Programmation système

Processus, threads, goroutines

loig.jezequel@univ-nantes.fr

Concurrence

Étant donné un ensemble d'évènements

Il est dit **sequentiel**

si tous ses évènements ont toujours lieu dans le même ordre

Concurrence

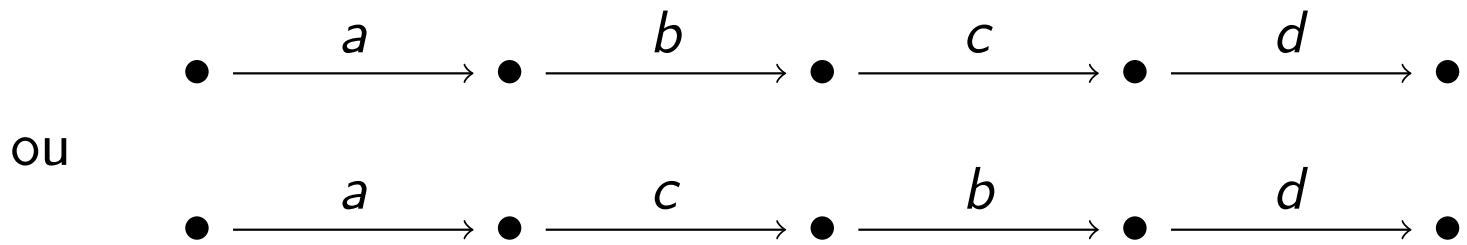
Étant donné un ensemble d'évènements

Il est dit **sequentiel**

si tous ses évènements ont toujours lieu dans le même ordre

Il est dit **concurrent**

si certains de ses évènements peuvent survenir dans différents ordres



Concurrence

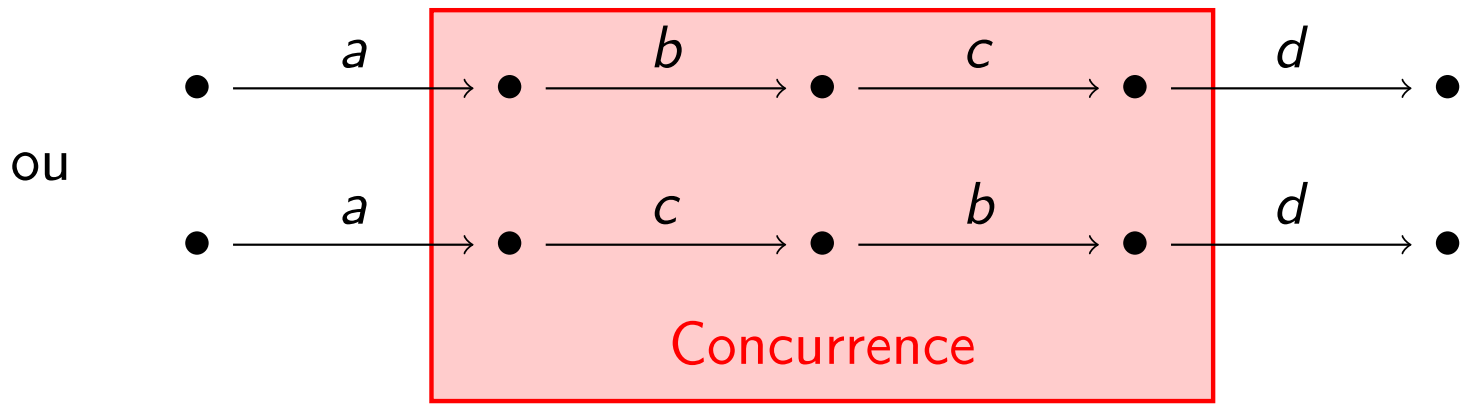
Étant donné un ensemble d'évènements

Il est dit **sequentiel**

si tous ses évènements ont toujours lieu dans le même ordre

Il est dit **concurrent**

si certains de ses évènements peuvent survenir dans différents ordres



Importance de la concurrence en programmation système

- Temps «long» pour accéder aux périphériques
  - quelques millisecondes pour accéder à des données sur un disque dur
  - quelques milliards d'opérations par seconde pour un processeur
- Temps «très long» et imprévisible pour les interactions de l'utilisateur
- Exécution séquentielle des tâches non souhaitable

Concurrence ou parallélisme ?

Concurrence

On ne sait pas dans quel ordre des évènements concurrents se déroulent

Parallélisme

Des évènements parallèles peuvent se dérouler en même temps

Parallélisme $\Longrightarrow$ concurrence

Des évènements parallèles sont aussi concurrents

Concurrence ou parallélisme ?

Concurrence

On ne sait pas dans quel ordre des évènements concurrents se déroulent

Parallélisme

Des évènements parallèles peuvent se dérouler en même temps

Parallélisme $\Longrightarrow$ concurrence

Des évènements parallèles sont aussi concurrents

Concurrence $\not\Longrightarrow$ parallélisme

On peut exécuter des évènements concurrents sur une architecture ne permettant pas le parallélisme, mais des évènements concurrents **se parallélisent naturellement**.

Processus

Processus

Definition (Processus)

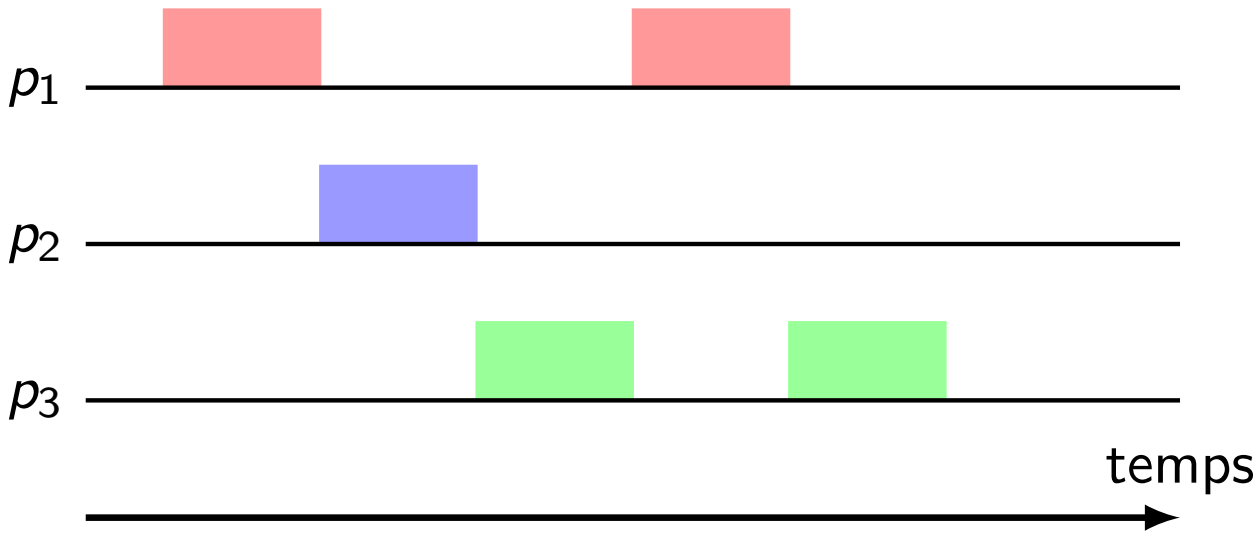
On appelle processus l'ensemble des activités résultant de l'exécution d'un programme au sein d'un système informatique
Un processus encapsule:

- un flot de contrôle logique: donne l'illusion d'un accès permanent et exclusif à l'UC
- un espace d'adressage virtuel: donne l'illusion d'un accès complet et exclusif à la mémoire

Un processus peut être vu comme une coquille d'exécution associée à un programme.

Processus concurrents

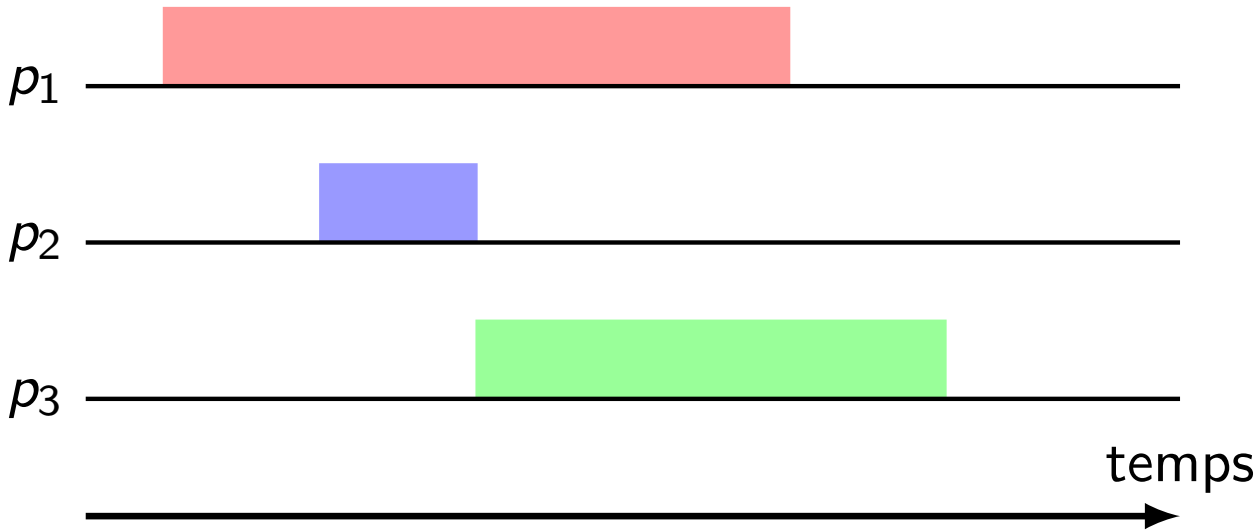
Lorsque plusieurs processus sont actifs simultanément, le noyau entrelace leurs exécutions. Le flot de contrôle logique masque cet entrelacement.



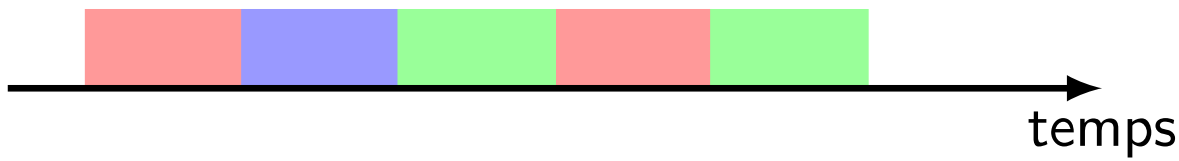
Deux processus actifs simultanément sont en concurrence pour accéder aux ressources de la machine.

Exécution de processus concurrents

Ce que voit l'utilisateur : tous les processus s'exécutent en parallèle



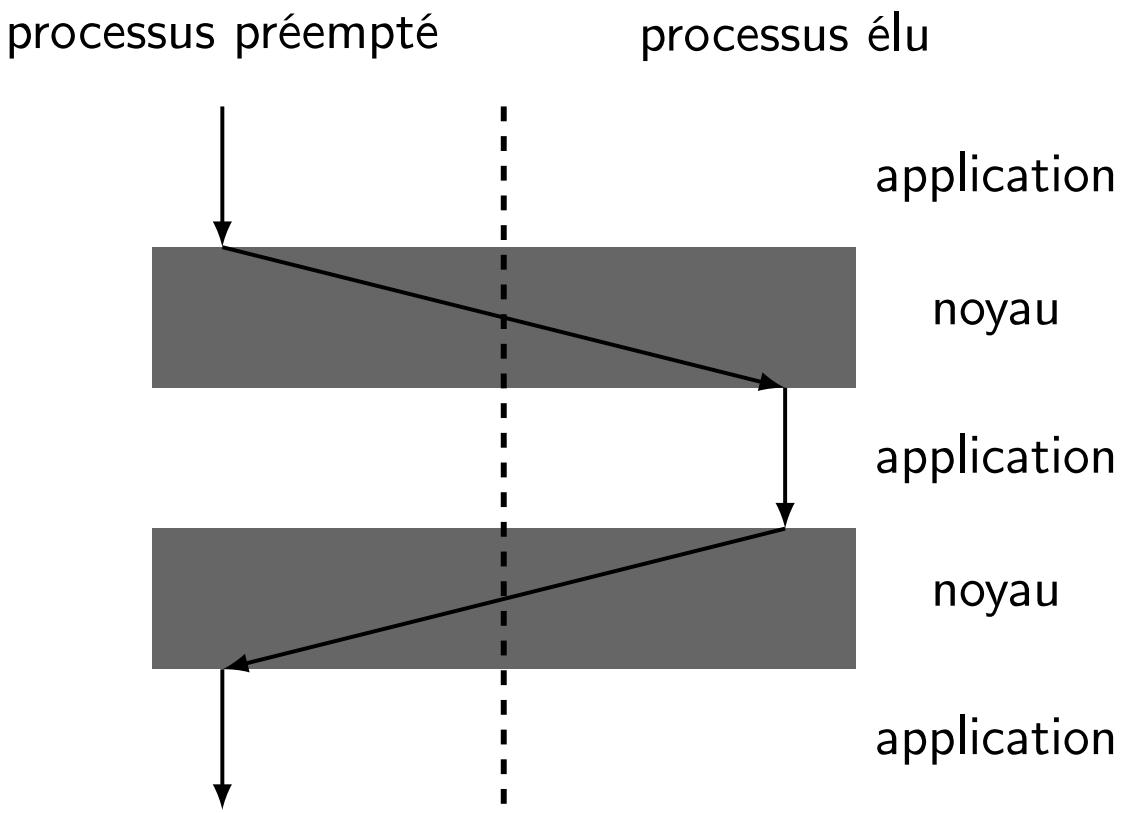
Ce qui se passe en réalité : le processeur est attribué par courtes tranches de temps aux processus concurrents



Changement de contexte (context switch)

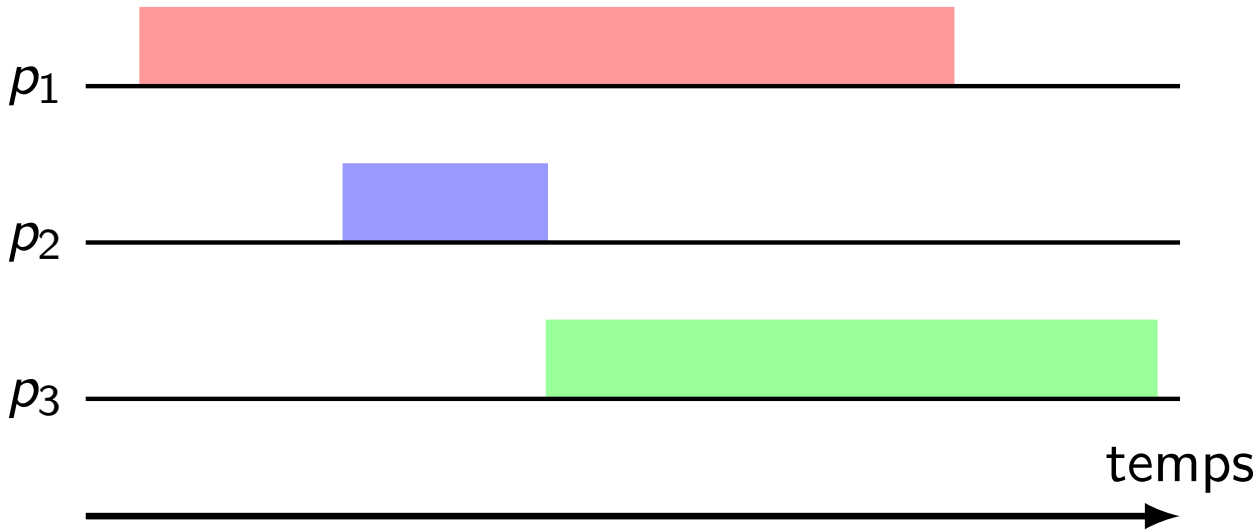
Pour donner l'impression de simultanéité, le SE change très régulièrement le processus en cours :

- sauvegarde l'état du processus préempté (processeur → mémoire)
- charge de l'état du processus élu (mémoire → processeur)

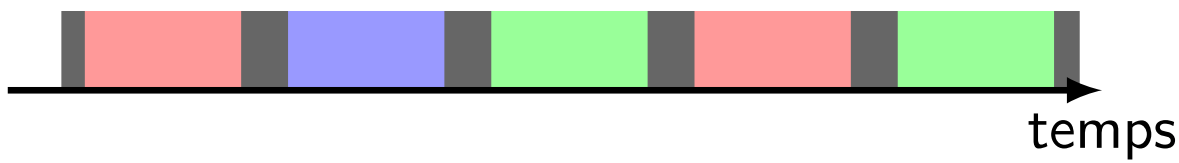


Exécution de processus concurrents

Ce que voit l'utilisateur : tous les processus s'exécutent en parallèle



Ce qui se passe en réalité : le noyau effectue régulièrement des changements de contexte.



Contexte d'un processus

Le contexte d'un processus est constitué de l'ensemble des informations relatives aux processus stockées dans des ressources communes de la machine qui doivent être sauvegardées ou restaurées lors d'un changement de contexte :

- le pointeur d'instruction
- les pointeurs de pile et de cadre
- les registres
- quelques autres informations utiles au système

Ces informations sont stockées dans une zone dédiée de la mémoire du noyau. Elles forment une partie du Process Control Block (PCB).

Organisation des processus

Caractéristiques d'un processus

- Un PID (identifiant de processus)
- Un parent (processus qui l'a créé)
- Un état (en cours d'exécution, stoppé, terminé)

Premier processus

Au démarrage, le noyau du système d'exploitation crée un processus d'initialisation.

- il porte le PID 1
- il ne s'arrête jamais
- il est la racine de l'arbre des processus
- il adopte les processus orphelins

Quelques commandes utiles sur les processus

ps

Lister les processus

top

Voir dynamiquement les processus qui s'exécutent, leur état, les ressources qu'ils consomment, etc

kill

Envoyer un signal à un processus (par exemple pour lui demander de s'arrêter)

Threads

Limitations des processus

Communication

En l'absence de mémoire partagée, la communication entre processus est lourde à mettre en place.

Contexte

Le contexte d'un processus contient beaucoup d'informations ce qui rend coûteux :

- la création de processus,
- la destruction de processus,
- le passage d'un processus à un autre (changement de contexte).

Gestion

La gestion assez rigide des processus implique fortement le système d'exploitation, notamment pour maintenir la structure arborescente qui doit exister entre eux.

Threads (processus légers)

- Fils d'exécution séquentiels
- Concurrents au sein d'un processus
- Ordonnancés par le noyau du système d'exploitation
- Mémoire partagée (entre threads d'un même processus)
- Pile et registres privés
- Changement de contexte peu coûteux
- Pas de notion d'enfants/parents

Goroutines

Goroutine

- Fil d'exécution séquentiel
- Plusieurs par thread (potentiellement)
- Très léger (10^6 par processus)
- Mémoire partagée (entre goroutines d'un programme)
- API minimale (seule chose possible = démarrer une goroutine)
- Ordonnancement par le scheduler du runtime Go

Goroutine

- Fil d'exécution séquentiel
- Plusieurs par thread (potentiellement)
- Très léger (10^6 par processus)
- Mémoire partagée (entre goroutines d'un programme)
- API minimale (seule chose possible = démarrer une goroutine)
- Ordonnancement par le scheduler du runtime Go

Par défaut

main, ramasse miette

Utiliser les goroutines

On utilise le mot clé go pour démarrer une goroutine

ping-pong.go

Main est plus importante que les autres

main.go

L'ordre d'exécution n'est pas controllable

schedule.go

La mémoire est partagée

global.go
local.go

On peut lancer beaucoup de goroutines en même temps

many.go

Ordonnancement, vu de loin

M:P:N Threading (en général, $N > M > P$)

- M threads,
- P processeurs,
- N goroutines.

Pour s'exécuter une goroutine est associée à un thread, qui est lui-même associé à un processeur.

Choix des goroutines à exécuter

- Une file d'attente par processeur
- Points d'intérêt permettant un changement (appels de fonctions notamment)
- Préemption pour assurer l'équité entre goroutines

Pourquoi $M > P$?

Supposons $M = P$

1. Lors d'un appel système un couple (thread, goroutine) peut être bloqué dans le noyau
2. Le runtime Go ne peut pas savoir si c'est bloqué ou simplement long (le noyau est invisible aux utilisateurs)
3. Si tous les threads sont bloqués ainsi le programme est, au moins temporairement, bloqué, le temps processeur est gaspillé
4. Pire, si les goroutines concernées sont en attente d'une goroutine qui ne s'exécute pas actuellement (car il n'y a pas de thread disponible pour elle) le programme est définitivement bloqué (deadlock)

Solution

Le runtime Go est autorisé à créer de nouveaux threads lors des appels système