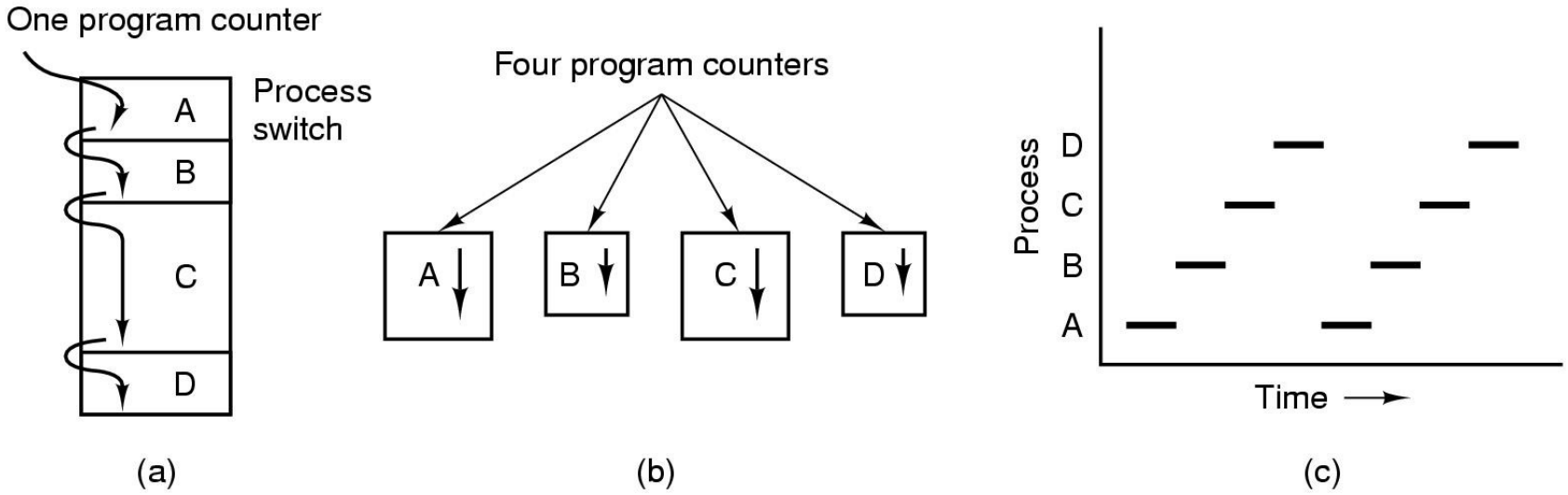
Gestion des activités parallèles

- Introduction, motivation
- Processus : définitions
- Synchronisation entre activités
- Les activités parallèles en Unix

Introduction Motivation

- Existence d'activités parallèles
  - Réelles : entrées sorties, multi-processeurs
  - Virtuelles : usagers
- Relations entre ces activités
  - Coopération
  - Compétition
- Difficultés de programmation et de mise au point

Comment faire ?



- Multiprogrammation de 4 programmes
- Quatre exécutions indépendantes
- Un seul programme actif à chaque instant

Définitions

- Action atomique = exécution d'une instruction
- Points observables
- Processus (séquentiel) = suite d'actions. Un processus correspond à l'exécution d'un programme
- Contexte ensemble des informations que les actions d'un processus peuvent consulter ou modifier

Concrètement, un processus =

- Un espace mémoire : code, données, pile
- Un compteur ordinal
- Des (images des) registres généraux

Un allocateur d'UC choisit le processus à exécuter

Création-Activation

- Créer un processus
  - Définir un espace mémoire
  - Définir un point de départ
- Lancer l'exécution d'un processus
  - Charger le compteur ordinal et les registres
- Arrêter un processus
  - Ranger les valeurs des registres
- Table des processus, pid

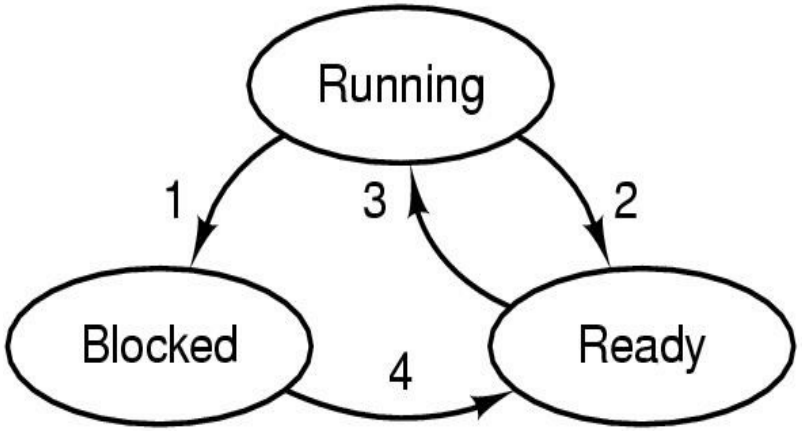
Destruction d’un processus

- Fin normale d’exécution
- Erreur fatale
- Destruction par un autre processus

Relations entre processus

- Exécution
  - Pseudo-parallèle
  - Parallèle
- Ressources et compétition entre processus
  - Ressources virtuelles
- Accès à des données partagées
  - Exclusion mutuelle
  - Synchronisation

Etats des processus



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Différents états
  - Élu (Running)
  - Bloqué (Blocked)
  - Pret (Ready)
- Diagramme de transitions

Ordonnanceur

- En charge de répartir le temps entre les processus
  - Stratégie d'ordonnancement
- Vise à optimiser un critère
  - Temps de réponse, débit des travaux,…
- Algorithmes non-préemptifs ou préemptifs
  - Le système peut interrompre un processus
  - Avec ou sans priorités

Gestion des activités parallèles

Deuxième partie : synchronisation entre processus communiquant par mémoire commune

Principaux éléments

- Synchronisation entre processus
- Un outil de synchronisation : le moniteur
- Comment résoudre un problème de synchronisation
- Schémas types de synchronisation

Synchronisation entre processus

- Exemples simples
  - Écriture/lecture
  - Rendez-vous

Écriture -> lecture

- Un processus écrit dans un tampon
- Un second processus doit prélever le résultat
- Condition : fin(écriture)<début(lecture)

Rendez-vous

- N processus, chacun avec un point de rendez-vous
- Lorsqu'un processus arrive à son point de rendez-vous, il se bloque sauf si tous les autres sont déjà arrivés
- Le dernier arrivé débloque tous les autres
- Condition d'attente : nb_arrivés < N

Synchronisation entre processus

- Points de synchronisation
- Contraintes : conditions de franchissement
- Difficultés de programmation et de mise au point
  - Indéterminisme
  - Exclusion mutuelle

Illustration problème d'exclusion mutuelle

Mov n, %eax
Inc %eax
Movl %eax,n

Processus A

Mov n, %eax
Inc %eax
Movl %eax,n

Processus B

Les moniteurs : un outil de synchronisation

- Définition : variables d'état + procédures utilisant ces variables
- Procédures en **exclusion mutuelle**
- Condition : variable spéciale avec deux opérations
  - Soit C une variable de condition
  - C.attendre : blocage du processus appelant « en attente de C »
  - C.signaler : réveil d 'un processus en attente de C, action vide si aucun processus en attente de C

Exemple 1 : écriture -> lecture

```
sync : moniteur;
var fait:booléen; fini : condition ;

procédure fin_écrire ;
        début
        fait:=vrai; fini.signaler;
        fin ;
procédure début_lire ;
        si non fait alors fini.attendre fsi;

début                          --initialisation
fait := faux ;
fin
fin sync
```

Exemple 2 : rendez-vous

```
rendezvous :moniteur
var n entier ; tousla :condition
procedure arriver;
    début
    n:=n+1;
    si n<N alors
                tousla.attendre
    fsi;
tousla.signaler
fin;
début n := 0 fin;    --initialisation
fin rendezvous
```

Sémantique de signaler

- Que se passe-t-il quand signaler réveille un processus ?
- Sans précaution, deux processus sont actifs dans le moniteur, celui qui signale et celui qui est réveillé
- Le processus qui effectue signaler se bloque et ne reprendra la main que lorsque le processus réveillé quitte le moniteur

Modèle du producteur et du consommateur

- Tampon de communication de N cases dans lequel le producteur (resp. le consommateur) dépose (resp. retire) un message à la fois
- Contraintes de synchronisation
  - Soit n le nombre de messages dans le tampon
  - aut(déposer) : $n < N$
  - aut (retirer) : $n > 0$

Producteur - Consommateur : solution

```
prod-cons : moniteur ;
var n : entier ; cprod,ccons :condition ;

    procedure deposer(m)
    si n=N alors
    cprod.attendre ;
    n++ ; dépôt ;
    ccons.signaler

    procedure retirer (m)
    si n=0 alors ccons.attendre ;
    n-- ; retrait ;
    cprod.signaler ;

n := 0 ;    --initialisation
fin prod-cons ;
```

Synchronisation temporelle

- Pourquoi ?
- Horloge
- Attendre (h)
  - Bloque le processus jusqu'à l'heure h
- Suspendre (t)
  - Bloque le processus pendant l'intervalle de temps t

Gestion des activités parallèles

Troisième partie : les processus en Unix

Objectifs

- Donner une vision simplifiée de la gestion des activités parallèles dans Unix
- Préparation du TP 2
- On ne considère que les processus utilisateurs

Langage de commande et processus

- Après la phase de login, un premier processus est créé et s'exécute : un interprète du langage de commande (un « shell »)
- Chaque commande entraîne la création d'un nouveau processus
- Options toto et toto&
  - Dans le premier cas, le shell est bloqué jusqu'à la fin de toto, dans le second, il reprend le contrôle immédiatement
- Communication par « pipe » toto | lulu

Processus

- Un espace de mémoire isolé (un espace virtuel)
  - La correspondance espace virtuel - mémoire physique sera vue dans le chapitre sur la mémoire
- Un compteur ordinal et des registres
- Un processus est identifié par un numéro (pid), entrée dans une table globale, la table des processus
- La commande ps permet d'obtenir la liste de tous les processus

Création d'un processus

- Appel système fork
  - Crée un nouveau processus par clonage du processus père : l'espace du processus créé est une copie de l'espace du père (mémoire et registres)
  - fork retourne au père le pid du processus créé et au fils la valeur 0

Définition d'un nouveau programme à exécuter

- Appel système « exec »
  - Modification de l'espace mémoire : chargement du nouveau programme dans l'espace du processus et branchement à ce nouveau programme
- Après fork, puis exec, les processus père et fils s'exécutent en parallèle
- Le père peut attendre la fin de l'exécution du fils par l'appel système « wait »

Les fils d'exécution : « threads » ou processus légers

- La création d'un processus est une opération lourde
- Les processus sont isolés dans des espaces mémoire différents
- D'où l'idée de faire coexister plusieurs activités parallèles à l'intérieur d'un même espace mémoire
- Ces activités sont appelées des « threads »

Les threads Unix

- Partagent l'espace mémoire de leur processus
- Ont en propre une pile et les registres
- Peuvent se communiquer des informations par l'intermédiaire de la mémoire commune