SÉANCE 14

INTRODUCTION À LA CONCURRENCE

SUJETS

Concepts de la concurrence

Concurrence au niveau sous-programme

Sémaphores

CONCEPT DE LA CONCURRENCE

uOttawa
L'Université canadienne
Canada's university

La concurrence peut être atteinte à différents niveaux:

- Niveau d'instruction
- Niveau d'unité
- Niveau de programme

L'exécution concurrentielle d'unités de programmes peut être effectuée:

- Physiquement sur des processeurs séparés,
- ou Logiquement dans un mode de temps tranché sur un système d'ordinateur à processeur

POURQUOI ÉTUDIER LA CONCURRENCE?

uOttawa
L'Université canadienne
Canada's university

Les systèmes d'ordinateur résous des problèmes du monde réel, où les événements se passent en même temps:

- Les réseaux de chemins de fer (beaucoup de trains qui circulent, mais qui doivent se synchroniser sur les sections partagées du chemin)
- Les systèmes d'exploitation, avec plusieurs processus qui se déroulent en même temps
- Les serveurs Web

Les ordinateurs à processeurs multiples ou multi-cœurs sont maintenant très populaire

- Ceci nécessite un logiciel qui utilise effectivement ce type de hardware

CONCURRENCE À NIVEAU SOUS-PROGRAMME

Une tâche est une unité de programme qui peut être exécutée en concurrence avec d'autres unités du même programme

- Chaque tâche dans un programme peut fournir un thread de contrôle

Une tâche peut communiquer avec une autre tâche à travers:

- Des variables non-locales partagées
- Le passage de message
- Des paramètres

SYNCHRONISATION

Un mécanisme pour contrôler l'ordre dans lequel les tâches exécutent

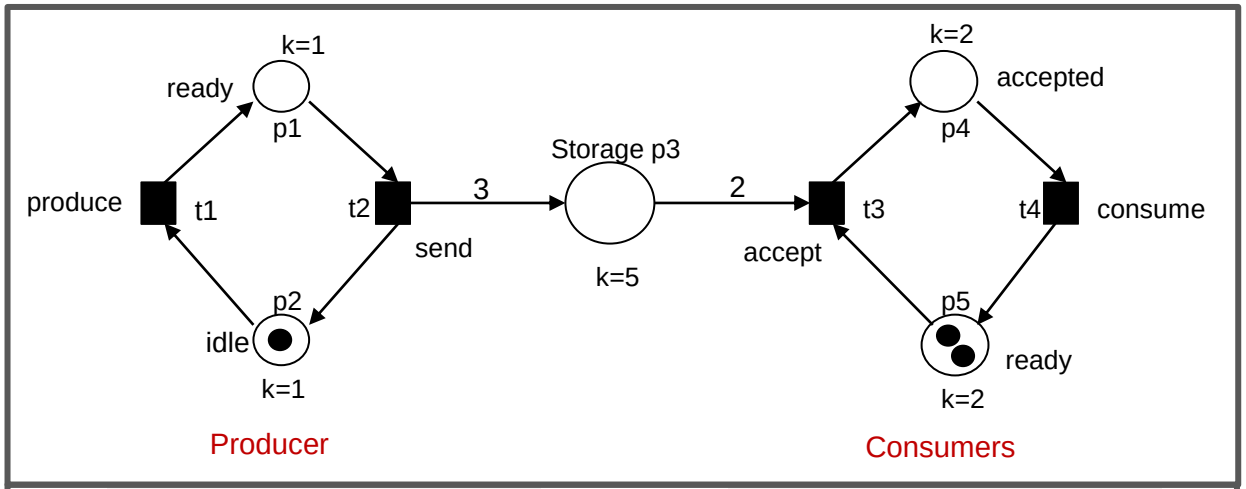
La synchronisation de coopération est requise entre la tâche A et la tâche B lorsque:

- **La tâche A** doit attendre **la tâche B** pour compléter une activité spécifique avant que la **tâche A** puisse continuer l'exécution
- **Rappel** du problème de réseaux de pétri producteur-consommateur

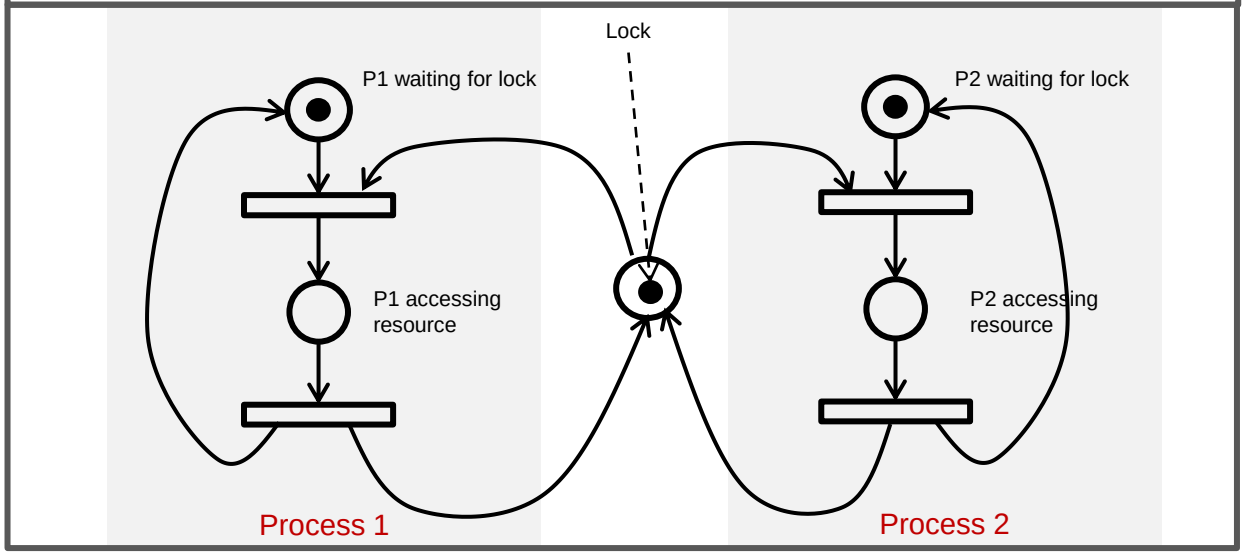
La synchronisation de compétition est requise entre deux tâches lorsque:

- Les deux tâches ont besoin d'utiliser une ressource qui ne peut pas être utilisée simultanément

SYNCHRONISATION (RÉSEAUX DE PETRI)

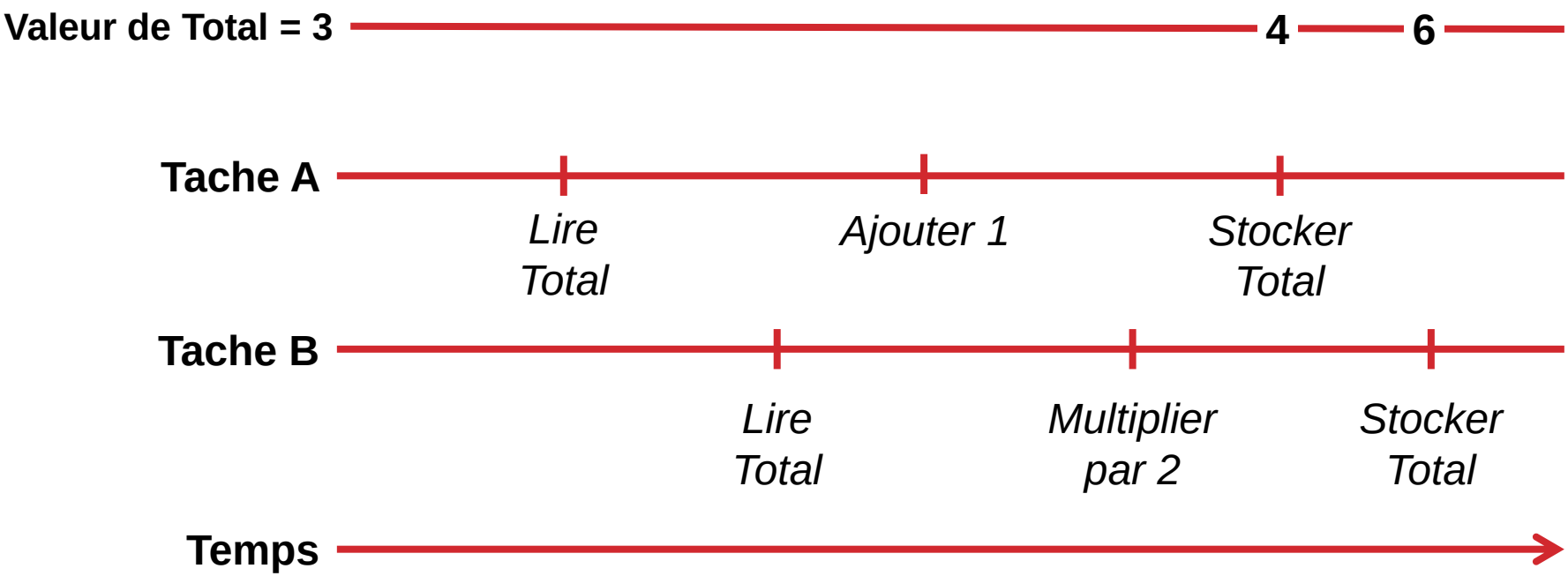


Synchronisation de coopération



Synchronisation de compétition

LA NÉCESSITÉ DE LA SYNCHRONISATION DE COMPÉTITION



SECTION CRITIQUE

Une section du code dans laquelle le thread peut faire:

- Le changement de variables communes,
- La mise à jour d'un tableau,
- L'écriture dans un document,
- Ou la mise à jour de ressources partagées

L'exécution de sections critiques par les threads est mutuellement exclusive dans le temps

ÉTATS DE TÂCHES (THREAD)

uOttawa
L'Université canadienne
Canada's university

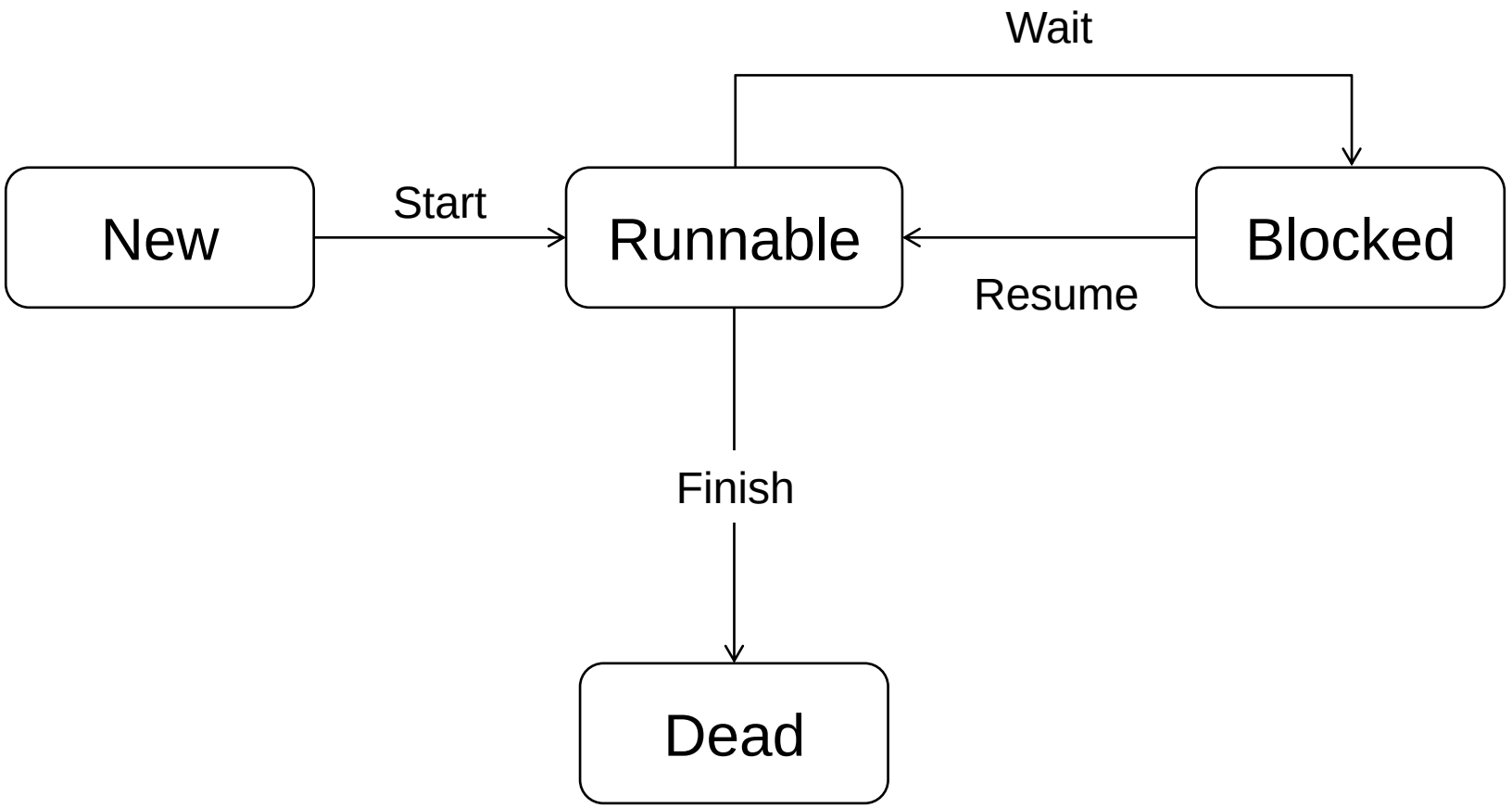
Nouveau (New): il a été créé, mais n'a pas encore commencer son exécution

Exécutable ou prêt (Runnable or Ready): il exécute couramment ou il est prêt à exécuter

Bloqué (Blocked): il exécutait, mais son exécution a été interrompue par un événement parmi plusieurs

Mort (Dead): il n'est plus actif dans aucun sens

ÉTATS DE TÂCHES (THREAD)



SÉMAPHORES

Le Sémaphore est une technique utilisée pour contrôler l'accès à une ressource commune pour plusieurs tâches

- C'est un objet qui consiste d'un nombre entier *(conteur)* et une file qui stocke les descripteurs de tâches

Une tâche qui a besoin de faire accès à une section critique a besoin « d'acquérir » le sémaphore

Classiquement: c'est un système pour communiquer des messages en tenant les bras de deux drapeaux dans certaines positions selon un code alphabétique

SÉMAPHORES

Deux opérations sont toujours associées avec les Sémaphores

L'Opération "P" est utilisée pour chercher la sémaphore, et "V" pour la relâcher

- P (proberen, qui veut dire tester et diminuer le nombre entier)
- V (verhogen, qui veut dire augmenter) l'operation

Alternativement, ces opérations sont appelées: wait et release

SÉMAPHORES

uOttawa
L'Université canadienne
Canada's university

Les systèmes d'exploitation distinguent souvent entre les sémaphores comptants et binaires

Le nombre entier d'un sémaphore comptant peut avoir n'importe quel valeur

Le nombre entier d'un sémaphore binaire peut seulement varier entre 0 et 1

SÉMAPHORES BINAIRE

La stratégie générale pour utiliser un sémaphore binaire afin de contrôler l'accès à une section critique est comme suit:

```
Semaphore aSemaphore;

wait(aSemaphore);
Critical section();
release(aSemaphore);
```

SYNCHRONISATION AVEC SÉMAPHORE BINAIRE

uOttawa
L'Université canadienne
Canada's university

wait(semaphoreBinaire):

```
if conteur de semaphoreBinaire == 1 then
  Change la valeur du conteur de semaphoreBinaire à 0
else
  Change l'état de la tache à bloqué (blocked)
  Met la tache dans la file du semaphoreBinaire
end
```

SYNCHRONISATION AVEC SÉMAPHORE BINAIRE

uOttawa
L'Université canadienne
Canada's university

release(semaphoreBinaire):

```
if file de semaphoreBinaire est vide then
  conteur = 1
else
  Change l'etat de la tache dans la tête du file à Exécutable (runnable)
  Fait un opération de défilage
end
```

SYNCHRONISATION AVEC SÉMAPHORE COMPTANT

uOttawa
L'Université canadienne
Canada's university

wait(semaphoreComptant):

```
if conteur de semaphoreComptant > 0 then
  Décrémente le conteur de semaphoreComptant
else
  Change l'état de la tache à bloqué (blocked)
  Met la tache dans la file du semaphoreComptant
end
```

SYNCHRONISATION AVEC SÉMAPHORE COMPTANT

uOttawa
L'Université canadienne
Canada's university

release(semaphoreComptant):

if file de semaphoreComptant est vide **then**
  incrémente le conteur de semaphoreComptant
else
  Change l'etat de la tache dans la tète du file à **Exécutable (runnable)**
  Fait un opération de défilage
end

PRODUCTEUR ET CONSOMMATEUR


```
semaphore fullspots, emptyspots;
fullspots.count := 0;
emptyspots.count := BUFLEN;
task producer;
  loop
  -- produce VALUE --
  wait(emptyspots);     { wait for a space }
  DEPOSIT(VALUE);
  release(fullspots);   { increase filled spaces }
  end loop;
end producer;

task consumer;
  loop
  wait(fullspots);      { make sure it is not empty }
  FETCH(VALUE);
  release(emptyspots); { increase empty spaces }
  -- consume VALUE --
  end loop
end consumer;
```

AJOUTER LA SYNCHRONISATION DE COMPÉTITION

uOttawa
L'Université canadienne
Canada's university

```
semaphore access, fullspots, emptyspots;
access.count := 1;
fullspots.count := 0;
emptyspots.count := BUFLEN;


task producer;
  loop
  -- produce VALUE --
  wait(emptyspots);       { wait for a space }
  wait(access);           { wait for access }
  DEPOSIT(VALUE);
  release(access);        { relinquish access }
  release(fullspots);     { increase filled spaces }
  end loop;
end producer;
```

AJOUTER LA SYNCHRONISATION DE COMPÉTITION

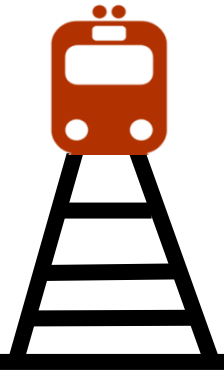
uOttawa
L'Université canadienne
Canada's university

```
task consumer;
   loop
   wait(fullspots);          { make sure it is not empty }
   wait(access);             { wait for access }
   FETCH(VALUE);
   release(access);          { relinquish access }
   release(emptyspots);      { increase empty spaces }
   -- consume VALUE --
   end loop
end consumer;
```

INTER-BLOCAGES

Une loi passée par la législature de Kansas vers le début du 20e siècle:

"…… Lorsque deux trains s'approchent à un croisement, les deux doivent s'arrêter complétement et aucun d'eux ne peut démarrer jusqu'à ce que l'autre est parti. ……"



CONDITIONS POUR UN INTER-BLOCAGE

uOttawa
L'Université canadienne
Canada's university

Exclusion mutuelle: **le fait de permettre à un seul processus d'avoir accès à une ressource partagée**

Tenir-et-attendre: **il doit y avoir un processus qui tiens au moins une ressource et attend d'acquérir des ressources additionnelles qui sont couramment tenus par d'autres processus**

Pas de préemption: **le manque de réallocation temporaire de ressource. La ressource peut être relâchée volontairement seulement**

Attente circulaire: **chaque processus impliqué dans l'impasse attend un autre processus de relâcher volontairement la ressource**

EXEMPLE D'INTER-BLOCAGE

uOttawa
L'Université canadienne
Canada's university

```
BinarySemaphore s1,s2;
```

```
Task A:
   wait(s1)
      // access resource 1
      wait (s2)
         // access resource 2
      release(s2)
   release(s1)
end task
```

```
Task B:
   wait(s2)
      // access resource 2
      wait(s1)
         // access resource 1
      release(s1)
   release(s2)
end task
```

STRATÉGIE DE GESTION DES INTER-BLOCAGES

uOttawa
L'Université canadienne
Canada's university

Prévention: **éliminer une des conditions nécessaires**

Évitement: **éviter si le système connaît à l'avance la séquence des demandes de ressources associées avec chaque processus actif**

Détection: **détecter en construisant des graphes de ressources dirigées et chercher les cycles**

Recouvrement: **lorsque détecté, il doit être démêlé et le système doit retourner à son état normal le plus rapidement possible**

- Terminaison du processus
- Préemption de ressources

MERCI!

QUESTIONS?