Chapitre 01 : Gestion des Processus

Dr Mandicou BA
mandicou.ba@esp.sn
http://www.mandicouba.net

Diplôme Universitaire de Technique (DUT, 2^e année)
Diplôme Supérieure de Technologie (DST, 2^e année)
Option : **Informatique**

ECOLE SUPERIEURE POLYTECHNIQUE

www.esp.sn

Plan du Chapitre

Sommaire

Partage de l'unité centrale

Mono-programmation

- Un seul programme (processus) s'exécute sans interruption :
  - Si le processus contient une instruction d'E/S, restera inactif durant une « longue période » en attendant que cette instruction se termine

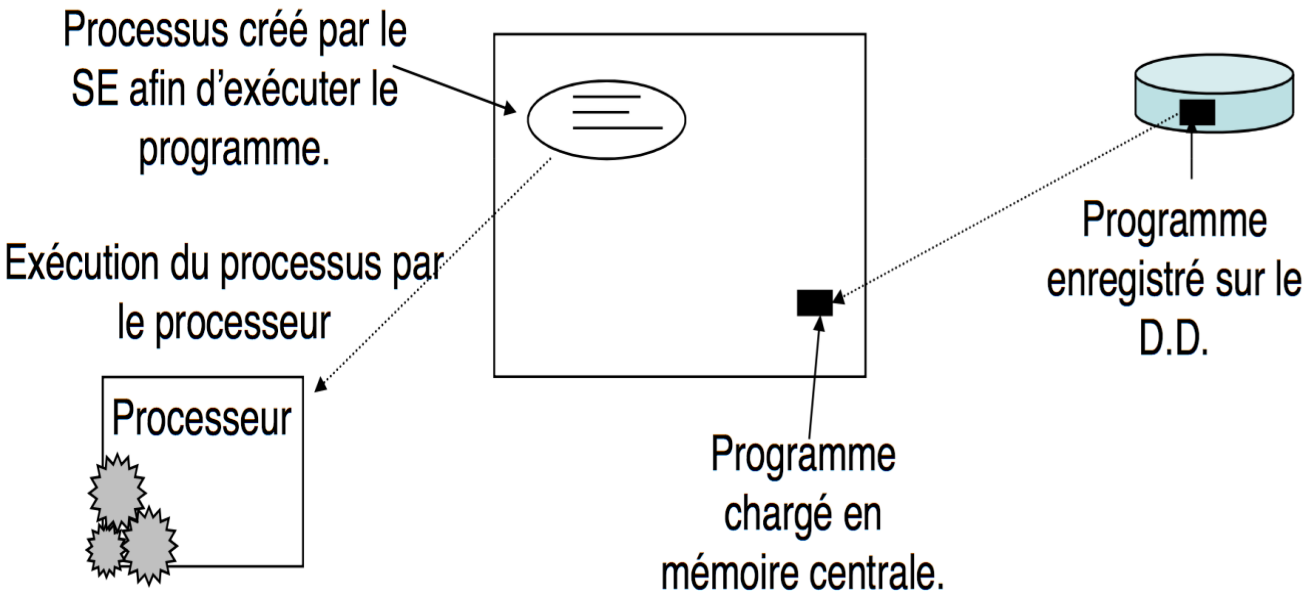
Multi-programmation

- Plusieurs programmes (processus) se partagent les ressources (mémoire, périphérique, etc.) de l'ordinateur :
  - problème de protection, de concurrence et contrôle
- Le processeur exécute un autre processus au lieu de restant inactif pendant tout le temps pris par l'instruction d'E/S du 1^e processus
- Cela donne à l'utilisateur que tous les processus s'exécutent en même temps : pseudo-parallélisme
  - sur une machine mono-processeur, un seul processus est exécuté à un instant donné

Définition d'un processus

Definition

- ☛ Processus = instance d'exécution d'un programme crée par le SE ou l'utilisateur
  - ➠ il possède son compteur ordinal, ses registres et ses variables en mémoire

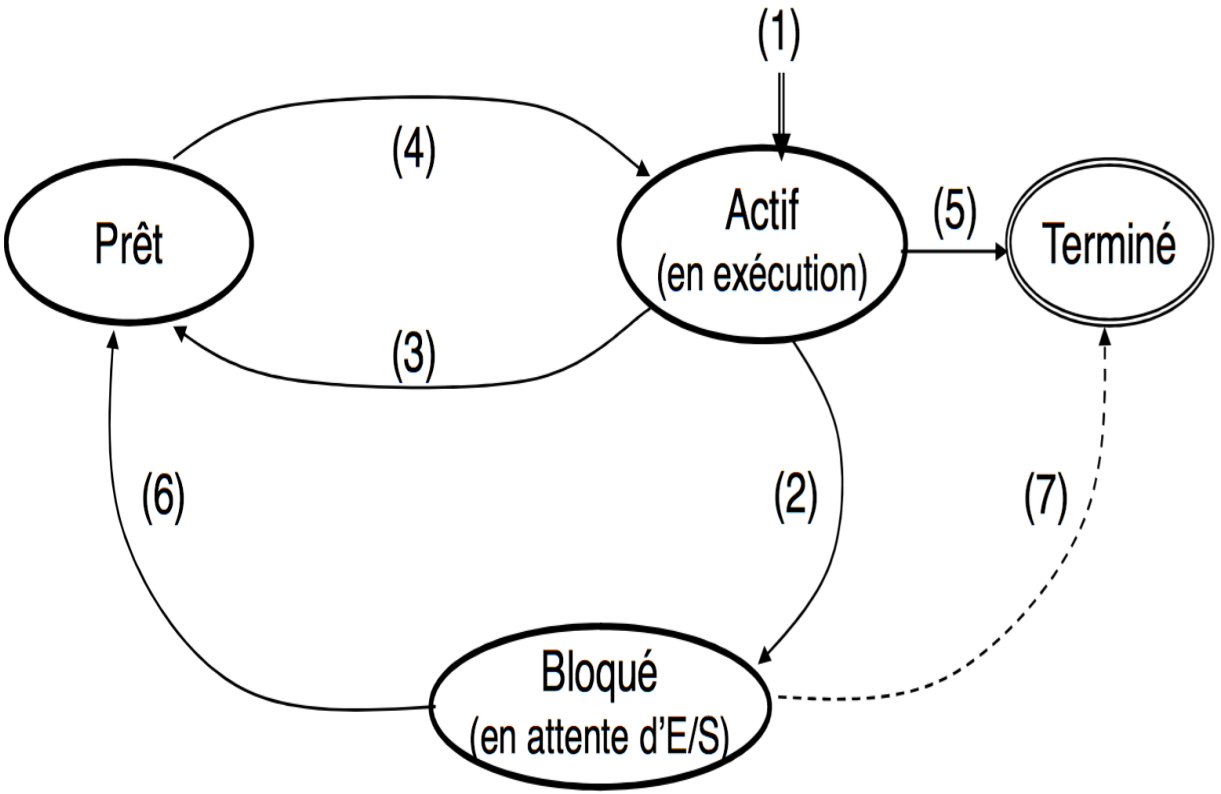


Rôle du SE dans la gestion des processus

- Création, suppression et interruption
- Ordonnancement des processus : *exécution équitable entre processus tout en privilégiant les processus systèmes*
- Synchronisation des processus :
  - Choisir le processus à exécuter à un instant donné
  - Choisir le moment où interrompre un processus
  - Choisir le processus qu'il faut exécuter ensuite (le suivant)
  - Spécifier les ressources dont à besoin (et qu'il faut affecter à) un processus
- Gestion des conflits d'accès ressources partagées
- Protection des processus d'un utilisateur contre les actions d'un autre processus
- etc.

Cycle de vie d'un processus (1/2)

- ☛ Processus = suite d'instructions. On peut l'exécuter et l'interrompre :
  - ➠ Peut se trouver dans plusieurs états (actif, suspendu, terminé, en attente d'un événement)



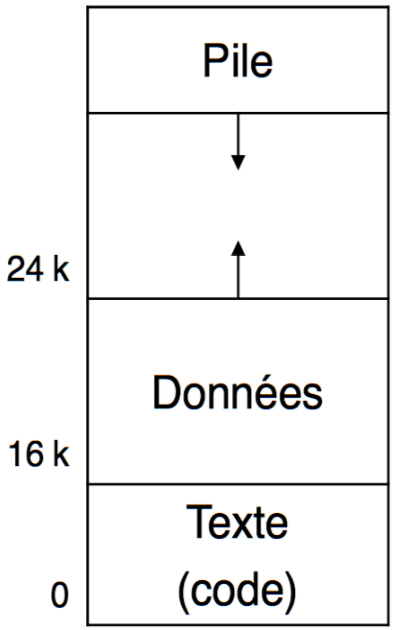
Cycle de vie d'un processus (1/2)

1. Naissance d'un processus
2. Transition « **actif** $\longmapsto$ **bloqué** » : se produit quand le processus est en cours d'exécution et a besoin d'une ressource non disponible.
3. Transition « **actif** $\longmapsto$ **prêt** » : se produit si le **quantum** alloué est épuisé ou si un processus plus prioritaire (processus urgent ou processus système) arrive.
4. Transition « **prêt** $\longmapsto$ **actif** » : se produit quand le SE sélectionne le SE en question pour l'exécuter.
5. Transition « **actif** $\longmapsto$ **terminé** » : se produit quand le processus a fini son exécution.
6. Transition « **bloqué** $\longmapsto$ **prêt** » : se produit quand l'événement externe attendu par le processus se produit.
7. Transition « **bloqué** $\longmapsto$ **terminé** » : se produit quand l'événement externe attendu par le processus ne peut se réaliser (ex. inter-blocage).

Espace mémoire d'un processus (1/5)

☛ Le mémoire utilisée par un processus est divisée en plusieurs zones, exactement 4.

1. Segment de code
2. Segment de données
3. Pile
4. Tas

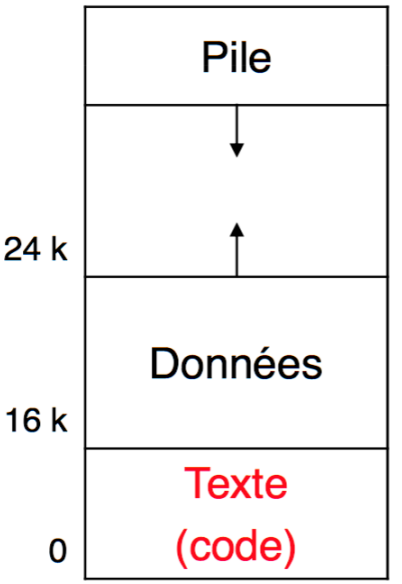


Structure de l'espace d'adressage d'un processus

Espace mémoire d'un processus (2/5)

Sagement de code

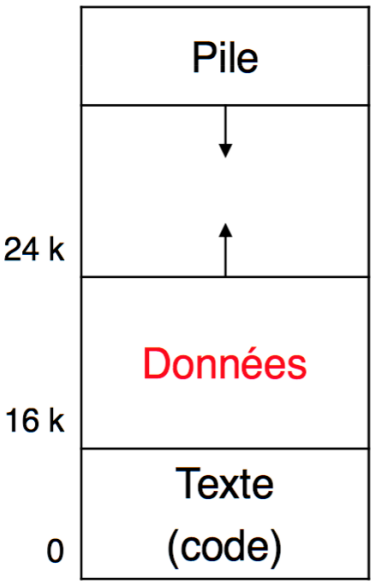
- Copie du segment de code du fichier exécutable
- Placé dans des zones fixes de la mémoire (début de la zone disponible)
- La prochaine instruction à exécuter dans ce segment est repérée par le pointeur d'instruction
- Cette zone est en lecture seule
- Elle peut être partagée par tous les programmes exécutant le même programme. Ce qui n'est pas le cas pour les segment de données et de pile



Espace mémoire d'un processus (3/5)

Sagement de données

- Il se trouve au dessus du segment de code
- Il est amené à grandir ou à rétrécir durant l'exécution
- il est composé de :
  1. **Un segment de données initialisées** : copié directement de l'exécution. Par exemple, les données initialisées correspondent aux variables globales et static initialisées d'un programme C
  2. **Un segment de données non initialisées** : crée dynamiquement. Les données non initialisées correspondent aux variables globales et static non initialisées



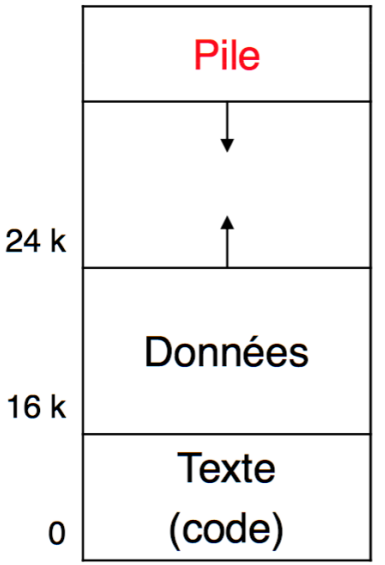
Espace mémoire d'un processus (4/5)

Pile

- Sert à stocker les données obtenues en cours d'exécution. Son nom, **pile**, (*stack en anglais*) vient de la manière dont elle est géré : empiler puis dépiler les données
- Le plus souvent en haut de l'espace d'adressage et croit vers le bas

Exemple 1 (Appel d'une fonction)

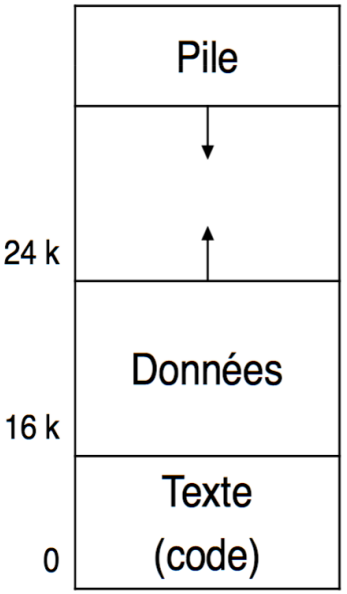
- *Empiler le nom de fonction, lui passer les paramètres et variables locales*
- *Exécuter la fonction. Une fois la fonction terminée, le SE dépile les données utilisées et retrouve les données d'avant*
- *Poursuivre l'exécution du programme*



Espace mémoire d'un processus (5/5)

Tas

☛ Est un autre segment utilisé par le système d'exploitation pour les allocations dynamiques.



Interruption d'un processus

Interruption

- Dans le cas des transitions « **actif** $\longmapsto$ **bloqué** » et « **actif** $\longmapsto$ **prêt** » on parle d'**interruption (IT)**

D'où viennent les IT

- Quand le processus atteint une instruction d'E/S
- Le quantum attribué au processus est écoulé
- Un processus plus urgent doit être exécuté
- Un processus nécessite une ressource (matérielle ou logicielle) ou une donnée (un résultat calculé par un autre processus, ou un ensemble d'instructions qui ne sont pas encore chargées en mémoire) détenue par un autre processus (elle n'est pas encore disponible)

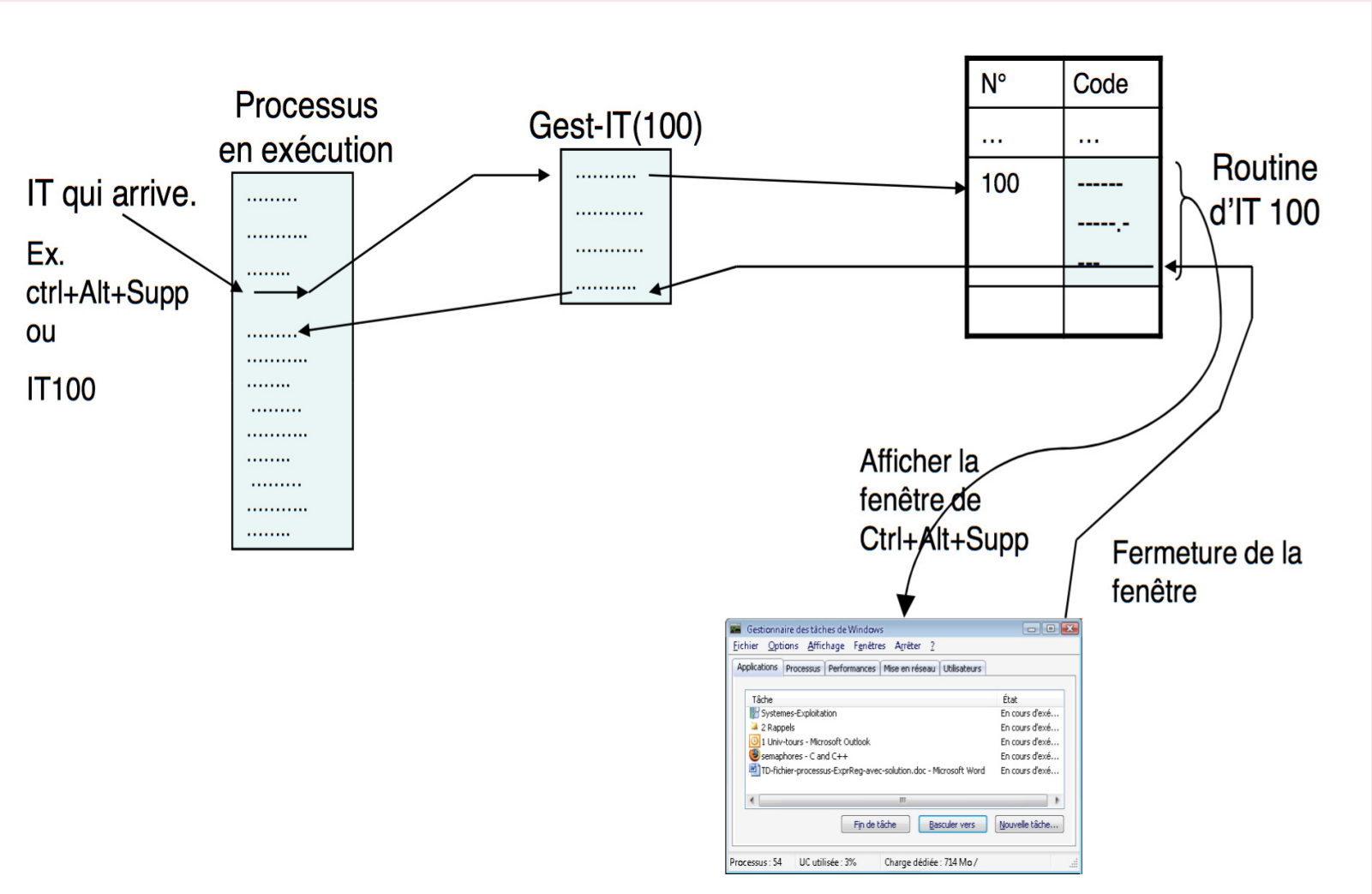
Interruption d'un processus

Traitement d'une IT

1. Arrivée d'une IT : Le processus en cours est interrompu et un gestionnaire d'IT est chargé dans les registres du processeurs et s'exécute pour traiter l'IT en question
2. Une fois le signal IT reconnu, le gestionnaire d'IT accède à la table des vecteurs d'IT et recherche l'adresse du programme associé et l'exécute
3. Une fois l'IT traitée, le SE charge un autre processus à partir de la file d'attente et l'exécute

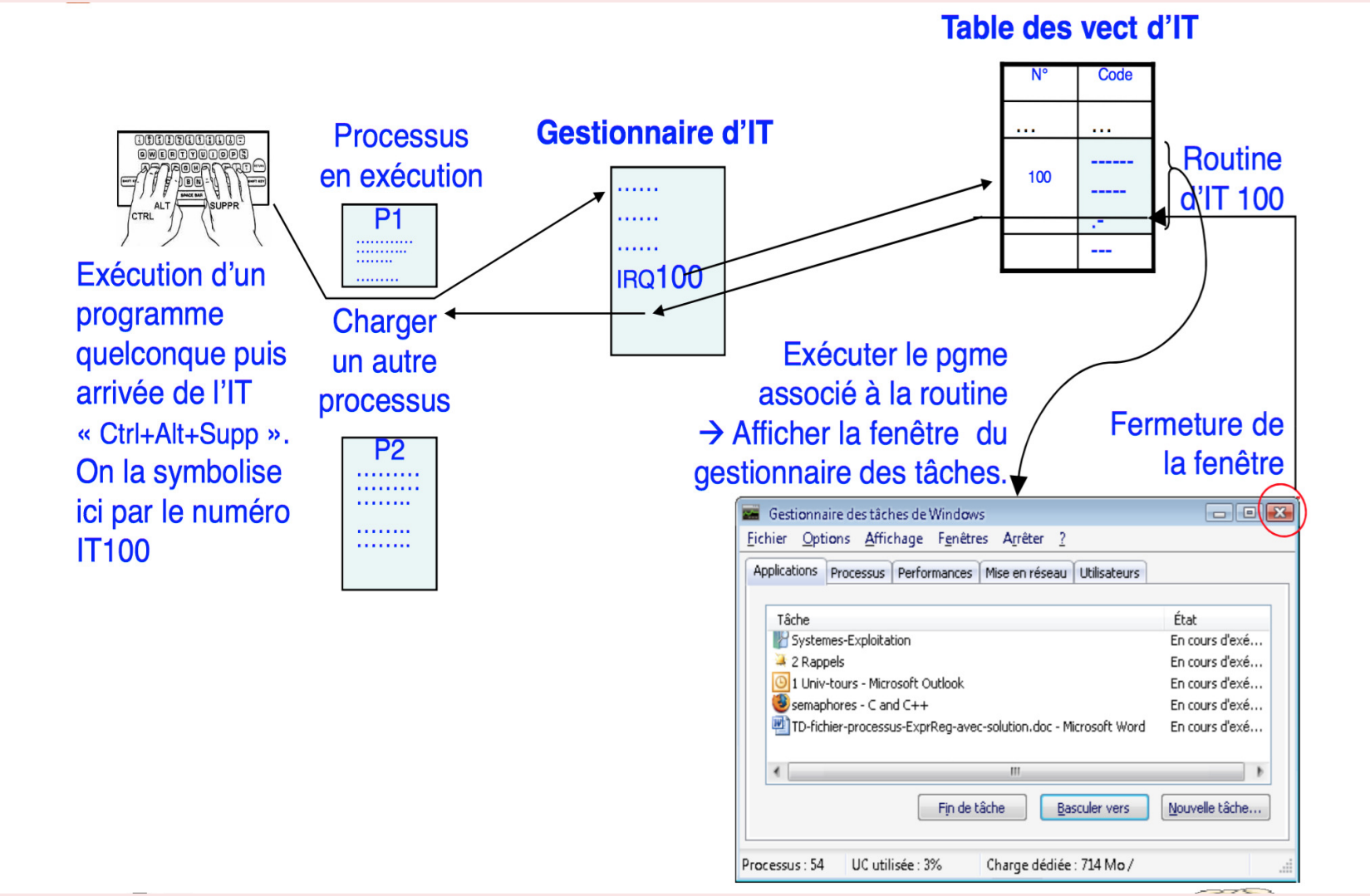
Interruption d'un processus

Traitement d'une IT : exemple



Interruption d'un processus

Traitement d'une IT : exemple



Interruption d'un processus

Traitement d'une IT

1. Une IT est provoquée par un signal généré soit par un événement interne soit par un événement externe :
   - ☛ **Événement interne** : lié au processus
     - Appel système
     - Déroutement : dû généralement aux erreurs telles que division par zéro, débordement de la mémoire, exécution d'une instruction non autorisée, etc
   - ☛ **Événement externe** : panne, intervention de l'utilisateur à l'aide d'une frappe au clavier. C'est l'exemple de «Ctrl+Alt+supp», bouton «reset», etc.

Interruption d'un processus

Traitement d'une IT

1. Deux sortes d'interruptions : Matérielles et Logicielles
   - **IT Matérielles (IRQ)** : générées par les périphériques. Parviennent au processeur par l'intermédiaire d'un contrôleur d'IT
   - **IT Logicielles** : des IT internes, c'est le processus qui appelle cette IT à l'aide du numéro d'IT.
     - : Par exemple, pour appeler une IT DOS, appeler l'IT N°21H
   - Si plusieurs interruptions arrivent au même temps, alors celle qui a le plus petit numéro qui a la plus grande priorité :
     - Exemple : IRQ horloge système = 0, IRQ port parallèle = 7

Sommaire

Structures de données utilisées

☛ Pour générer un processus, le SE manipule deux structures de données :

1. Le **bloque contexte** d'un processus
2. La **table des processus**

Contexte d'un processus

- Structures de données qui décrivent un processus en cours d'exécution
- Créées au même temps que le processus et sont mises à jours lors d'une IT d'un processus
- Informations sauvegardées par le SE lors d'une IT d'un processus
- Les données d'un contexte d'un processus sont : 6
  - le compteur ordinal : adresse la prochaine instruction
  - le contenu des registres généraux
  - les registres d'occupation mémoire
  - le registre de variable d'état
  - la valeur d'horloge d'un processus
  - la priorité du processus
- Lors de l'IT d'un processus, le SE sauvegarde le contexte de processus en cours et charge celui du processus à exécuter : **commutation de processus** (changement de contexte)

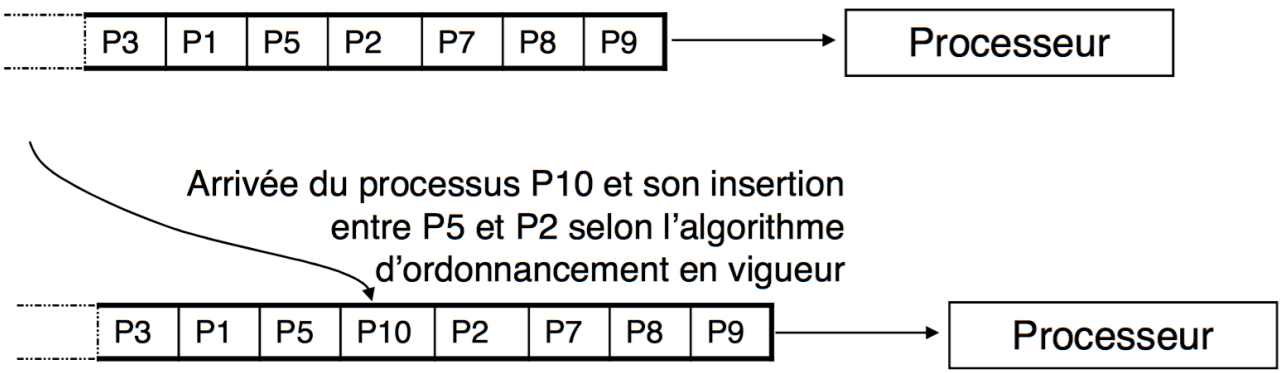
Table des processus

- ☛ Tout processus contient une entrée dans cette table
- ☛ Cette table contient toutes les informations indispensables au SE pour assurer une gestion cohérente des processus
- ☛ Ces informations sont :
  - Un pointeur vers le bloc de contexte du processus
  - l'identifiant du processus
  - son lien de parenté
  - les fichiers qu'il a ouvert
  - Occupation mémoire (pointeur sur le segment de code, de données et de pile)
- ☛ Cette table est stockée dans l'espace mémoire du SE
  - ☛ Donc **aucun processus ne peut y accéder**

Sommaire

Principe

- Plusieurs processus peuvent se trouver dans un état « prêt » (ou en « attente »)
- Le SE les place dans une file d'attente : une file pour chaque état
- Le SE dispose d'un programme qui choisit le processus à exécuter : **scheduler, dispatcher ou ordonnanceur**



Algorithmes d'ordonnancement

- Parmi les processus de la file d'attente, lequel choisir :
  - **Algorithme d'ordonnancement**
- Optimisation : améliorer le temps de réponse moyen (moyenne des date des fin d'exécution) du système, le temps d'attente (moyenne des délais d'attente pour commencer une exécution)

Quelques algorithmes d'ordonnancement

- Ordonnancement selon FIFO
- Ordonnancement circulaire (Round Robin ou tourniquet)
- Ordonnancement avec priorité
- Ordonnancement selon le plus court job
- Ordonnancement selon le job le plus court qui reste

Critères d'ordonnancement des processus

- ☛ L'utilisation intensive du processeur :
  - Le système perd son efficacité si le processeur passe trop de temps à attendre des E/S
- ☛ L'équité :
  - Tous les processus doivent avoir la possibilité d'utiliser le processeur
  - Nonobstant les priorités parfois indispensables, des processus comparables doivent être traités avec les mêmes égards.
- ☛ Utilisation répartie de l'ensemble des ressources :
  - En ne tenant compte que de l'utilisation optimale du CPU, on risque de sous-utiliser momentanément d'autres ressources que les processus devront se disputer ensuite.
  - Une bonne stratégie consiste donc à évaluer et à bien répartir l'emploi de l'ensemble des ressources.

Critères d'ordonnancement des processus

- Le nombre de processus par unité de temps :
  - Ce critère dépend cependant de la longueur des processus.
- La durée de rotation :
  - Délai moyen entre l'admission du processus et la fin de son exécution.
- Le temps d'attente :
  - Temps moyen qu'un processus passe à attendre
  - Il s'obtient en soustrayant la durée d'exécution du processus de sa durée de rotation
  - Cette durée est indépendante de la durée d'exécution du processus lui-même.
- Le temps de réponse :
  - Vitesse de réaction aux interventions extérieures. Les programmes d'avant-plan doivent pour cela avoir priorité sur les tâches de fond

Critères d'ordonnancement des processus

☛ La prévisibilité :

- Un système qui d'habitude réagit rapidement aux commandes mais qui parfois prend un temps beaucoup plus long sera perçu comme moins stable que s'il répondait à chaque fois dans un temps comparable même s'il est globalement plus lent.
- Le système semblera aussi plus convivial s'il respecte l'idée parfois fausse que les utilisateurs se font de la complexité des tâches.

Fist-come First-served - Premier arrivé, Premier servi

- Le premier arrivé est le premier servi
- placer le nouveau processus qui arrive à la fin du liste
- Avantage :
  - Simple,
  - Ne consomme pas de temps processeur
- Inconvénient :
  - Comme si on fait de la mono-programmation
  - Problème avec les processus prioritaires
  - Le processeur peut être trop occupé par un gros travail

RR : Round Robin - L'algorithme du tourniquet

- Allouer à chaque processus un temps d'exécution q (quantum)
- Une fois qu'un processus a consommé son quantum, il est interrompu et mis à la fin de la file d'attente
- L'ordonnanceur sélectionne le prochain (premier) processus de la file d'attente et l'exécute pendant le même quantum de temps
- Avantage :
  - Tous les processus ont la chance (la même chance) d'être exécuté
- Inconvénients :
  - Si le temps q est faible et comparable à celui du chargement du contexte, l'ordonnanceur passe plus de temps à changer et à décharger des processus qu'à les exécuter
  - Problème avec les processus prioritaires

L'ordonnancement avec priorité

- Le processus de plus haute priorité est exécuté le premier
  - Les processus sont classés dans l'ordre décroissant de leur priorité
- Chaque processus s'exécute jusqu'à la fin
- Une priorité peut être fixe ou variable, static ou dynamique
- Avantages :
  - Pouvoir privilégié certains processus. Par exemple exemple, les processus systèmes (ceux du SE) ont la plus haute priorité
- Inconvénients :
  - **Famine** : processus de haute priorité monopolises l'UC, ceux de faible priorité ne s'exécute pas
  - Solution : décrémenter de 1 la priorité du processus à chaque impulsion d'horloge ou à chaque quantum q et le mettre (après avoir exécuter son temps q) à la fin de la file de priorité inférieure

SJF : Shorted Job First - le job le plus court d'abord

- ☛ C'est le plus court (du point de vu temps d'exécution) processus qui est mis à la tête de la file d'attente et qui est exécuté en premier
- ☛ Avantage :
  - Utile dans les chaînes de production
- ☛ Inconvénient :
  - Suppose la connaissance de la durée d'exécution de tous les processus, ce qui est rare en pratique sauf si on a utilisé au préalable au moins une fois ces processus

Conclusion sur l'ordonnancement

- ☛ Pas d'algorithme idéal sur tous les plans
- ☛ Les critères de choix dépendent des besoins et des attentes

Sommaire

Accès concurrents

- Le problème des accès concurrents se produit quand deux processus partagent une ressource, matérielle ou logicielle, alors que celle-ce ne peut pas être partagée (accès exclusif).
  - L'accès en lecture ne pose de problème contrairement aux accès en écriture

Exemple 2

- *P_1 et P_2 sont deux ressources voulant accéder à une imprimante*
- *L'imprimante est gérée par le processus deamon-plt qui inspecte :*
  1. *une variable de type tableau contenant les fichiers à imprimer*
  2. *et 2 variables entières prochain (qui indique numéro du prochain fichier à imprimer) et libre (qui indique le premier emplacement libre où déposer le fichier)*
- *Supposons le scénario suivant :*

Accès concurrents

Exemple 3

- *P_1 et P_2 sont deux ressources voulant accéder à une imprimante*
- *L'imprimante est gérée par le processus deamon-plt qui inspecte :*
  1. *une variable de type tableau contenant les fichiers à imprimer*
  2. *et 2 variables entières prochain (qui indique numéro du prochain fichier à imprimer) et libre (qui indique le premier emplacement libre où déposer le fichier)*
- *Supposons le scénario suivant :*
  1. *P_1 lit libre, la trouve à 5 puis est immédiatement interrompu*
  2. *P_2 lit libre, la trouve à 5 puis met son fichier à cet emplacement, incrémente libre à 6 et est ensuite interrompu*
  3. *P_1 reprends et écrase le fichier de P_2 par le sien*
- *Solution : ne pas interrompre P_1 au moment où il a été interrompu*

☛ *Notions de **section critique** et **exclusion mutuelle***

Section critique

Définition

- Une ressource est dite **ressource critique** lorsque des accès concurrents à cette ressources peuvent résulter dans un état incohérent
- On parle aussi de situation de compétition (race condition) pour décrire une situation dont l'issue dépend de l'ordre dans lequel les opérations sont effectuées
- Une section critique est une section de programme manipulant une ressource critique.
- **Section critique** = partie du processus où il peut y avoir un conflit d'accès
  - Contient des variables et/ou ressources partagées

Exclusion mutuelle

Définition

☛ **Exclusion mutuelle** : si une ressource a été accédée par un processus P_1, aucun autre processus ne peut y accéder tant qu'elle n'a pas été libérée par P_1.

- Si P_1 est interrompu, il faut attendre à ce qu'il reprenne son exécution pour qu'il puisse libérer la ressource
- Une section de programme est dite atomique lorsqu'elle ne peut pas être interrompue par un autre processus manipulant les mêmes ressources critiques.
- C'est donc une atomicité relative à la ressource
- Un mécanisme d'exclusion mutuelle sert à assurer l'atomicité des sections critiques relatives à une ressource critique

Section critique et Exclusion mutuelle

Résolution des problèmes d'accès concurrents

1. Les **sémaphores**
2. Les moniteurs
3. Communication entre processus : primitives *send* et *receive*
4. Solutions matérielles

Sémaphores [E. W. Dijkstra, 1965] : présentation

Définition 1

- Un **sémaphore (S)** est un compteur entier qui désigne le nombre d'autorisations d'accès à une section critique
- Un sémaphore est une variable entière qui compte le nombre de processus en attente d'une section critique.
- Un sémaphore est une variable protégée (implémentation cachée), dont une "donnée-membre" est un compteur "NATURAL" (entier positif ou nul) initialisé à sa création à une valeur s_0 de type "NATURAL"
- Les sémaphores sont un outil élémentaire de synchronisation qui évitent l'attente active
- Le sémaphore est un objet de type abstrait. Il est caractérisé par la donnée d'un compteur et d'une file d'attente

Manipulation des Sémaphores

Deux types de sémaphores

1. **Sémaphore binaire** : peut prendre comme valeur 0 ou 1. Il est utilisé pour réaliser l'exclusion mutuelle
2. **Sémaphore n-aire** : peut prendre *n* valeurs. Il sert à spécifier le nombre d'accès maximal à une ressource (en lecture, évidement !)

Manipulation des Sémaphores

- Un sémaphore S =
  - un entier E(S) ;
  - une file d'attente F(S) ;
  - deux primitives P(S) et V(S).

- ☛ Manipulés au moyen des opérations **P** ou **wait** et **V** ou **signal**
- ☛ L'opération $P(S)$ décrémente la valeur du sémaphore S si cette dernière est supérieure à 0
- ☛ Sinon le processus appelant est mis en attente
- ☛ Le test du sémaphore, le changement de sa valeur et la mise en attente éventuelle sont effectués en une seule opération atomique indivisible
- ☛ L'opération $V(S)$ incrémente la valeur du sémaphore S, si aucun processus n'est bloqué par l'opération P(S)
- ☛ Sinon, l'un d'entre eux sera choisi et redeviendra prêt. V est aussi une opération indivisible

Manipulation des Sémaphores

☛ **Soit p le processus qui effectue P(S) ou V(S).**

P(S)

- E(S) = E(S) - 1;
  - Si E(S) < 0 alors
    - état(p) = bloqué;
    - entrer(p, F(S))

➠ Si le sémaphore est binaire, alors une seule exécution de la section critique

V(S)

- E(S) = E(s) + 1;
- si E(s) <= 0 alors
  - sortir(q, F(S));
  - état(q) = éligible;
  - entrer(q, F(éligibles));

Sémaphores d'exclusion Mutuelle

- But : protéger l'accès à une ressource unique (e.g. variable, imprimante, etc)
  - E(S) est initialisée à 1.
  - Utilisation :
    - P(S)
      < Section critique >
    - V(S)
  - Tous les processus doivent suivre la même règle.

Sémaphores de Synchronisation

☛ But : un processus doit en attendre un autre pour continuer (ou commencer) son exécution.

- ➠ E(S) est initialisée à 0.
- ➠ Utilisation :

Processus 1

- 1er travail
- V(s) // réveil processus 2

Processus 2

- P(s) // attente processus 1
- 2ème travail

Sommaire

Définitions

- ☛ Un ensemble de processus est en inter-blocage si chaque processus attend un événement que seul un autre processus de l'ensemble peut engendrer
- ☛ 2 (ou +) processus sont en attente de la libération d'une ressource attribuée à l'autre
- ☛ Un ensemble de processus est bloqué !

Vers un situation d'inter-blocage

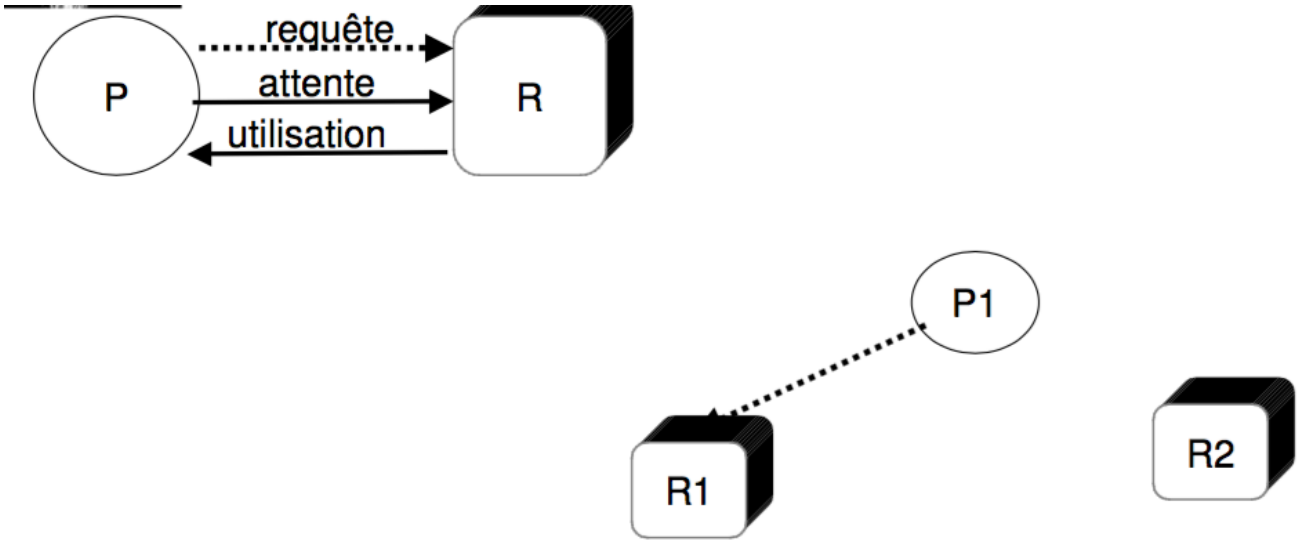


Figure: Vers un situation d'inter-blocage : 1

Vers un situation d'inter-blocage

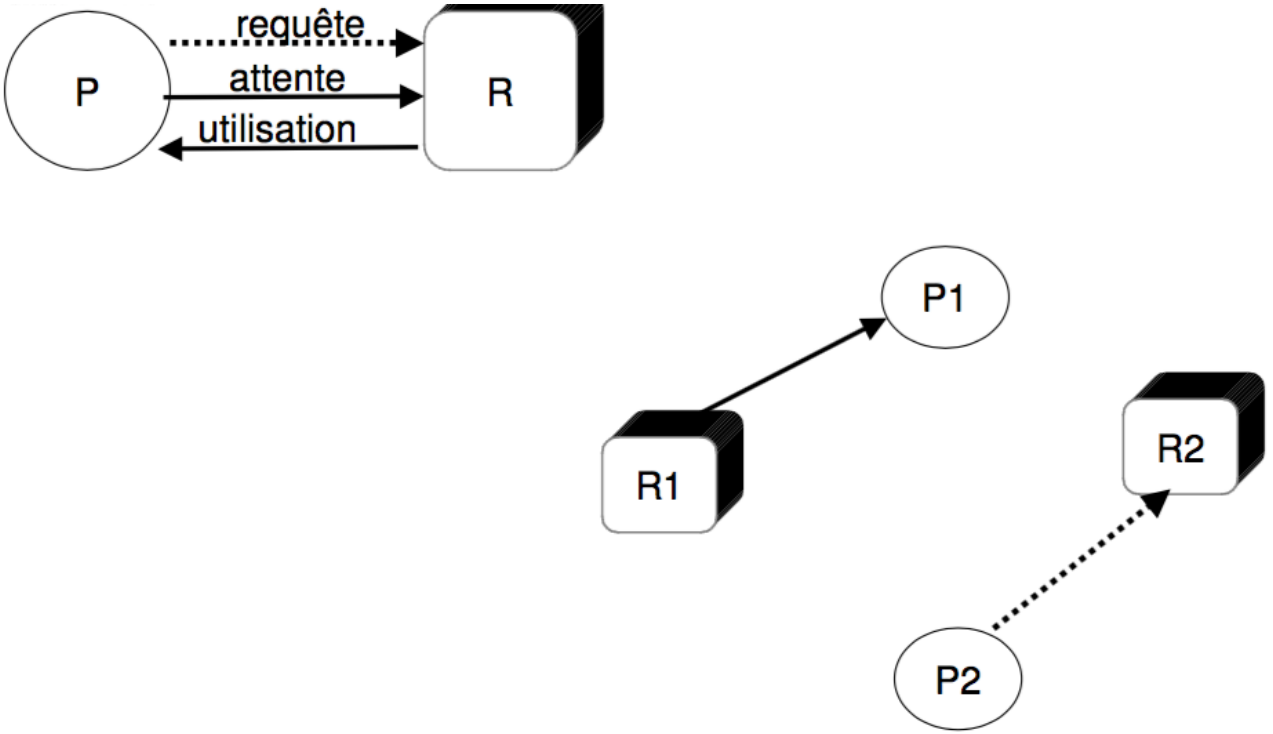


Figure: Vers un situation d'inter-blocage : 2

Vers un situation d'inter-blocage

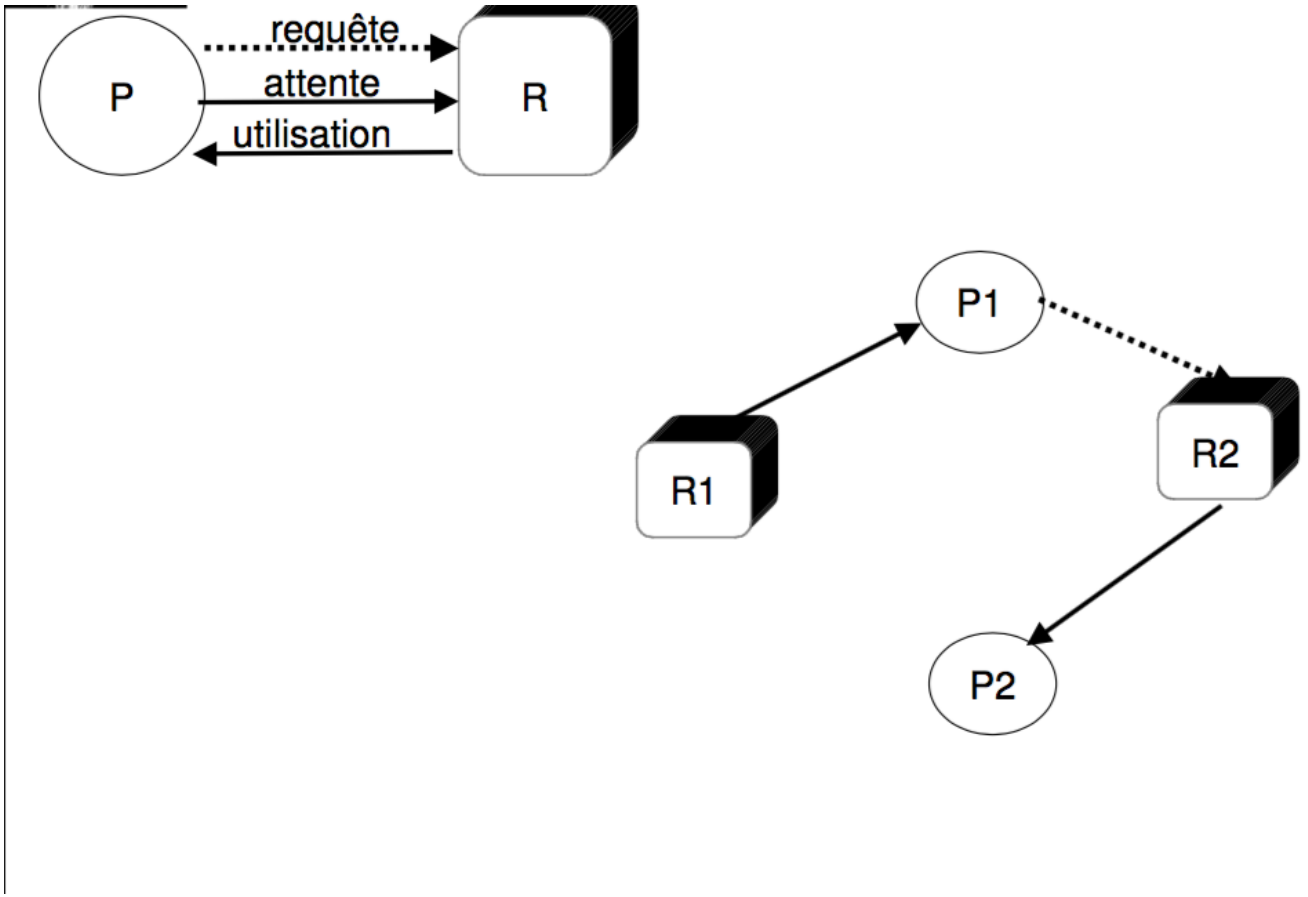


Figure: Vers un situation d'inter-blocage : 3

Vers un situation d'inter-blocage

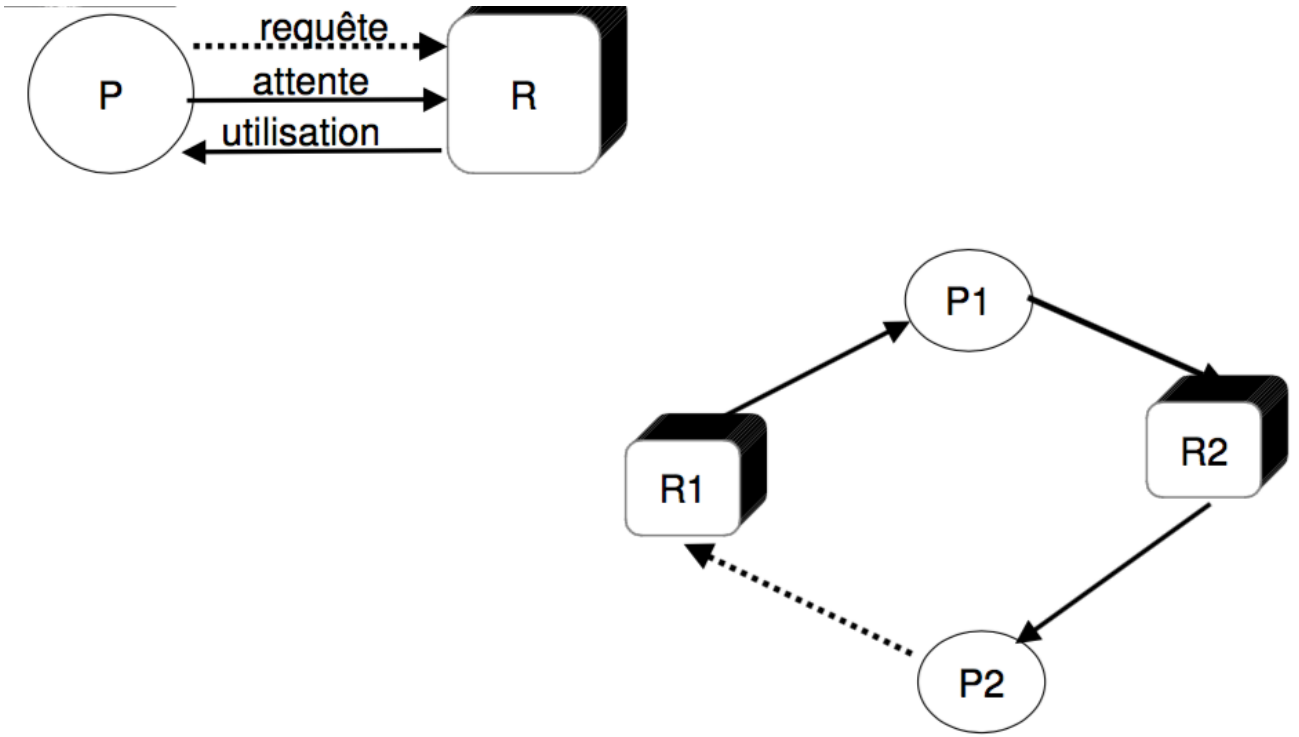


Figure: Vers un situation d'inter-blocage : 4

Vers un situation d'inter-blocage

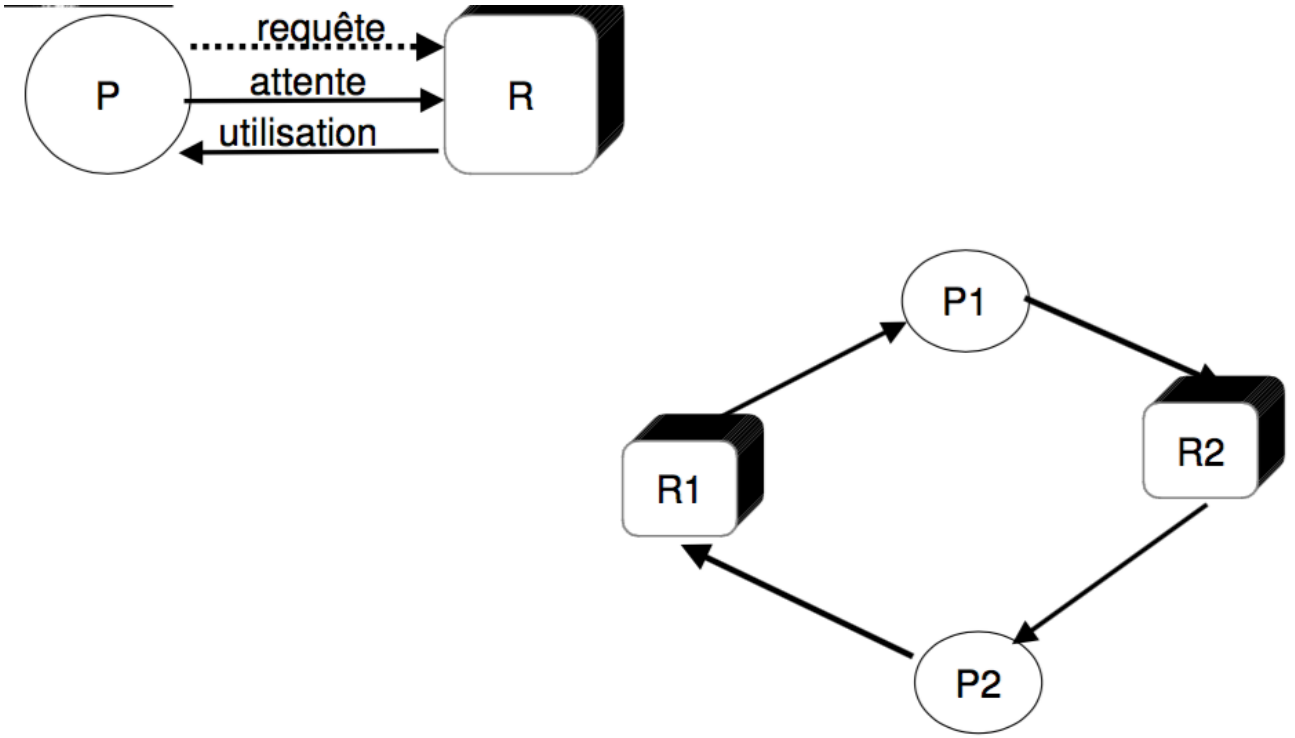


Figure: Vers un situation d'inter-blocage : 5

4 Conditions conduisant à un inter-blocage

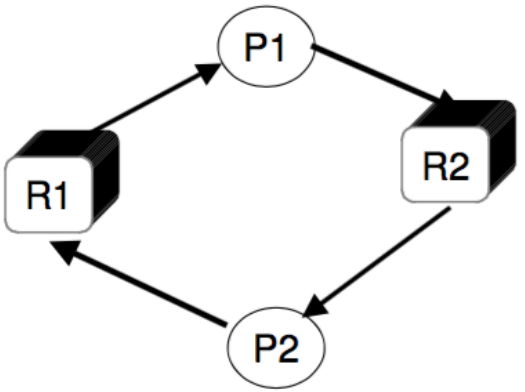
1. Ressource limitée (exclusion mutuelle)
2. Attente circulaire : il doit y avoir au moins deux processus chacun attendant une ressource détenu par un autre
3. Occupation et attente : les processus qui détiennent des ressources peuvent en demander de nouvelles
4. Pas de réquisition(non préemption) : les ressources sont libérées par le processus

Stratégie de gestion

- ☛ Détecter et résoudre
- ☛ Prévenir la formation d'inter-blocages en empêchant l'apparition d'une des 4 conditions
- ☛ Éviter les inter-blocages en allouant de manière « intelligente » les ressources

Détecter et résoudre

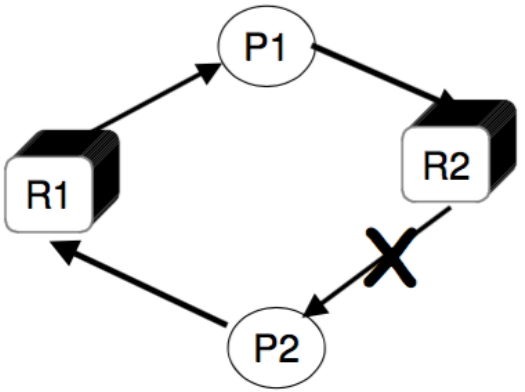
- graphe de dépendance + détection des cycles



- Comment continuer? :

Détecter et résoudre

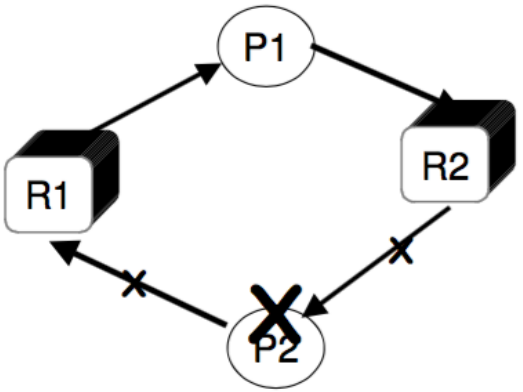
- graphe de dépendance + détection des cycles



- Comment continuer? :
  1. Retirer la ressource (La préemption) :
     - Certaines ressources, de par leur nature peuvent être retirées temporairement (parfois avec une intervention humaine).

Détecter et résoudre

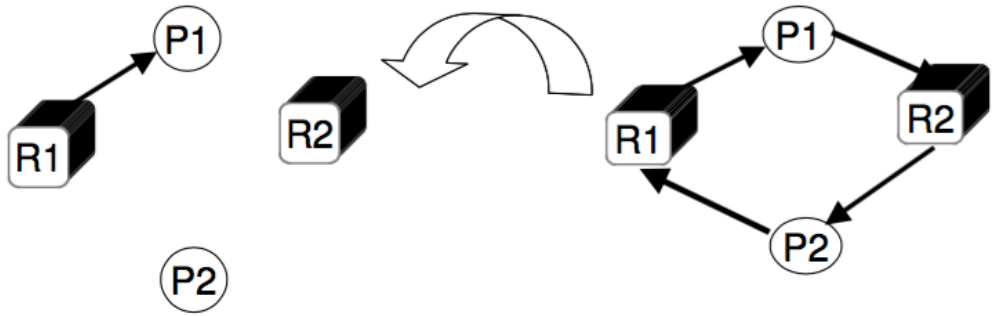
- graphe de dépendance + détection des cycles



- Comment continuer? :
  1. Retirer la ressource (La préemption)
  2. Suppression d'un processus :
     - on peut tuer un processus pris en inter-blocage afin de libérer ses ressources.
     - Un autre processus qui n'est pas dans le coup peut également être tué, si ça marche... ça marche

Détecter et résoudre

- graphe de dépendance + détection des cycles



- Comment continuer? :
  1. Retirer la ressource (La préemption)
  2. Suppression d'un processus
  3. Gestion de processus avec des points de reprise (rollback) :
     - à certains points clé, les états des processus sont enregistrés afin de pouvoir être restaurés ultérieurement.
     - Lors d'un inter-blocage, un processus est ramené à un point de reprise antérieur à l'acquisition de la ressource.

Prévenir l'inter-blocage

- exclusion mutuelle :
  - empêcher un processus de s'accaparer une ressource (spooling)
- attente circulaire :
  - les processus doivent demander les ressources dans un certain ordre, fixé par le SE.
- occupation et attente :
  - forcer les processus à demander toutes leurs ressources au début
  - exécuter un processus ssi toutes les ressources sont disponibles
  - réquisition

Allouer « correctement »

- ☛ Méthode de l'allocation globale des ressources à une tache
  - immobilisation non productive de ressources
- ☛ « Algorithme du banquier » :
  - L'algorithme du banquier utilise la métaphore d'un banquier d'une petite ville qui dispose d'un certain montant d'argent à prêter à ses clients.
  - Les clients sont des processus, l'argent correspond aux ressources et le banquier à l'OS.
  - L'algorithme vérifie si une requête mène à un état non sûr et la refuse en telle cas.
  - Sinon, tant que les requêtes laissent un état sûr derrière, elles sont allouées
    - Annonce des besoins
    - Déterminer à quel moment la ressource peut être alloué en sécurité

La prévention des inter-blocages

- ☛ On peut s'attaquer aux quatre conditions de l'inter-blocage en essayant de les éliminer, empêchant ainsi tout inter-blocage.
  1. La condition d'exclusion mutuelle : le fait de ne pas réserver exclusivement les ressources permet d'éliminer cette condition.
     - démon d'impression est un exemple de solution de ce type
  2. La condition de détention et d'attente : on peut exiger que tous les processus commandent toutes leurs ressources avant de commencer à les utiliser.
     - solution, en plus de mal utiliser les ressources exigent de connaître par avance les ressources nécessaires.
     - En tel cas, un algorithme du banquier ferait l'affaire.
  1. La condition de non-préemption : cette solution est au mieux très délicate et au pire impossible. Penser à une table traçante que l'on retirerait.

La prévention des inter-blocages

☛ On peut s'attaquer aux quatre conditions de l'inter-blocage en essayant de les éliminer, empêchant ainsi tout inter-blocage.

4 la condition d'attente circulaire :

- peut interdire l'accès à plusieurs ressources, ce qui est un peut contraignant
- Une autre solution consiste à ordonner les ressources et exiger que celles-ci soient réservées dans un ordre précis.
- Un processus demandant une ressource d'ordre inférieure à l'une qu'elle détient se la verra alors refusée.
- Il peut arriver qu'il ne soit pas possible en pratique d'ordonner les ressources
- De plus, cela est contraignant pour les processus.

La vérité

- ☛ Le programmeur doit utiliser son cerveau et être responsable
- ☛ Organiser le système pour minimiser les risques :
  - Tous les appels systèmes susceptibles de bloquer sont implémentés avec des délais d'attente

Sommaire

Introduction

- Les processus coopèrent souvent pour traiter un même problème
- Ces processus s'exécutent en parallèle sur un même ordinateur (mono-processeur ou multiprocesseurs) ou bien sur des ordinateurs différents.
- Ils doivent alors s'échanger des informations : **communication inter-processus**
- Il existe plusieurs moyens de communication inter-processus :
  1. les signaux
  2. les messages :
     - les tubes de communication
     - les sockets

les signaux

- Les **interruptions logicielles** ou **signaux** sont utilisées par le système d'exploitation pour aviser les processus utilisateurs de l'occurrence d'un événement important.
- De nombreuses erreurs détectées par le matériel, comme l'exécution d'une instruction non autorisée (une division par 0, par exemple) ou l'emploi d'une adresse non valide, sont converties en signaux qui sont envoyés au processus fautif.
- Ce mécanisme permet à un processus de réagir à cet événement sans être obligé d'en tester en permanence l'arrivée.
- Les processus peuvent indiquer au système ce qui doit se passer à la réception d'un signal

les signaux

- ☛ On peut ainsi ignorer le signal, ou bien le prendre en compte, ou encore laisser le SE appliquer le comportement par défaut :
  - en général tuer le processus
- ☛ Certains signaux ne peuvent être ni ignorés, ni capturés
- ☛ Si un processus choisit de prendre en compte les signaux qu'il recroit, il doit alors spécifier la procédure de gestion de signal.
- ☛ Quand un signal arrive, la procédure associée est exécutée.
- ☛ A la fin de l'exécution de la procédure, le processus s'exécute à partir de instruction qui suit celle durant laquelle l'interruption a eu lieu

Synthèse

- ☛ Les signaux sont utilisés pour établir une communication minimale entre processus, une communication avec le monde extérieure et faire la gestion des erreurs.

les signaux

Décision prise à l'arrivée d'un signal

- Tous les signaux ont une routine de service, ou une action par défaut. Cette action peut être du type :
  - terminaison du processus
  - ignorer le signal
  - Créer un fichier *core*
  - Stopper le processus
  - La procédure de service ne peut pas être modifiée
  - Le signal ne peut pas être ignoré

Les messages

- Un autre mécanisme de communication entre processus est l'échange de **messages**
- Chaque message véhicule des données
- Un processus peut envoyer un message à un autre processus se trouvant sur la même machine ou sur des machines différentes
- Unix offre plusieurs mécanismes de communication pour l'envoie de messages : les tubes de communication sans nom, nommés et les sockets

Présentation générale

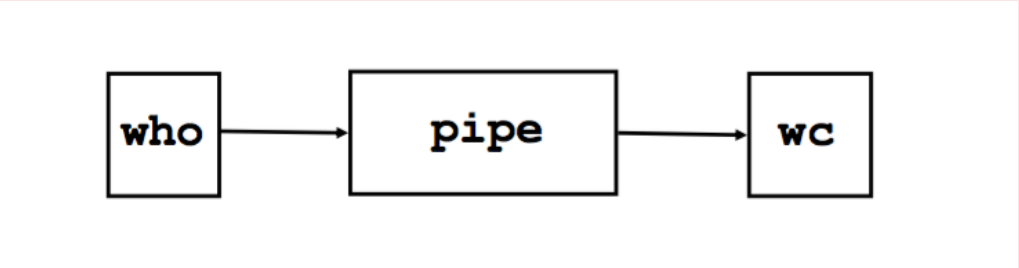
- Les tubes de communication ou pipes permettent à deux ou plusieurs processus d'échanger des informations.
- On distingue deux types de tubes :
  1. **tubes sans nom** (unamed pipe)
  2. **tubes nommés** (named pipe)

Les tubes sans nom

- Les tubes sans nom sont des liaisons de communication unidirectionnelles.
- La taille maximale d'un tube sans nom varie d'une version à une autre d'Unix, mais elle est approximativement égale à 4 Ko
- Les tubes sans nom peuvent être créés par le shell ou par l'appel système pipe()
- Les tubes sans nom du shell sont crées par l'opérateur binaire « | » :
  - la sortie standard d'un processus vers l'entrée standard d'un autre processus
  - Les tubes de communication du shell sont supportés par toutes les versions d'Unix

Les tubes sans nom : exemple

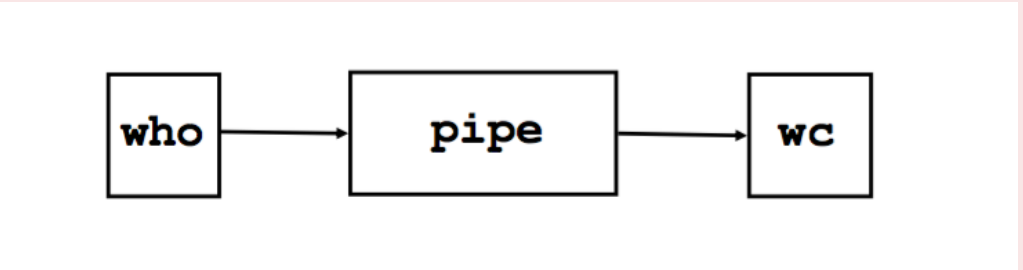
- Soit la commande suivante :
  - **who | wc -l**
- Elle crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe



1. Elle détermine le nombre d'utilisateurs connectés au système en appelant who, puis en comptant les lignes avec wc
2. Le premier processus réalise la commande who. Le second processus exécute la commande wc -l

Les tubes sans nom : exemple

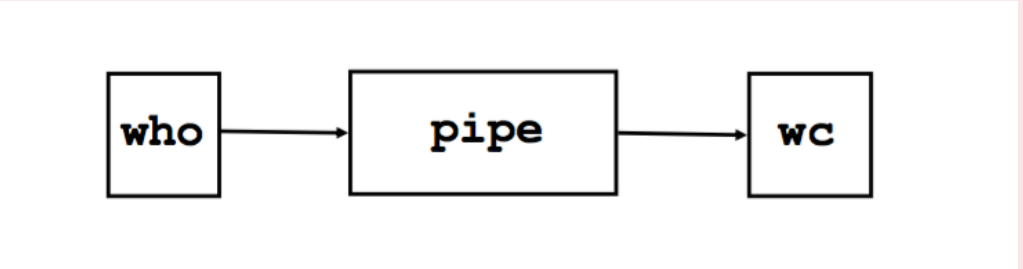
- Soit la commande suivante :
  - **who | wc -l**
- Elle crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe



4. Le processus réalisant la commande **who** ajoute dans le tube une ligne d"information par utilisateur du système
5. Le processus réalisant la commande **wc -l** récupère ces lignes d'information pour en calculer le nombre total
6. Le résultat est affiché à l?écran

Les tubes sans nom : exemple

- Soit la commande suivante :
  - **who | wc -l**
- Elle crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe

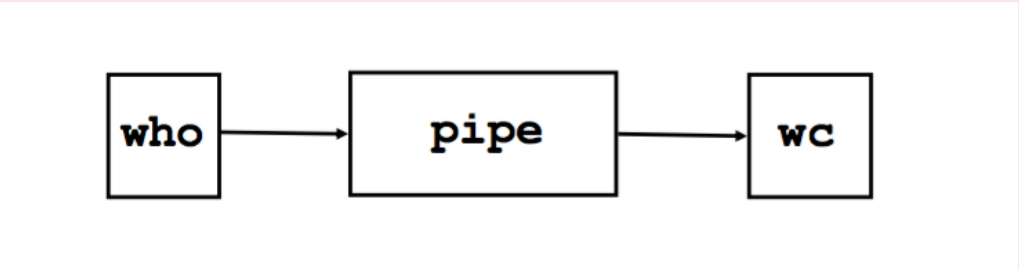


7 Les deux processus s'exécutent en parallèle, les sorties du premier processus sont stockées sur le tube de communication.

8 Lorsque le tube devient plein, le premier processus est sus- pendu jusqu'à ce qu'il y ait libération de l'espace nécessaire pour stocker une ligne d'information

Les tubes sans nom : exemple

- Soit la commande suivante :
  - **who | wc -l**
- Elle crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe



8. Lorsque le tube devient plein, le premier processus est sus- pendu jusqu'à ce qu'il y ait libération de l'espace nécessaire pour stocker une ligne d'information
9. De même, lorsque le tube devient vide, le second processus est suspendu jusqu'à ce qu'il y ait au moins une ligne d'information sur le tube.

Tubes de communication nommés

- Linux supporte un autre type de tubes de communication, beaucoup plus performants
- Il s'agit des tubes nommés (named pipes).
- Les tubes de communication nommés fonctionnent aussi comme des files du type FIFO (First In First Out)
- Ils sont plus intéressants que les tubes sans nom car ils offrent, en plus, les avantages suivants :
  1. Ils ont chacun un nom qui existe dans le système de fichiers (une entrée dans la Table des fichiers)
  2. Ils sont considérés comme des fichiers spéciaux.
  3. Ils peuvent être utilisés par des processus indépendants, à condition qu'ils s'exécutent sur une même machine
  4. Ils existeront jusqu'à ce qu'ils soient supprimés explicitement
  5. Ils sont de capacité plus grande, d'environ 40 Ko.

Sockets

- Les tubes de communication permettent d'établir une communication unidirectionnelle entre deux processus d'une même machine
- Le système Unix/Linux permet la communication entre processus s'exécutant sur des machines différentes au moyen de **sockets**.
- Les sockets ou prises sont les mécanismes traditionnels de communication inter-processus du système Unix
- Elles permettent la communication entre processus s'exécutant sur des machines différentes connectées par un réseau de communication.
- Par exemple, l'utilitaire **rlogin** qui permet à un utilisateur d'une machine de se connecter à une autre machine, est réalisé au moyen de sockets
- Les sockets sont également utilisé pour imprimer un fichier se trouvant sur une autre machine et pour transférer un fichier d'une machine à autre.

Sommaire

Les Appels systèmes

- Un appel système est un moyen de communiquer directement avec le noyau de la machine
- Le noyau regroupe toutes les opérations vitales de la machine
- Ainsi il est impossible d'écrire directement sur le disque dur
- L'utilisateur doit passer par des appels systèmes qui contrôlent les actions qu'il fait
- Ceci permet de garantir :
  1. la sécurité des données car le noyau interdira à un utilisateur d'ouvrir les fichiers auxquels il n'a pas accès
  2. l'intégrité des données sur le disque. Un utilisateur ne peut pas par mégarde effacer un secteur du disque ou modifier son contenu
- Ainsi, les appels système sont les fonctions permettant la communication avec le noyau
  - open, read, write, etc.

Les Appels système : utilisation

- Travaillent en relation directe avec le noyau
- Retournent un entier positif ou nul en cas de succès et -1 en cas d'échec
- Par défaut le noyau peut bloquer les appels systèmes et ainsi bloquer l'application si la fonctionnalité demandée ne peut pas être servie immédiatement
- Ne resservent pas de la mémoire dans le noyau.
- Les résultats sont obligatoirement stockés dans l'espace du processus (dans l'espace utilisateur)
- il faut prévoir cet espace par allocation de variable (statique, pile) ou de mémoire (malloc(). . .)

Chapitre 01 : Gestion des Processus

Dr Mandicou BA
mandicou.ba@esp.sn
http://www.mandicouba.net

Diplôme Universitaire de Technique (DUT, 2^e année)
Diplôme Supérieure de Technologie (DST, 2^e année)
Option : **Informatique**

ECOLE SUPERIEURE POLYTECHNIQUE

www.esp.sn