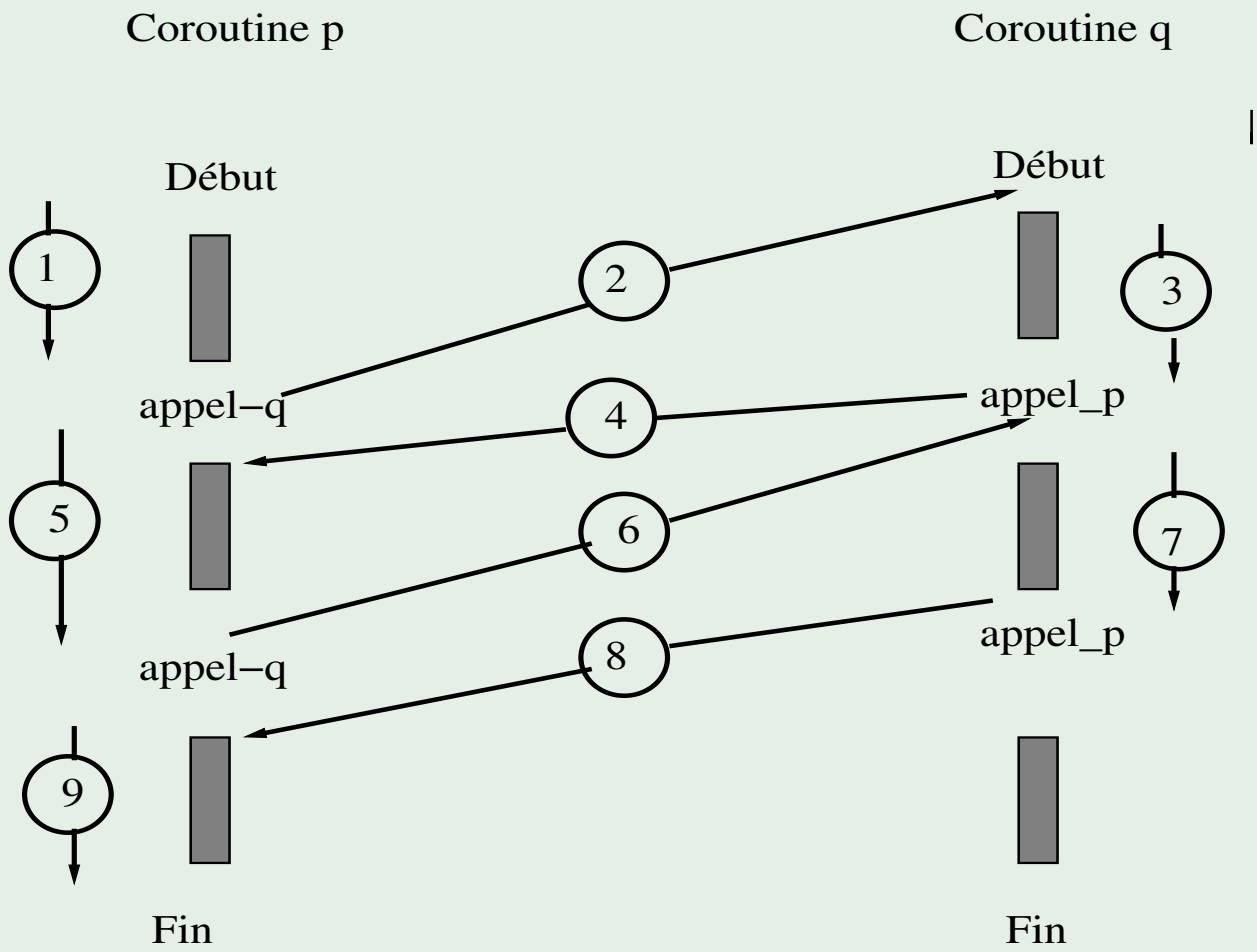
Contexte du processus

Coroutine

Une coroutine

- est un thread léger collaboratif. C'est-à-dire que la transition vers une autre coroutine s'effectue à sa demande
- généralise la notion de fonction
  au lieu d'avoir un seul point d'entrée et de sortie, les coroutines en ont plusieurs, ce qui leur permet de suspendre et de reprendre leur exécution
- les coroutines s'exécutent de manière séquentielle mais entrelacée
- la symétrie entre les coroutines est totale, chacune d'elles pouvant être appelée la première $\rightsquigarrow$ une pile par coroutine

Mécanisme d'appel de coroutines



Contexte du processus

Contexte lié au programme

c'est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu

- registres généraux (de calcul, pile, ...) et
- du mot d'état du programme, le PSW

Contexte d'UC d'un processus

L'UC comprend des registres (mémoires) dont :

- l'accumulateur : reçoit le résultat d'une instruction
- le registre d'instruction : contient l'instruction en cours
- le compteur ordinal : pointe sur l'adresse de la prochaine instruction à exécuter
- le registre d'adresse
- les registres de données : utilisés pour lire ou écrire une donnée à une adresse spécifiée en mémoire
- les registres d'état du processeur (*actif, mode (user/system), retenue, vecteur d'interruption, etc...*)
- les registres d'état du processus (droits, adresses, priorité, etc...)

Commutation de contexte

C'est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu

- registres généraux (de calcul, pile, ...) et
- du mot d'état du programme (PSW)

Commutation de contexte

Changer la valeur du PSW

Algorithme réalisé de façon matérielle et de manière atomique

- sauvegarder le PSW du processus dans un emplacement mémoire
- charger le nouveau PSW à partir d'un emplacement spécifié

la commutation de contexte peut avoir lieu après chaque instruction du processeur

Commutation de contexte

3 types

Nom	Cause	Utilisation	Syntaxe
Interruption	Extérieure au programme	Réaction à un événement extérieur	ancien_psw[i] nouveau_psw[i]
Déroutement	Lié à l'instruction en cours	Traitement des exceptions	ancien_derout nouveau_derout
Appel au superviseur	Instruction spécifique	Appeler les procédures du SE (passage en mode maître)	svc

Seuls les déroutement sont des événements synchrones

Les interruptions

- Les interruptions d'E/S sont générées par le matériel des périphériques d'E/S
- Les appels au système sont provoqués par le processus en exécution et résultent de l'insertion dans le corps du programme associé au processus, d'une instruction particulière `svc`

Les interruptions

Algorithme de traitement

- le programme en cours est arrêté
- le SE préserve le contexte en cours
- le SE détermine le type d'interruption. Pour chaque type, il détermine l'action qui doit être exécutée
- dès que cette procédure est terminée, *le programme interrompu reprend son exécution*[a]
- au début de la reprise, la machine doit se trouver exactement dans l'état où elle était au moment de la prise en compte de l'interruption

SE modernes : orientés interruptions

---

a. À chaque it, l'ordonnanceur prend la main, il n'affectera pas forcément le processeur au dernier processus interrompu

Les interruptions

Vecteur d'interruption

lors d'une interruption (interne ou externe), le signal modifie un (ou plusieurs) bit(s) du mot d'état consulté(s) régulièrement par le système : le vecteur d'interruption

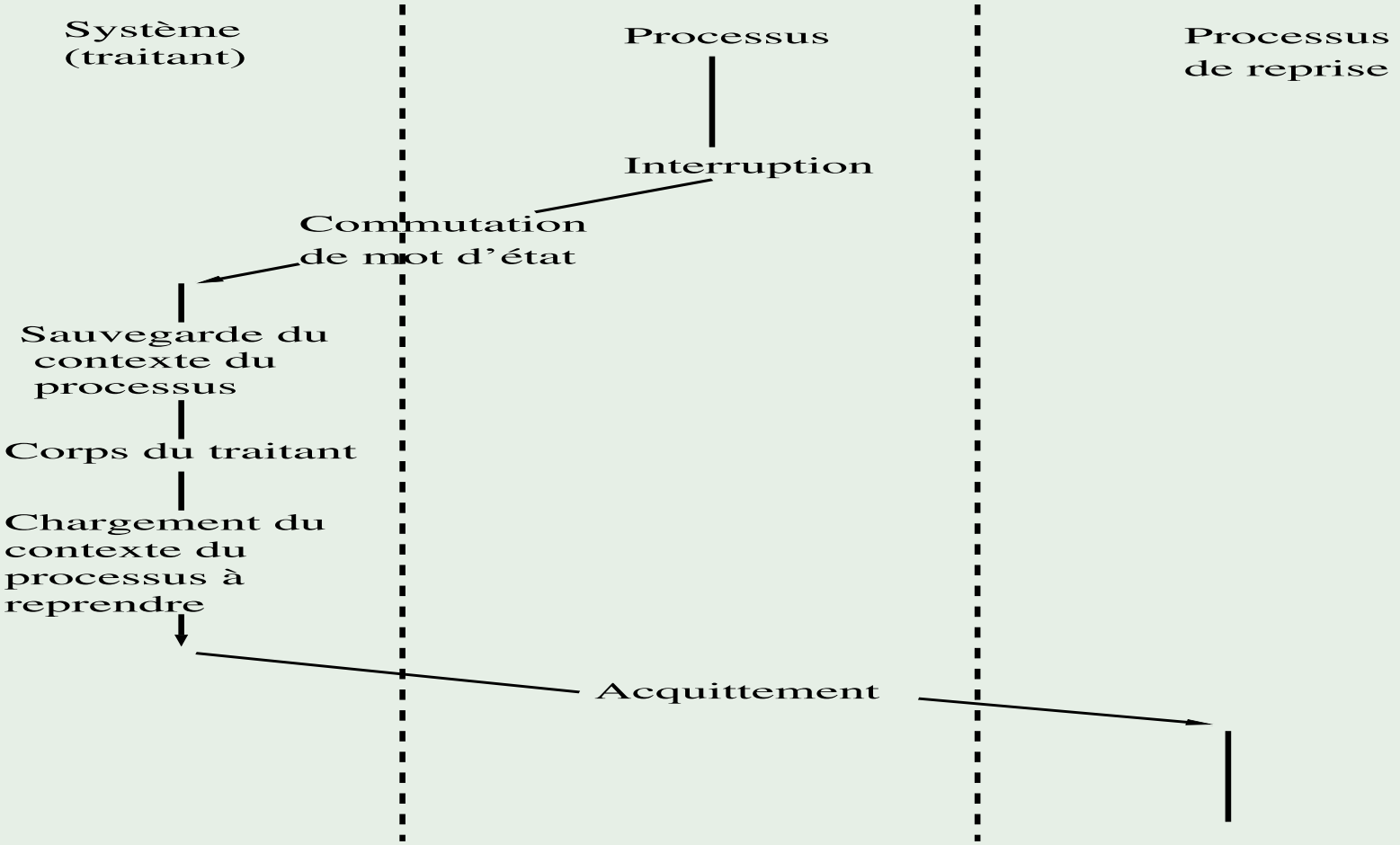
- à chaque cause est associé un niveau d'interruption
- le processus en cours d'exécution est suspendu ou mis en attente (état `bloqué`) et son contexte d'UC est sauvegardé
- le chargement d'un nouveau mot d'état provoque l'exécution d'un autre processus, appelé le traitant d'interruption

Les interruptions

- le traitant d'interruption réalise la sauvegarde si besoin, de la partie supplémentaire du contexte du processus interrompu
- après avoir exécuté les opérations liées à l'interruption, le traitant charge le contexte d'un processus, (pas nécessairement celui du processus interrompu) $\rightsquigarrow$ nouvelle commutation de contexte : acquittement

Les interruptions

Schématiquement



Les interruptions

où trouver le vecteur d'interruption ?

- adresse fixe en mémoire dictée par le fabriquant
- un registre contient l'adresse du vecteur

où sauvegarder le compteur ordinal ?

- adresse fixe en mémoire
- dans la pile
- dans une autre pile (pile système)
  présence éventuelle d'un deuxième pointeur de pile

Les interruptions

Niveau de priorité

- il peut exister plusieurs niveaux de priorité
- si plusieurs indicateurs d'interruption positionnés, le SE traite l'interruption de niveau le plus élevé

 le SE peut exécuter sans cesse des commutations de contexte si pendant l'exécution d'un traitant d'interruption, une seconde interruption se produit, puis une $3^{\text{ème}}$ ...

Les interruptions

Masquage des interruptions

on retarde une interruption,

- en masquant un niveau d'interruption. En pratique,
  - le SE positionne un indicateur qui fait partie du mot d'état de l'UC
  - cet indicateur est consulté avant toute prise en compte d'une interruption
- le démasquage du niveau autorise de nouveau la commutation du mot d'état du niveau considéré.

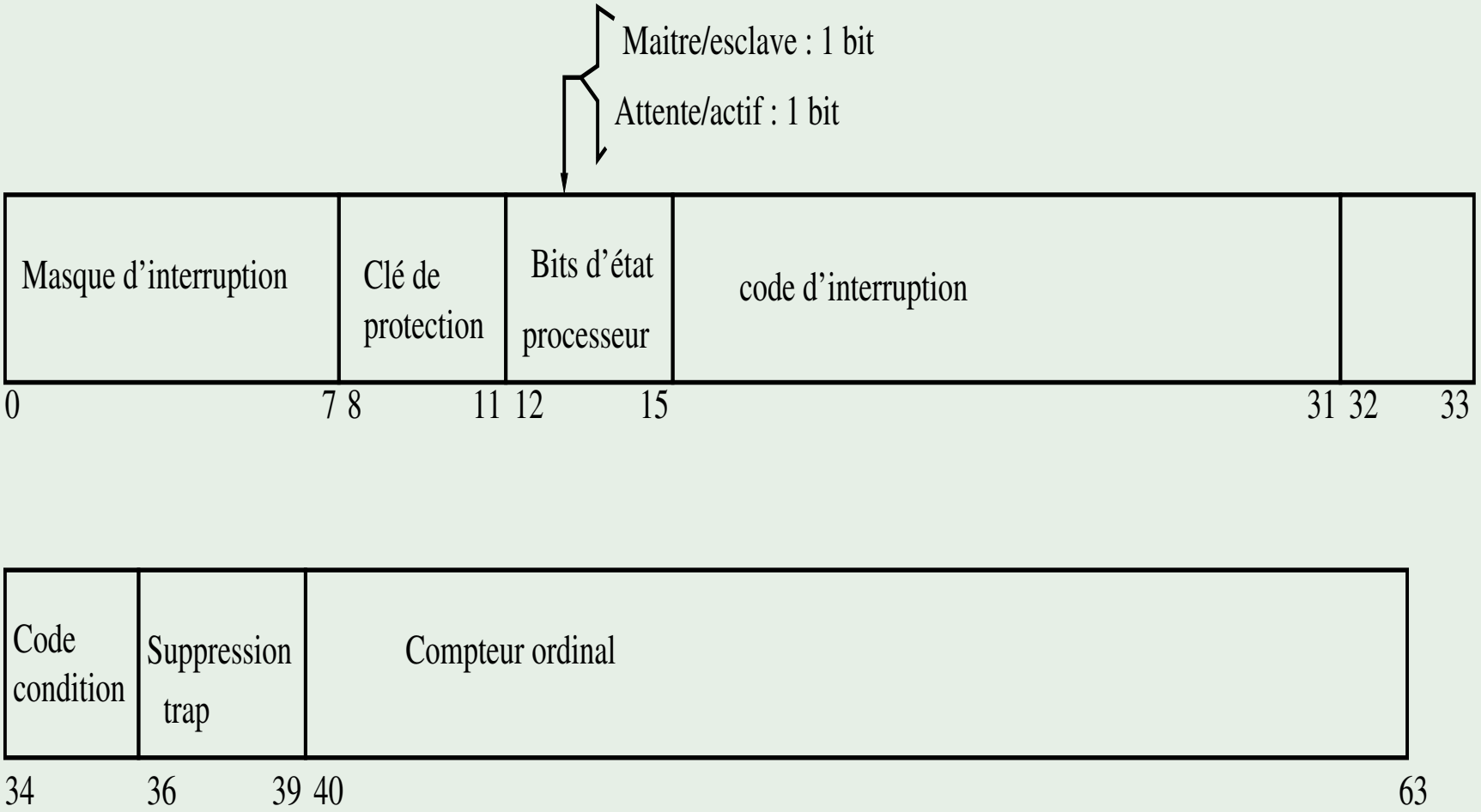
Les interruptions

Suppression des interruptions

Le niveau est désarmé.

Pour armer ou désarmer un niveau d'interruption, le SE modifie un indicateur dans le mot d'état

Registre du mot d'état sur l'IBM 360



Les appels système

Définition

- fonction primitive fournie par le noyau d'un SE et utilisée par les programmes s'exécutant dans l'espace utilisateur, i.e. tous les processus distincts du noyau
- permet de contrôler de façon sécurisée les applications dans l'espace utilisateur.

Les appels système

Comment se présentent-ils ?

peuvent être utilisés comme de simples fonctions écrites en C :

- sur la plupart des noyaux (notamment les noyaux monolithiques comme le noyau Linux) les appels systèmes sont implantés par une instruction machine (interrupt, supervisor call, ...) qui fait basculer le processeur dans le noyau en mode superviseur
- mode superviseur $\Rightarrow$ privilège : un simple utilisateur n'est pas forcément en mesure d'exécuter l'appel système
- un programme est limité à son espace d'adressage. Il n'a pas accès à certaines ressources de bas niveau

Linux a 300 appels système !

Les appels système

- la mise en œuvre des appels système nécessite un transfert de contrôle qui implique une interaction spécifique à l'architecture.
- une façon typique de mettre en œuvre l'appel système est d'utiliser une interruption logicielle
- l'interruption transfère le contrôle au noyau du SE de telle sorte que la partie logicielle a simplement besoin de mettre en place une correspondance avec le numéro de l'appel système nécessaire, et exécuter l'interruption logicielle

Les appels système

Un exemple : l'instruction `fork` en C

procède par clonage du processus père

- le processus fils reçoit une copie de l'environnement du processus père
- le processus fils commence son exécution à l'instruction qui suit `fork`

Les déroutements (trap)

Un déroutement

- se produit lorsqu'un programme effectue une opération interdite. Ex.
  - débordement de tableaux hors de la zone d'adresses allouée au programme
  - données incorrectes, (ex. division par zéro)
  - l'instruction ne correspond pas à un code exécutable
- peut être supprimé
  Ex. on peut autoriser la suite des opérations si l'erreur n'est pas trop grave : un arrondi à zéro *(underflow)* peut être légitime et il serait stupide d'arrêter le calcul
- ne peut pas être retardé. C'est un événement synchrone au programme en cours d'exécution pour lequel la notion de masque d'interruption ne s'applique pas
  Ex. si on décide de surveiller les underflows, le déroutement signale l'erreur. Cependant, le programme peut terminer

Les déroutements (trap)

Un déroutement

la gestion des déroutements est très variable d'un système d'exploitation à l'autre

$\rightsquigarrow$ un programme qui semble fonctionner correctement sur une machine peut se révéler erroné sur une autre

les choix par défaut sont souvent mal documentés $\rightsquigarrow$ peut amener des surprises !

Exemple 1 d'interruption

Une lecture sur disque dur

soit un programme qui lit des données sur un disque dur, les traite et les affiche à l'écran
Une version simple de l'algorithme :

```
répéter
  envoyer au contrôleur de disque une demande de lecture
      d'un bloc de données
  attendre tant que le disque ne répond pas (scrutation)
  traiter les données
  afficher les données
```

E/S par scrutation

Exemple 1

Une boucle de scrutation

```
répéter
    regarder si le transfert du disque est terminé
tant qu'il n'est pas terminé
```

Simple mais inefficace 

Exemple d'interruption

Exemple 1 :

Au lieu de répéter la boucle de scrutation, réaliser une autre tâche :

1. installer un traitant d'interruption disque qui traite les données reçues et les affiche
2. envoyer au contrôleur de disque une demande de lecture des données
3. faire autre chose

Dès que des données arrivent,

- le contrôleur de disque envoie une it au processeur, qui arrête temporairement la tâche 3 pour s'en occuper
- lorsque les données sont traitées, la tâche 3 reprend

pendant l'opération (lente) de lecture du disque, le processeur peut faire autre chose.

Exemple 2

Prélèvement de mesures :

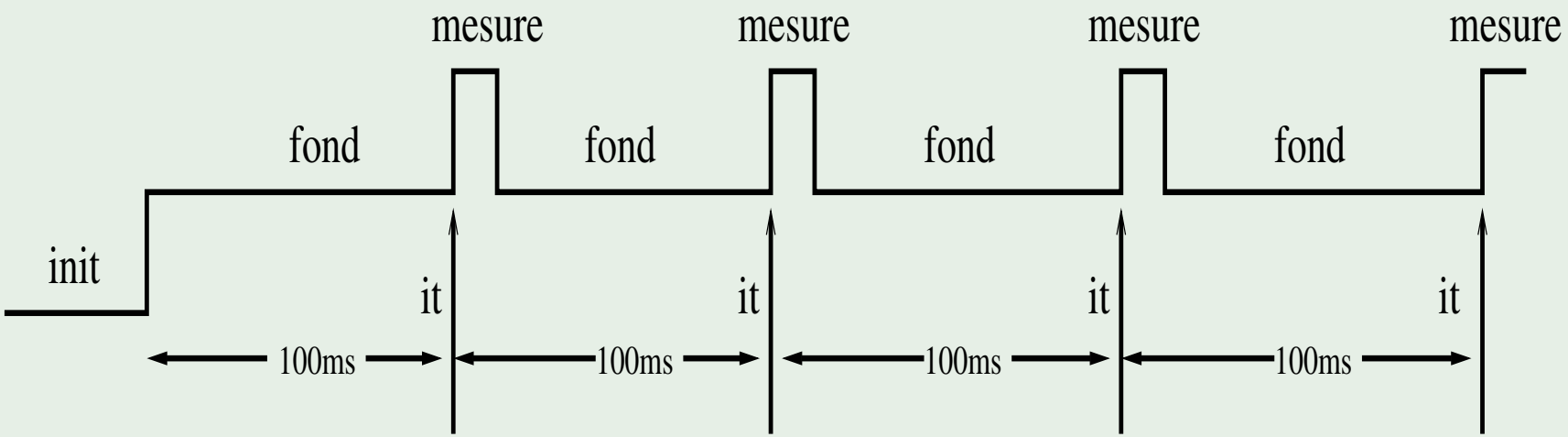
un ordinateur est chargé de relever périodiquement des mesures sur une installation industrielle

- la période de l'horloge $= 5\mu s$
- la prise de mesures doit être déclenchée toutes les $100ms$

$5\mu s \ll 100ms \Rightarrow$ le processeur sera occupé à l'exécution d'un travail de fond, qui sera périodiquement interrompu

Exemple 2

Prélèvement de mesures :



Exemple 2

Prélèvement de mesures :

```
const q = 20000  // 100ms / 5μs

procédure initialisation:
  début
      nouv_it_horl <- <actif, maître, masqué, adr it_horl>
      mep_trav <- <actif, esclave, démasqué, adr fond>
      mep_mesure <- <actif, esclave, démasqué, adr mesure>
      horloge <- q
      charger_mep(mep_trav)
   fin
```

Exemple 2

Prélèvement de mesures :

```
procédure it_horloge:
  début
      sauvegarder contexte  // sauvegarder contexte de fond
      mep_sauv <- anc_it_horl
      horloge <- q
      charger_mep(mep_mesure)
   fin
```

Le programme de `mesure` doit se terminer par un appel svc dont l'effet est de restaurer le contexte du travail de fond :

```
  restaurer contexte
  charger_mep(mep_sauv)
```

Exemple 2

Prélèvement de mesures :

Pourquoi ne pas inclure la procédure `mesure` au traitant d'interruption ?

```
sauvegarder contexte // sauvegarder contexte de fond
horloge <- q
mesure
restaurer contexte
charger_mep(anc_it_horl)
```

- fait exécuter le pg de `mesure` en mode maître et it masquées. Ne permet pas sa modification ni son remplacement par un user non priviliégié

Ordonnancement des processus

Scheduler (ordonnanceur)

programme du SE qui

- fait le choix du processus à exécuter lorsqu'il n'y a qu'un seul processeur et plusieurs processus dans l'état `prêt`
- en faisant un usage efficace de l'UC
  si les changements de processus sont trop nombreux, la consommation de temps processeur ↗
- même avec monopole du SE pour attribuer les ressources, on peut se retrouver dans une situation de blocage (étreinte fatale)

L'étreinte fatale

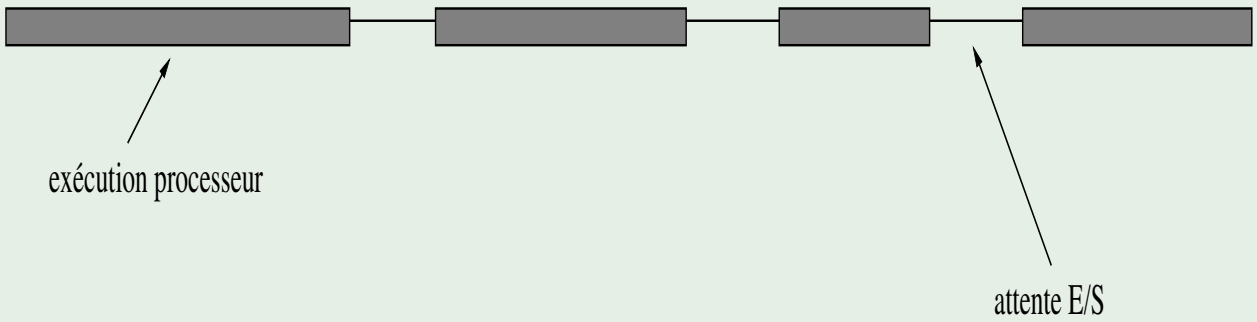
$P_1, P_2, \ldots, P_n$ en situation d'étreinte fatale si chaque P_i est bloqué en attente d'une ressource détenue par P_j différent

Processus P_1	Processus P_2
...	...
...	...
allocation ressource A	...
...	allocation ressource B
...	...
tentative allocation ressource B;	...
échec, blocage	...
...	...
...	tentative allocation ressource A;
...	échec, blocage
...	...
libération ressource A	...
...	libération ressource B

Ordonnancement des processus

2 types de processus

1. les processus de traitement



2. les processus d'E/S

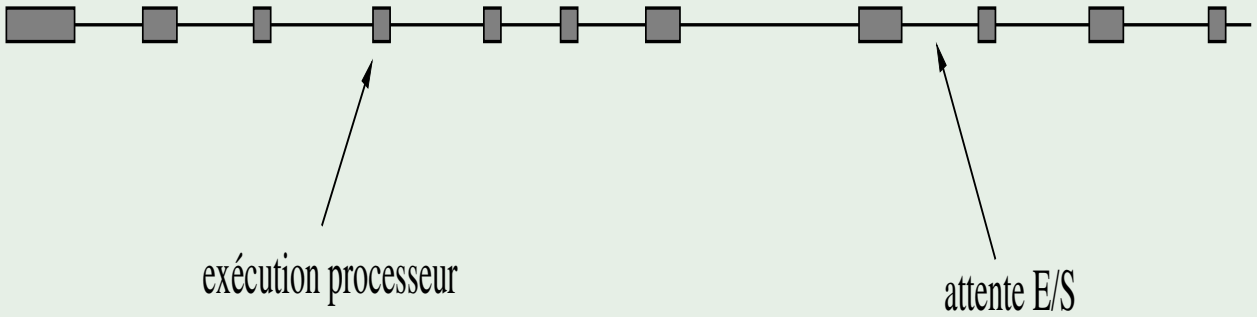
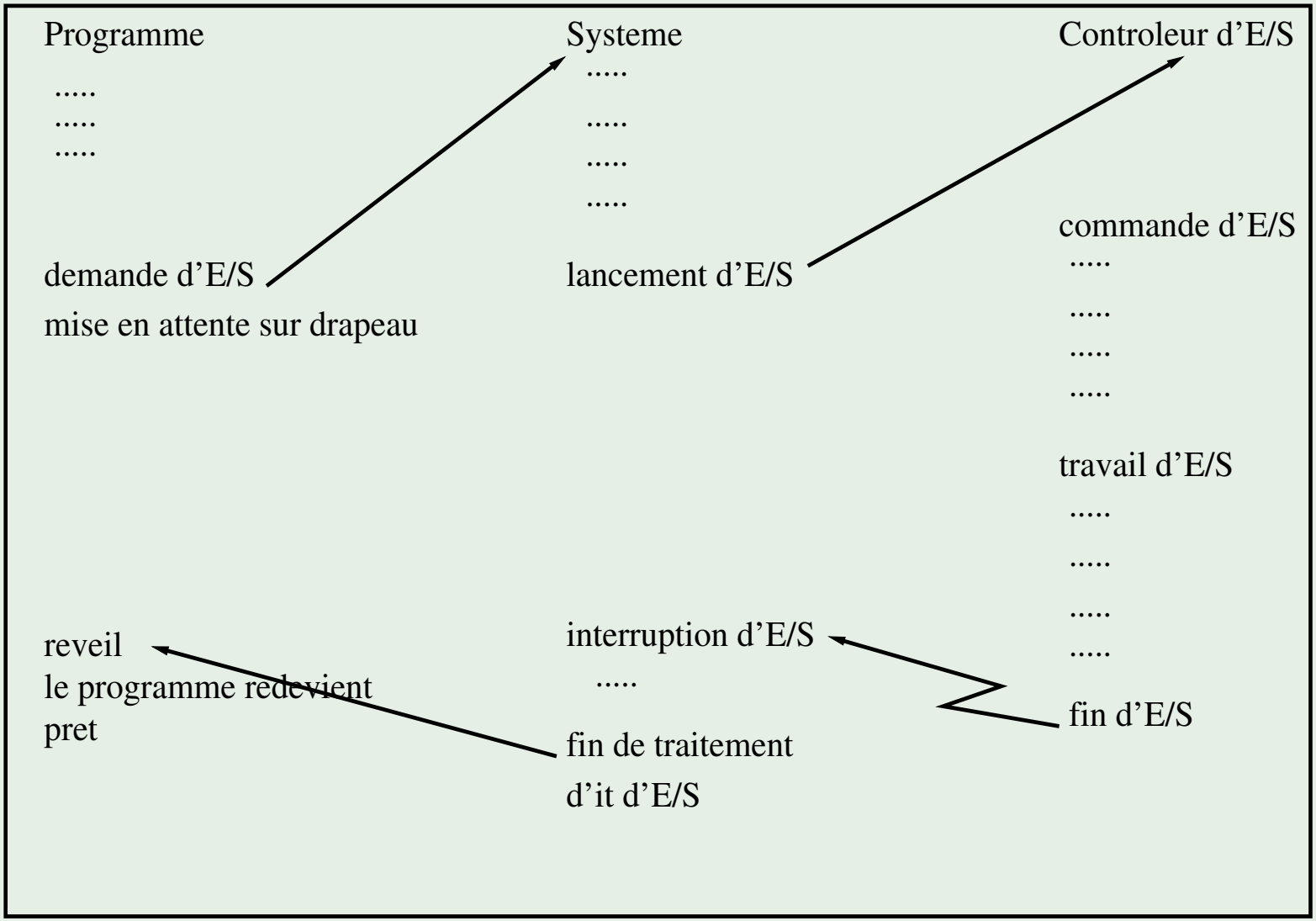


Diagramme d'une opération d'E/S



Ordonnancement des processus

Vers une priorité donnée aux processus d'E/S

- un programme qui fait beaucoup de calculs et très peu d'E/S risque de monopoliser l'UC
- les processus qui font beaucoup d'E/S se mettent en attente *volontairement*

Ordonnancement des processus

quand faut-il ordonnancer ?

- lorsqu'un nouveau processus est créé : exécuter le père ou le fils ?
- lorsqu'un processus se termine. Choisir dans la file des processus prêts. Si la file est vide, le SE exécute un processus d'inactivité
- quand un processus bloque sur des E/S, un sémaphore, ...
- lorsqu'une interruption d'E/S se produit pour indiquer la fin de son travail.
  Redonner la main au processus qui a provoqué l'E/S ou donner la priorité à celui qui était en cours d'exécution au moment de l'interruption, ou encore à un autre ?

Ordonnancement des processus

2 types d'algorithmes d'ordonnancement

- algorithme non préemptif
  - sélectionne un processus
  - le laisse s'exécuter jusqu'à ce qu'il bloque (E/S ou attente d'un autre processus) ou qu'il libère le processeur

  aucune décision d'ordonnancement pendant les interruptions d'horloge.
- algorithme préemptif
  - sélectionne un processus et le laisse s'exécuter pendant un délai déterminé
  - suspend le processus à la fin du délai, et en sélectionne un autre

  interruption à la fin du délai pour redonner le contrôle de l'UC à l'ordonnanceur

Ordonnancement des processus

Catégories d'algorithmes d'ordonnancement

Selon les types d'environnements

1. Traitements par lots
2. Interactifs
3. Temps réel

Algorithmes d'ordonnancement

- dans les systèmes de traitements par lots
  - algorithmes non préemptifs ou préemptifs avec un long délai pour les processus
    réduit le nombre de changements de processus $\rightsquigarrow$ améliore les performances
- dans les systèmes interactifs
  - essentiellement des algorithmes préemptifs, y compris pour les serveurs
     évite le blocage des processus par le processus actif en raison d'une anomalie dans son programme
- dans les systèmes temps réel
  - la préemption est parfois inutile !
    Les processus s'exécutent sur de courtes périodes

Ordonnancement des processus

Objectifs :

- équité : attribuer à chaque processus un temps processeur équitable
- équilibre : toutes les parties du système occupées
- application de la politique définie
- dans les systèmes par lots :
  - capacité de traitement : optimiser le nombre de jobs à l'heure
  - délai de rotation : réduire le délai entre la soumission et l'achèvement
  - utilisation de l'UC : processeur occupé en permanence
- dans les systèmes interactifs :
  - temps de réponse aux requêtes
  - proportionnalité : répondre aux attentes des utilisateurs
- dans les systèmes temps réel :
  - respect des délais
  - prévisibilité : éviter la dégradation de la qualité dans les systèmes multimédias

Algorithmes d'ordonnancement

Premier arrivé, premier servi (FIFO)

utilisé pour des tâches indépendantes ex. les systèmes de traitements par lots

- le temps processeur est attribué au processus selon son ordre d'arrivée
- une seule file d'attente de processus prêts
- lorsque le processus actif se bloque, le processeur est attribué au 1er processus de la file
- lorsque le processus bloqué redevient prêt, il est placé en queue de file d'attente

Algorithme FIFO

-  facile à mettre en œuvre
- avec un découpage du temps en unités, et combiné avec les priorités, 

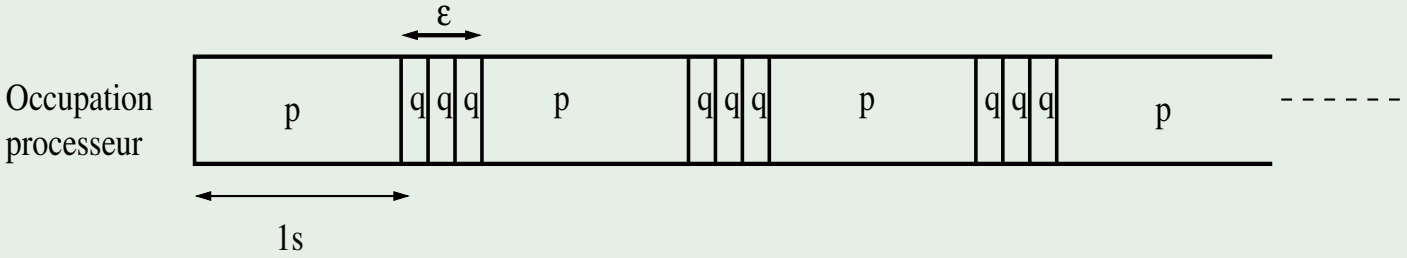
  les processus de traitement sont pénalisés s'ils cohabitent dans la file avec plusieurs processus d'E/S !
  - les *bons* processus (d'E/S) qui se seront interrompus volontairement verront leur priorité augmenter
  - les *mauvais* processus (de traitement) qui monopoliseront l'UC dans une tranche de temps verront leur priorité diminuer

  objectif : fuildifier la multiprogrammation

Exemple

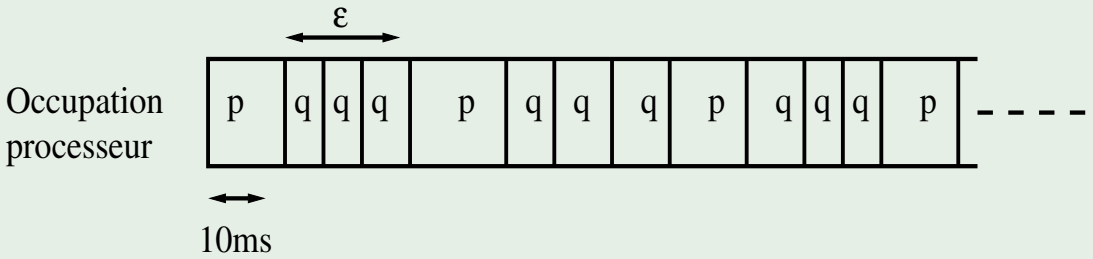
un processus p de traitement, s'exécute 1s puis se bloque en attente d'E/S. Trois processus d'E/S q1, q2, q3, effectuent 1000 lectures disque

- FIFO



temps de traitement total = 1bloc/s $\Rightarrow$ 1000s

- algorithme préemptif avec un délai de 10ms :

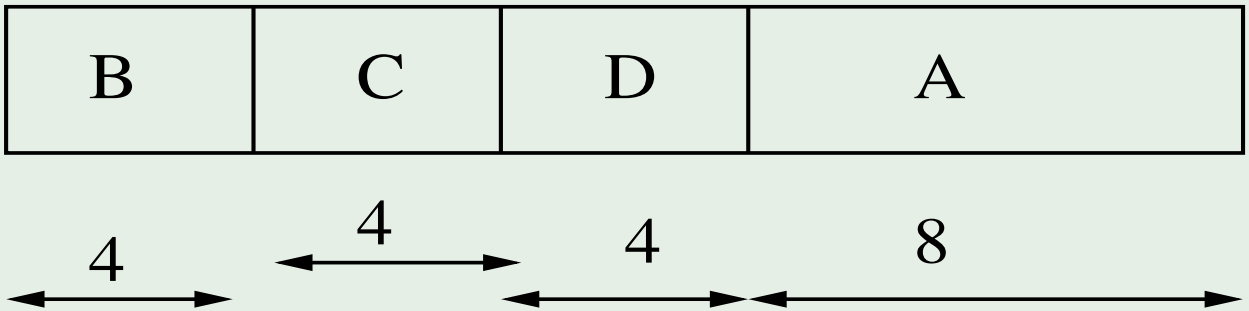


lecture d'un bloc toutes les 10ms. Temps de traitement total = 1000 * 10ms = 10^4 ms = 10s

Algorithmes d'ordonnancement

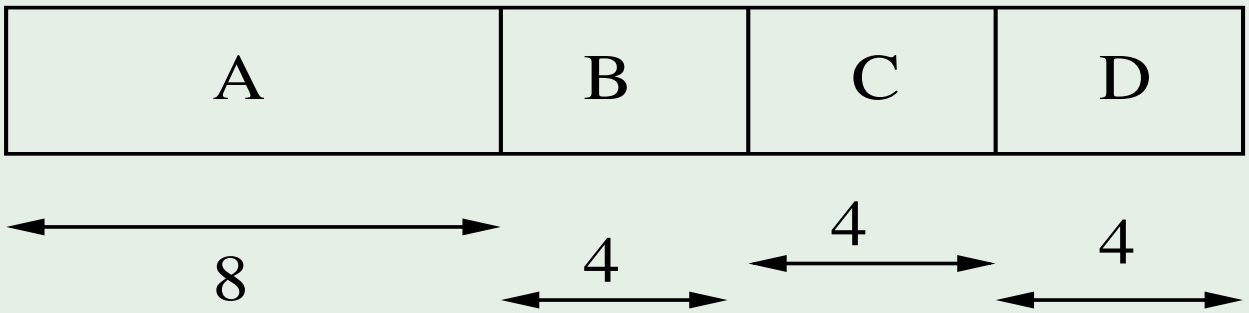
dans les systèmes de traitements par lots, où on suppose que les temps d'exécution sont connus à l'avance

- job le plus court (shortest job first ou PCTE)
  optimal si tous les processus sont disponibles en même temps



temps de rotation moyen = 11mn

- on compare au temps moyen si FIFO :



temps de rotation moyen = 14mn.

Job le plus court

Exemple

	A	B	C	D	E
temps exécution	2	4	1	1	1
temps arrivée	0	0	3	3	3

initialement, seuls A et B peuvent être exécutés

A	B	C	D	E

temps de rotation moyen = 4,6mn

pas la meilleure solution :

B	C	D	E	A

Temps de rotation moyen = 4,4mn

Diagramme de Gantt

Exemple

- le schéma

A	B	C	D	E

est représenté par le diagramme de Gantt

$$^{0}\,A^{2}\,B^{6}\,C^{7}\,D^{8}\,E^{9}$$

- et le schéma

B	C	D	E	A

par

$$^{0}\,B^{4}\,C^{5}\,D^{6}\,E^{7}\,A^{9}$$

$$\text{tps de rotation moyen} = \frac{\Sigma_i(\texttt{tps fin - tps arrivée})}{\text{nb processus}}$$

Algorithmes d'ordonnancement

- job dont le temps restant est le plus court (shortest remaining time next ou PCTER)
  - quand un processus arrive, son temps d'exécution est comparé au temps restant du processus actif. S'il est plus court, le processus actif est suspendu et le nouveau processus est lancé
  - favorise les processus courts

Algorithmes d'ordonnancement

Algorithmes préemptifs

- le tourniquet.
  - chaque processus se voit allouer le processeur pendant un quantum de temps déterminé
  - si le processus actif est bloqué ou a fini avant l'expiration du quantum de temps, l'UC est allouée à un autre processus
  - une liste circulaire de processus prêts

durée du quantum ?

Algorithmes préemptifs

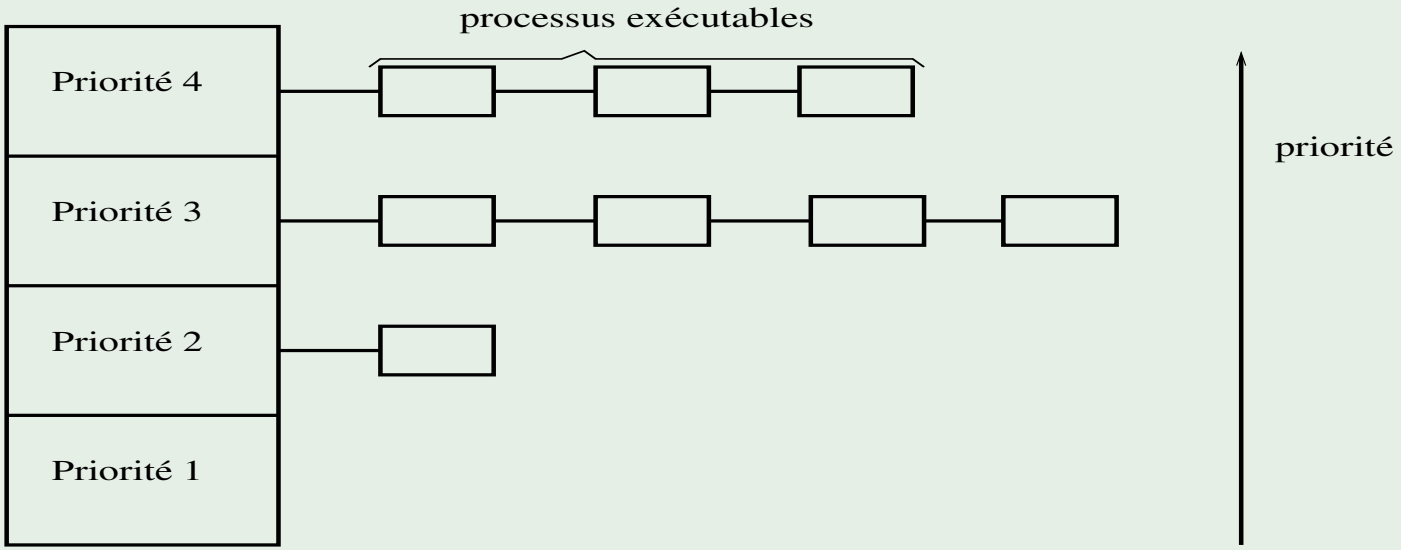
Le tourniquet

- si quantum de temps trop long, $\Rightarrow$
  - préemption impossible
  - temps de réponse aux requêtes simples trop long
- si quantum de temps trop court, $\Rightarrow$ gaspillage d'un pourcentage de temps non négligeable : trop de commutations de contexte
- si les processus prêts demandent des temps de calcul très différents, la valeur du quantum est difficile à fixer

conclusion compromis raisonnable autour de 20-50 ms

Files d'attente par priorité et tourniquet

les processus sont regroupés par catégorie de priorité. Les catégories sont organisées selon l'algorithme du tourniquet



Files d'attente par priorité et tourniquet

- tant qu'il reste des processus dans la catégorie 4, on en exécute un pendant un quantum selon le tourniquet
- si la catégorie 4 est vide, on exécute les processus de catégorie 3
- si les files 3 et 4 sont vides, on exécute un processus de catégorie 2 en tourniquet etc. . .

famine pour les catégories inférieures si les priorités ne sont pas réévaluées.

Algorithmes d'ordonnancement

Ordonnancement par priorités

à chaque processus est affecté (de façon statique ou dynamique) un niveau de priorité. Les processus sont exécutés suivant leur priorité

- le niveau de priorité est fixé une seule fois au processus. $\Rightarrow$ peut provoquer un phénomène de famine.
- la priorité évolue dynamiquement
  le système recalcule (réduit) la priorité du processus actif à la fin de chaque quantum, lorsque l'interruption d'horloge lui redonne le contrôle

les processus d'E/S ont souvent une priorité augmentée dynamiquement

Files d'attente par priorités

Tourniquet avec une politique de gestion des priorités.
Pour n niveaux de priorité :

1. Règle 1 :
   - Processus de priorité $n = 1$ quantum
   - Processus de priorité $n - 1 = 2$ quanta
   - Processus de priorité $n - 2 = 4$ quanta
   - Processus de priorité $n - 3 = 8$ quanta ...
2. Règle 2 : Quand un processus a utilisé tous ses quanta, il descend d'une catégorie.

Exemple :

- p a un temps de traitement de 100 quanta.
- Il obtient 1, 2, 4, 8, 16, 32, 64 quanta successivement pour s'exécuter complètement.
- 7 commutations de contexte suffisent (au lieu de 100 avec l'algorithme du tourniquet).

Algorithmes d'ordonnancement

Ordonnancement par tirage au sort (lottery scheduling)

- Des *billets de loterie* sont alloués aux processus.
- Quand il faut affecter le processeur à un processus, un billet est choisi au hasard.
- Le système a la capacité de tirer un billet 50 fois/seconde $\rightsquigarrow$ un processus peut récupérer 20 ms de temps processeur.
- Les processus les plus importants peuvent recevoir des billets supplémentaires $\Rightarrow$ un processus possédant une fraction f des billets obtiendra une fraction f du processeur.

Ordonnancement par tirage au sort

- Des processus collaboratifs peuvent échanger des billets si besoin. Ex.
  - Un processus client envoie un message au processus serveur et se bloque $\rightsquigarrow$ il remet ses billets au serveur ;
  - quand le serveur a terminé, il rend les billets au client.

  Permet de résoudre des problèmes difficiles à prendre en charge par les autres algorithmes  .
  Exemple : serveur vidéo avec des processus à débit différents $\rightsquigarrow$ les processus se partagent le temps processeur dans une proportion adaptée à leur débit.

Ordonnancement équitable

Point de vue de l'utilisateur.

- user_1 : 9 processus, user_2 : un processus $\rightsquigarrow$ 90% temps processeur à user_1 et seulement 10% à user_2 !
- Alternative : 50% du temps processeur à chacun, $\forall$ le nombre de processus lancés par l'un et par l'autre.
  Ex. user_1 : 4 processus `A, B, C, D`, user_2 : `E`
  `A E B E C E D E A E B E C E D E...`
- Variante : une proportion de temps processeur pondérée par le nombre de processus lancés.
  Ex. 2 fois plus de temps à user_1 qu'à user_2
  `A B E C D E A B E C D E...`

Algorithmes d'ordonnancement

Ordonnancement des systèmes temps réel

Le temps est essentiel.

- Temps réel à tolérance 0 : impérativement respecter son délai.
- Temps réel avec tolérance : quelques échecs restent tolérables.

Dans tous les cas, processus généralement très courts. 2 catégories d'événements :

1. périodiques
2. apériodiques.

Événement périodiques

périodicité P_i, temps processeur C_i exprimé en secondes. Le système peut être ordonnancé (peut traiter la charge) si

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Ex.

	p_1	p_2	p_3
période	100	200	500
temps d'exécution (ms)	50	30	100

Le système peut être ordonnancé : $0,5 + 0,15 + 0,2 < 1$

Les algorithmes d'ordonnancement pour le temps réel peuvent être statiques ou dynamiques.