Programmation Système Multi-tâche
Notions de base

Thierry Vaira

BTS SN Option IR

v.1.1 - 20 mars 2018

Notions de processus

- Un **processus** (*process*) est un **programme en cours d'exécution** par un système d'exploitation.
- Un **programme** est une **suite d'instructions permettant de réaliser un traitement**. Il revêt un caractère statique.
- Une **image** représente l'**ensemble des objets (code, données, ...) et des informations (états, propriétaire, ...)** qui peuvent donner lieu à une exécution dans l'ordinateur.
- Un **processus** est donc l'**exécution d'une image**. Le processus est l'aspect dynamique d'une image.
- Un **fil d'exécution** (*thread*) est l'**exécution séquentielle d'une suite d'instructions**.

Rôle du système d'exploitation

- Le **système d'exploitation** est chargé d'**allouer les ressources** (mémoires, temps processeur, entrées/sorties) nécessaires aux processus et d'**assurer son fonctionnement isolé** au sein du système. Il offre des **services** aux processus.
- Un des rôles du système d'exploitation est d'**amener en mémoire centrale l'image mémoire d'un processus avant de lui allouer le processeur**. Le système d'exploitation peut être amené à sortir de la mémoire les images d'autres processus et à les copier sur disque. Une telle gestion mémoire est mise en oeuvre par un algorithme de **va et vient appelée aussi *swapping***.
- Il peut aussi fournir une **API** pour **permettre leur gestion et la communication inter-processus (IPC)**.

Qu'est-ce que le multitâche ?

- Un système d'exploitation est **multitâche** (*multitasking*) s'**il permet d'exécuter, de façon apparemment simultanée, plusieurs programmes informatiques** (*processus*).
- La simultanéité apparente est le résultat de l'**alternance rapide d'exécution des processus présents en mémoire** (notion de **temps partagé et de multiplexage**).
- Le passage de l'exécution d'un processus à un autre est appelé **commutation de contexte**.
- Ces commutations peuvent être initiées par les programmes eux-mêmes (**multitâche coopératif**) ou par le système d'exploitation (**multitâche préemptif**). Tous les systèmes utilisent maintenant le multitâche préemptif.

Remarques

- Le **multitâche** n'est pas dépendant du **nombre de processeurs** présents physiquement dans l'ordinateur : un système multiprocesseur (ou multi-cœur) n'est pas nécessaire pour exécuter un système d'exploitation multitâche.
- Le **multitâche coopératif** n'est plus utilisé (cf. Windows 3.1 ou MAC OS 9).
- Unix et ses dérivés, Windows et MAC OS X sont des systèmes basés sur le **multitâche préemptif**.

À quoi ça sert ?

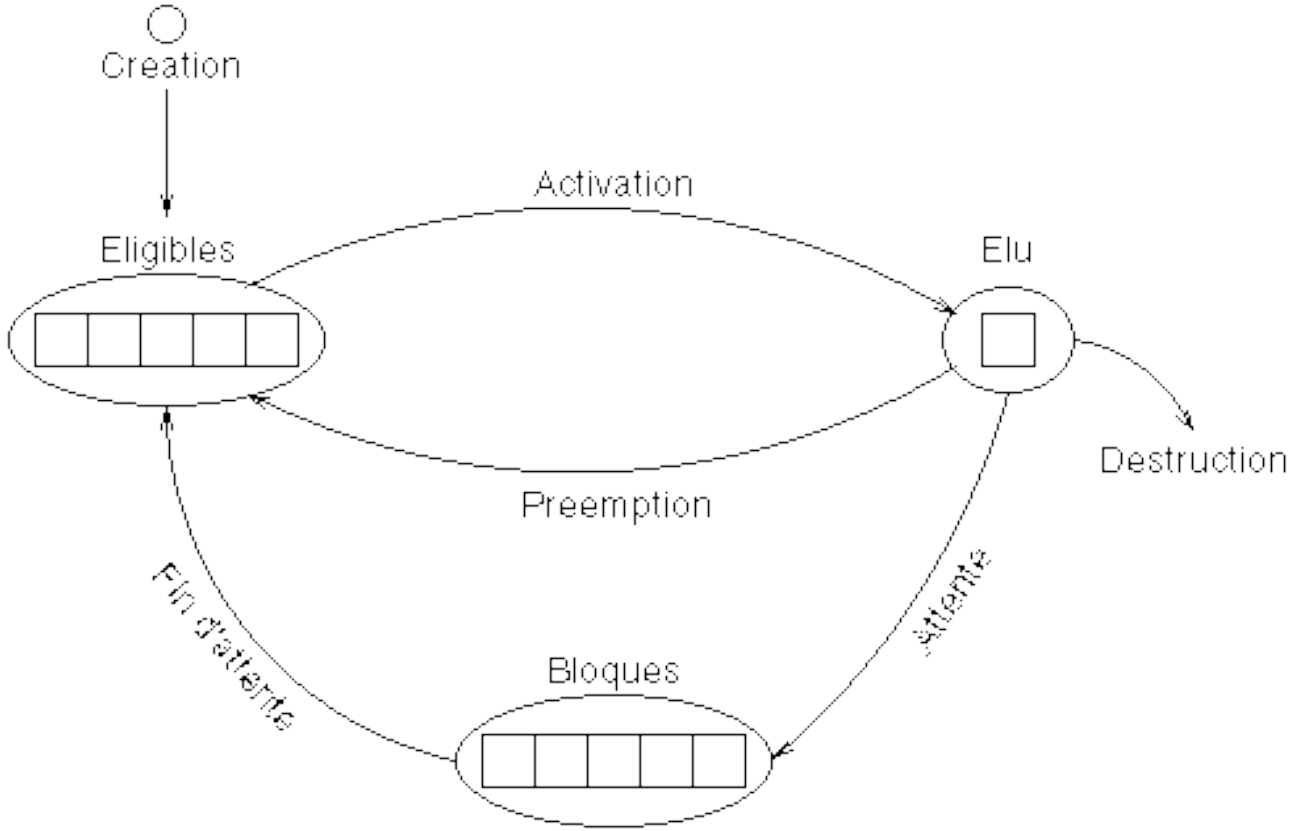
- Le multitâche permet de **paralléliser les traitements** par l'exécution simultanée de programmes informatiques.
- Exemples de besoins :
  - permettre à plusieurs utilisateurs de travailler sur la même machine.
  - utiliser un traitement de texte tout en naviguant sur le Web.
  - transférer plusieurs fichiers en même temps.
  - améliorer la conception : écrire plusieurs programmes simples, plutôt qu'un seul programme capable de tout faire, puis de les faire coopérer pour effectuer les tâches nécessaires.

Comment ça marche ?

- La **préemption** est la capacité d'un système d'exploitation multitâche **à suspendre un processus au profit d'un autre**.
- Le **multitâche préemptif** est assuré par l'**ordonnanceur** (*scheduler*), un service de l'OS.
- L'**ordonnanceur** distribue le **temps du processeur entre les différents processus**. Il peut aussi interrompre à tout moment un processus en cours d'exécution pour permettre à un autre de s'exécuter.
- Une **quantité de temps définie** (*quantum*) est attribuée par l'**ordonnanceur** à chaque processus : les processus ne sont donc pas autorisés à prendre un temps non-défini pour s'exécuter dans le processeur.

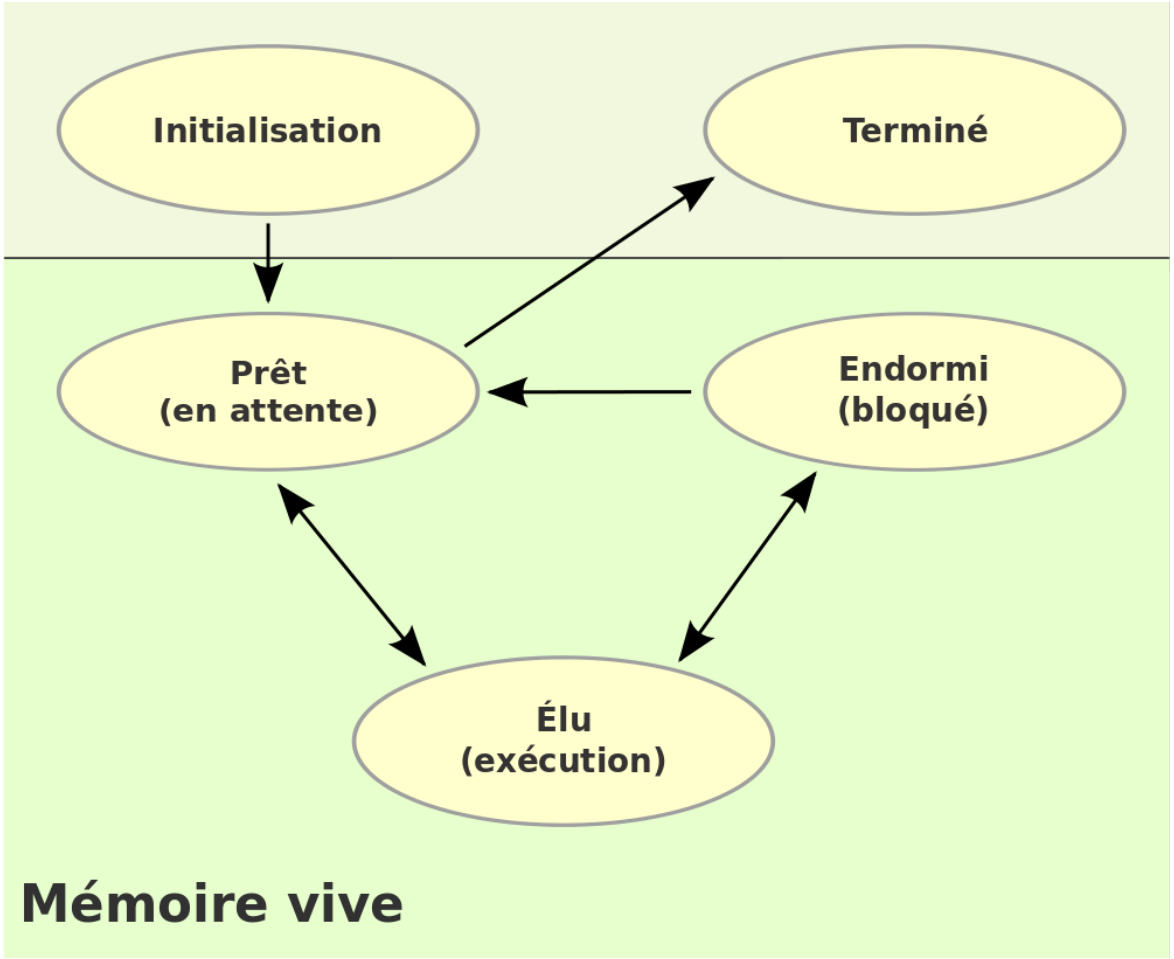
L'ordonnancement en action

Dans un ordonnancement (statique à base de priorités) avec préemption, un processus peut être préempté (remplacé) par n'importe quel processus plus prioritaire qui serait devenu prêt.



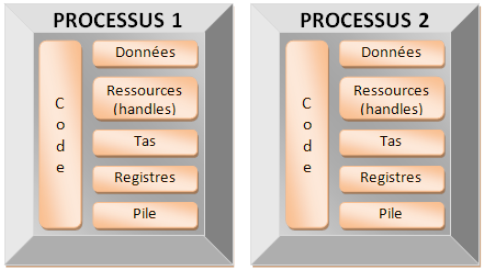
États d'un processus

- Ces états existent dans la plupart des systèmes d'exploitation :

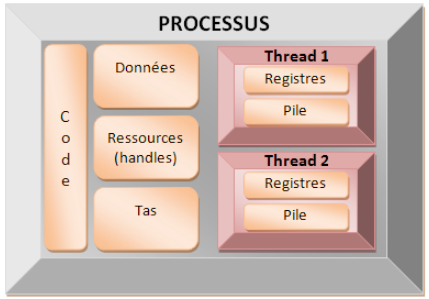


Processus lourd et léger

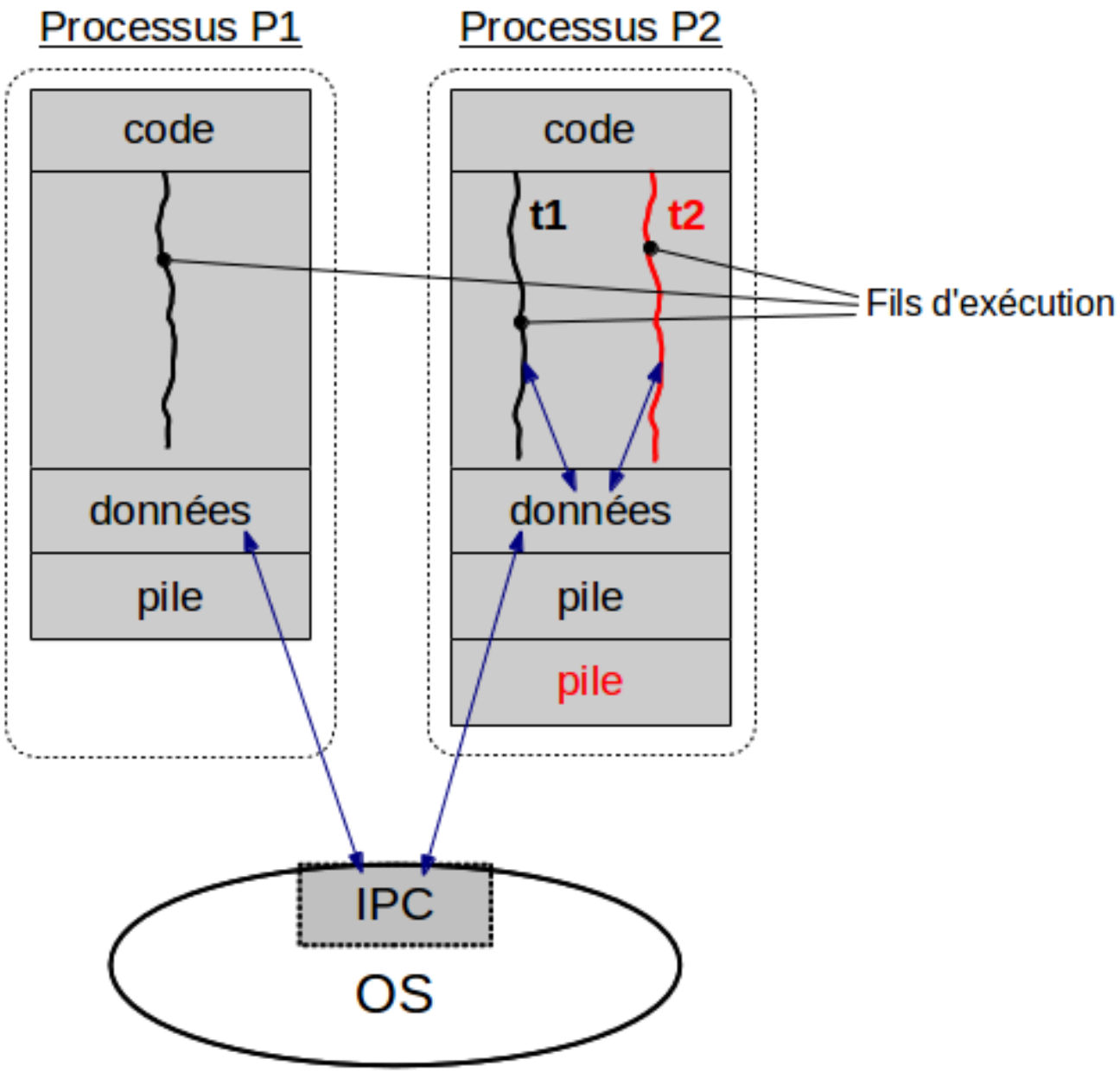
- *Rappel : L'exécution d'un processus se fait dans son contexte. Quand il y a changement de processus courant, il y a une commutation ou changement de contexte.*
- En raison de ce contexte, la plupart des systèmes offrent la distinction entre :
  - **« processus lourd »**, qui sont complètement isolés les uns des autres car ayant chacun leur contexte, et



  - **« processus légers » (*threads*)**, qui partagent un contexte commun sauf la pile (les *threads* possèdent leur propre pile d'appel).

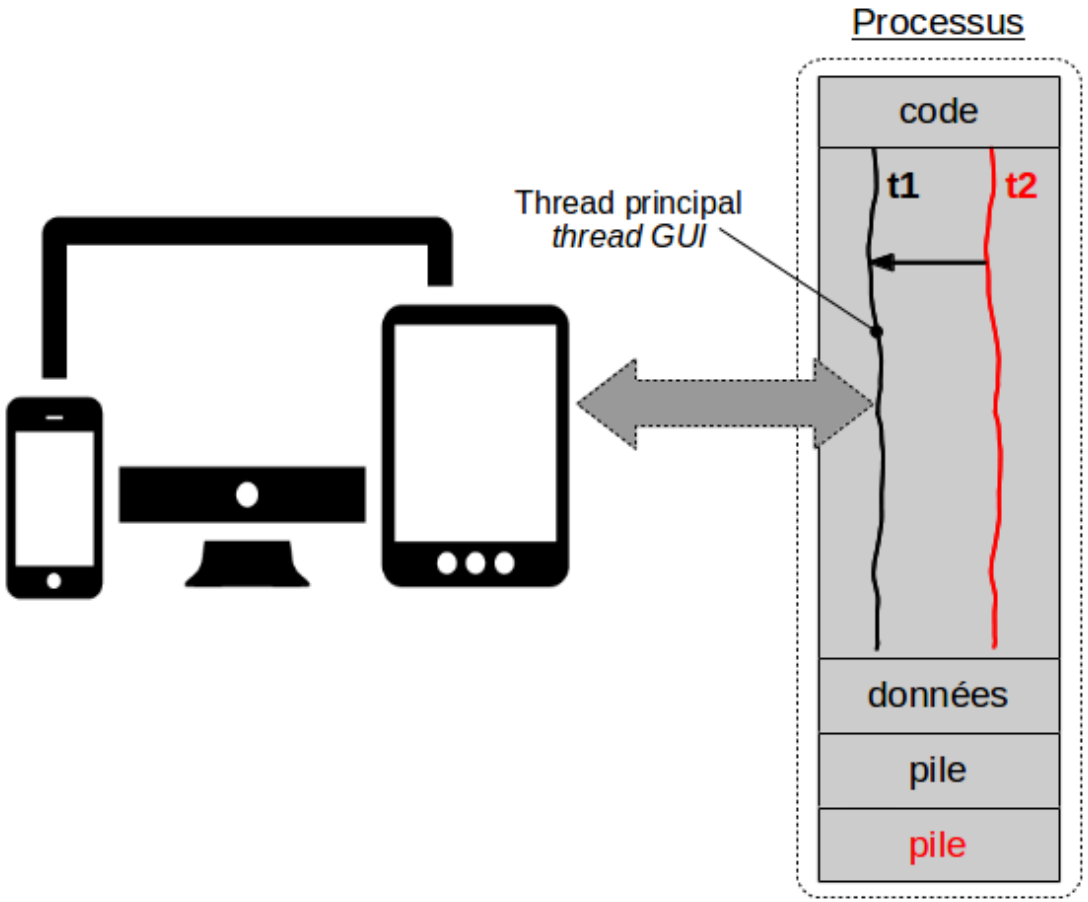


Processus lourds vs processus légers



Cas du thread GUI

- Seul le *thread* principal GUI (*Graphical User Interface*) peut accèder aux ressources graphiques (*widegts*). Il faudra alors prévoir une communication *inter-threads* si un *thread* non-gui désire afficher ou interagir avec l'interface graphique.



Synchronisation de tâches

Dans la programmation concurrente, le terme de **synchronisation** se réfère à deux concepts distincts (mais liés) :

- La **synchronisation de processus ou tâche** : mécanisme qui vise à bloquer l'exécution des différents processus à des points précis de leur programme de manière à ce que tous les processus passent les étapes bloquantes au moment prévu par le programmeur.
- La **synchronisation de données** : mécanisme qui vise à conserver la cohérence entre différentes données dans un environnement multitâche.

Les problèmes liés à la synchronisation rendent toujours la programmation plus difficile.

Communication Inter-Processus (IPC)

Les **communications inter-processus** (*Inter-Process Communication* ou **IPC**) regroupent un ensemble de mécanismes permettant à des processus concurrents (ou distants) de communiquer. Ces mécanismes peuvent être classés en trois catégories :

- les outils permettant aux processus de **s'échanger des données**
- les outils permettant de **synchroniser les processus**, notamment pour gérer le principe de section critique
- les outils offrant directement les caractéristiques des deux premiers (échanger des données et synchroniser des processus)

Synchronisation

Les mécanismes de synchronisation sont utilisés pour résoudre les problèmes de **sections critiques** et plus généralement pour bloquer et débloquer des processus suivant certaines conditions :

- Les **verrous** permettent de bloquer tout ou une partie d'un fichier
- Les **sémaphores** (et les **mutex**) sont un mécanisme plus général, ils ne sont pas associés à un type particulier de ressource et permettent de limiter l'accès concurrent à une section critique à un certain nombre de processus
- Les **signaux** (ou les **évènements**) permettent aux processus de communiquer entre eux : réveiller, arrêter ou avertir un processus d'un événement

L'utilisation des mécanismes de synchronisation est difficile et peut entraîner des **problèmes d'interblocage** (tous les processus sont bloqués).

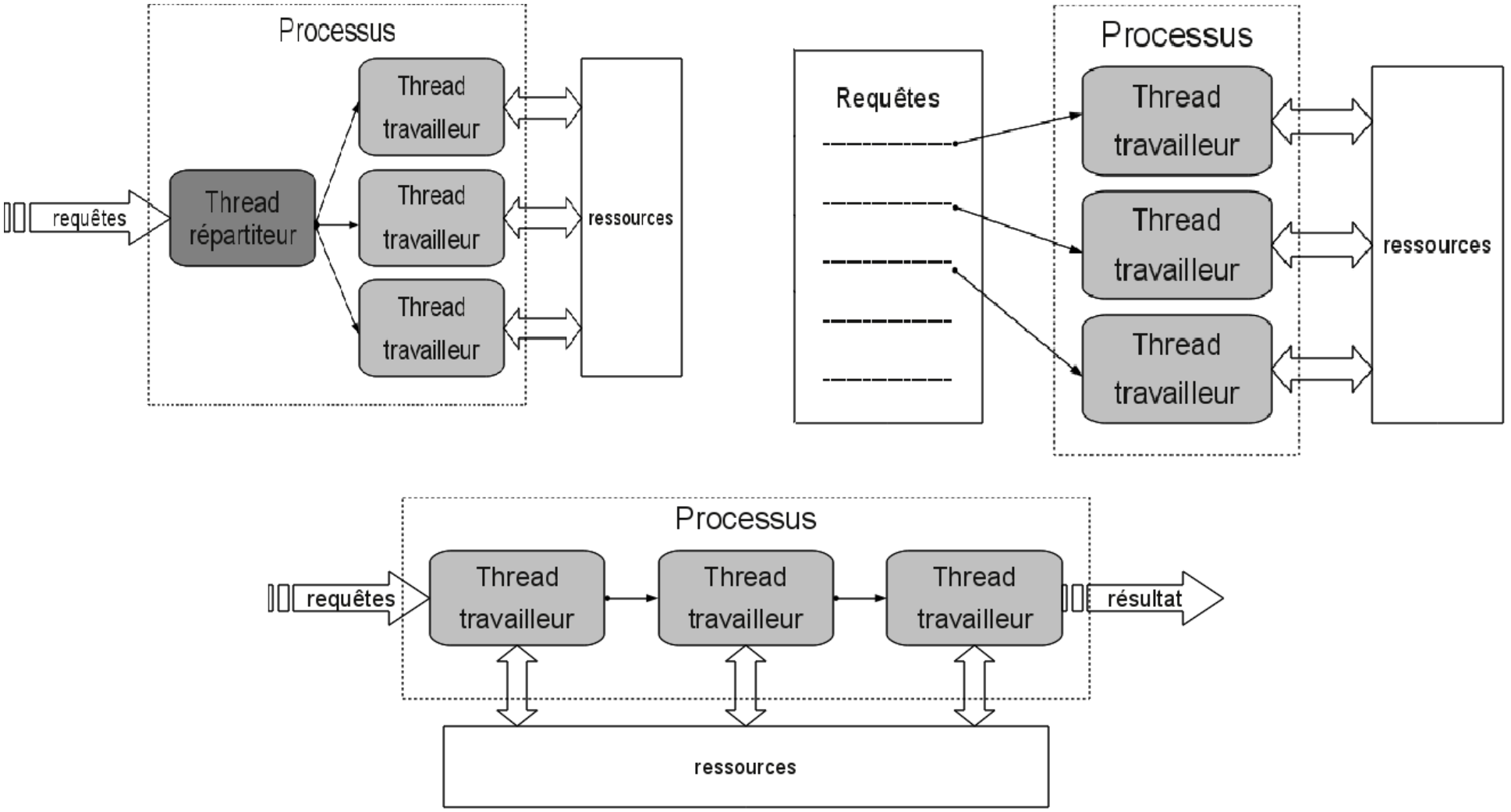
Échange de données et synchronisation

Ces outils regroupent les possibilités des deux autres et sont souvent plus simples d'utilisation.

- Les **segments de mémoire partagée** (*shared memory*), qui sont accessibles simultanément par deux processus ou plus.
- L'idée est communiquer en utilisant le principe des **files** (notion de boite aux lettres), les processus voulant envoyer des informations (**messages**) les placent dans la file ; ceux voulant les recevoir les récupèrent dans cette même file. Les opérations d'écriture et de lecture dans la file sont bloquantes et permettent donc la synchronisation.
- Ce principe est utilisé par :
  - les **files d'attente de message (*message queue*)** sous Unix,
  - les ***sockets* Unix ou Internet**,
  - les **tubes (nommés ou non)**, et
  - la transmission de **messages (*Message Passing*)** (DCOM, CORBA, SOAP, ...)

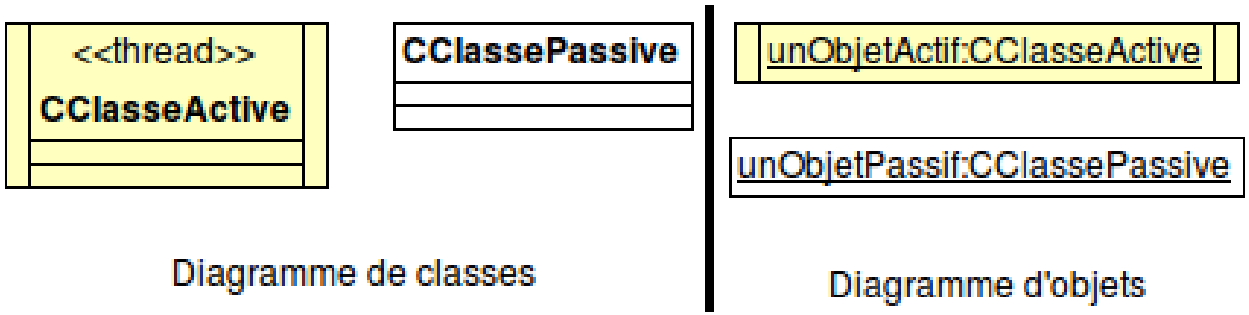
Modèles de programmation

On distingue généralement ces 3 modèles : **répartiteur/travailleurs ou maître/esclaves**, le **modèle en groupe** et le **modèle en pipeline**.



Classe active et objet actif

- Une classe active est une classe qui possède une méthode qui s'exécute dans un *thread*. Cela définit alors un **flot de contrôle distinct** (c'est le fil d'exécution). Une **instance d'une classe active** sera nommée **object actif**.
- UML fournit un repère visuel (bord en trait épais ou double trait) qui permet de distinguer les éléments actifs des éléments passifs. Il est conseillé d'ajouter le séréotype « thread ».



Synthèses I

- Processus : programme en cours d'exécution. C'est l'exécution d'une image composée de code machine et de données mémoire.
- Contexte : image d'un processus en mémoire auquel s'ajoute son état (registres, ...).
- Multitâche : exécution en parallèle de plusieurs tâches (*processus* ou *threads*).
- Commutation de contexte : passage de l'exécution d'un processus à un autre.

Synthèses II

- Multitâche préemptif : mode de fonctionnement d'un système d'exploitation multitâche permettant de partager de façon équilibrée le temps processeur entre différents processus.
- Ordonnancement : mécanisme d'attribution du processeur aux processus (blocage, déblocage, élection, préemption).
- Processus lourd : c'est un processus « normal ». La création ou la commutation de contexte d'un processus a un coût pour l'OS. L'exécution d'un processus est réalisée de manière isolée par rapport aux autres processus.

Synthèses III

- Processus léger : c'est un *thread*. Un processus lourd peut englober un ou plusieurs *threads* qui partagent alors le même contexte. C'est l'exécution d'une fonction (ou d'une méthode) au sein d'un processus et en parallèle avec les autres *threads* de ce processus. La commutation de *thread* est très rapide car elle ne nécessite pas de commutation de contexte.

- API : c'est une interface de programmation (*Application Programming Interface*) qui est un ensemble de classes, de méthodes ou de fonctions qui offre des services pour d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un *framework*.

- WIN32 : c'est l'API de Windows qui permet la programmtion système (création de processus ou de thread, communication inter-processus, ...). Elle est orientée *handle*.

Synthèses IV

- *System calls* : c'est l'API Unix/Linux composée d'appels systèmes pour la création de processus ou de thread, la communication inter-processus, ...). Elle est orientée *fichier*.
- POSIX : c'est une norme d'API que l'on retrouve notamment sous Unix.
- Section critique : section de code exécutée par une et une seule tâche en même temps
- Exclusion mutuelle : éviter que des ressources partagées ne soient utilisées en même temps par plusieurs tâches
- Chien de garde (*watchdog*) : tâche de fond assurant la protection contre le blocage de tâches

Synthèses V

- Mutex : technique permettant de gérer un accès exclusif à des ressources partagées
- Sémaphore : variable compteur permettant de restreindre l'accès à des ressources partagées