EPAI, Jérôme Frossard (2019)

ÉLÉMENTS DE PROGRAMMATION PARALLÈLE (MULTITHREADING)

Développer une interfaces utilisateur graphique

Programme et processus

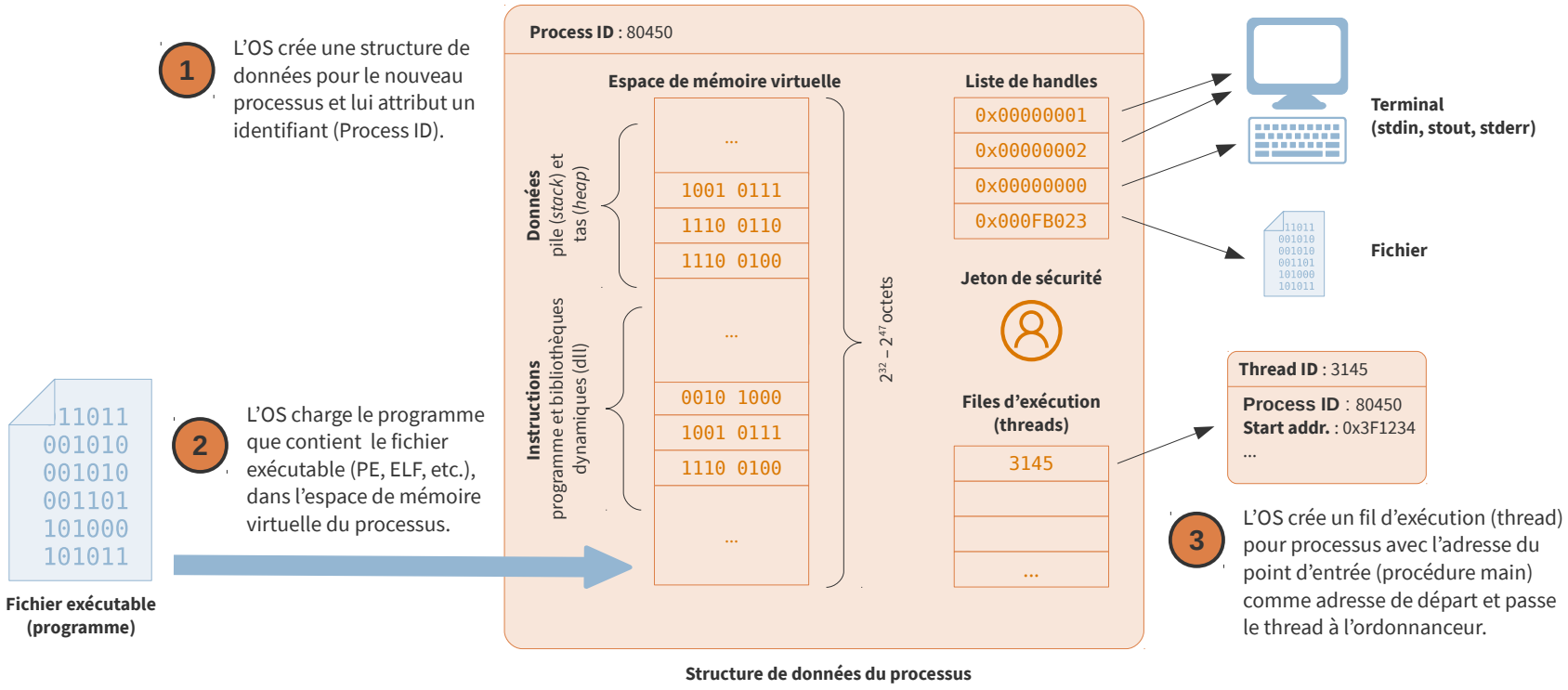
- **Un programme est une liste d'instructions** exécutables stockée sur un support numérique, par exemple :
  - Un fichier .exe sous Windows
  - Un fichier de type ELF sous Linux
- **Un processus est un programme en cours d'exécution,** représenté par une structure de données dans le noyau du système d'exploitation.
- La structure de données comprend notamment l'identifiant du processus (process ID ou PID) visible avec la commande ps de Linux ou dans l'onglet Détails du gestionnaire de tâches de Windows.

Structure de données d'un processus

- Dans le noyau du système d'exploitation, un processus est représenté par une structure de données qui comprend :
  - Un identificateur de processus (***process ID***).
  - Un espace de mémoire virtuelle qui contient les **instructions** du programme (.exe) et des bibliothèques dynamique (.dll), une pile (***stack***) et un tas (***heap***).
  - Une **liste de descripteurs** (*handle* ou *descriptor*) des différentes ressources systèmes (p. ex. les fichiers ouverts) accessibles par le ou les fils d'exécution (*threads*) du processus.
  - Un **jeton de sécurité** qui permet au système d'exploitation de déterminer ce que le processus à le droit de faire.
  - Au moins **un fil d'exécution**.

Exécution d'un programme

- Description simplifiée du lancement de l'exécution d'un programme par le système d'exploitation.



Mutlitâche (*multitasking*)

- Les systèmes d'exploitation modernes sont multitâches et peuvent donc gérer l'exécution de plusieurs programmes simultanément ou quasi simultanément.
- Si le nombre de fils d'exécution (*threads*) est plus grand que le nombre de processeurs, le système d'exploitation attribue successivement une tranche de temps (*time slice* ou *quantum*), de l'ordre de la dizaine de ms, à chacun d'eux.
- Grâce au multitâche, un programme peut être réalisé comme si toues les ressources du système étaient à sa disposition.
- **Le programmeur n'a pas à se soucier du multitâche.**

Programmation parallèle

- Il est parfois utile d'exécuter simultanément plusieurs sous-programmes pour l'accomplissement d'une tâche.
- Il est possible de réaliser ces sous-programmes comme des programmes séparés et de les exécuter simultanément (p. ex. : scripts CGI).
- Cela présente toutefois certains désavantages :
  - La gestion des processus est relativement lourde pour le système.
  - Le temps nécessaire au démarrage d'un processus peut être long par rapport au temps d'exécution du programme.
  - Les processus sont isolés les uns des autres, l'échange de données ou la synchronisation entre les processus est mal aisé.

Thread et multithreading

- Un fil d'exécution (***thread***) représente l'exécution d'une séquence d'instruction.
- À la création d'un processus, le système d'exploitation crée automatiquement un thread qui démarre au point d'entrée du programme (procédure *main*).
- L'ordonnanceur (***scheduler***) du système d'exploitation partage les ressources CPU entre les différents threads.
- Les systèmes d'exploitation modernes permettent de créer des threads supplémentaires dans un même processus pour exécuter plusieurs sous-programmes simultanément.
- **Le programmeur est responsable de la création de threads supplémentaires.**

Avantages et inconvénients

- Avantages du multithreading :
  - La création et la destruction de thread sont des opérations rapides et peu coûteuses par rapport à la gestion de processus.
  - Les threads d'un même processus partagent le même espace de mémoire virtuelle et peuvent donc partager des variables pour s'échanger des informations ou pour se synchroniser.
- Inconvénients :
  - La programmation concurrente (programmation parallèle) est **vraiment très difficile** et les problèmes potentiels sont nombreux (*deadlocks*, *race conditions*, etc.).
  - Si le nombre de threads est grand par rapport au nombre de processeurs, le coût des changements de contexte peut excéder le gain de la parallélisation.

Recommandations

- À moins de n'avoir aucune autre option, on ne devrait pas utiliser directement une fonction de création de thread.
- **L'utilisation directe de threads dans le code d'un GUI est toujours une mauvaise idée.**
- Il est préférable d'utiliser des threads de manière indirecte :
  - Utilisation de fonctions d'**E/S asynchrones**. Les threads sont gérés par la bibliothèque d'E/S et non par le programme lui-même (voir **programmation asynchrone et promesses**).
  - Utilisation d'un framework pour l'exécution de procédure qui demande une **utilisation intensive du CPU** dans un ***worker thread*** (p. ex. : l'API Web Workers en JavaScript).
- Dans tous les cas, l'échange d'informations entre threads doit être strictement canalisé et limité autant que possible.