Définition 8.1 - *processus*

Un *processus* est un programme en cours d'exécution. Il occupe la mémoire selon l'organisation suivante

- le *segment de donnée*, qui constitue une zone de mémoire "fixe" pour stocker les *variables globales*, ainsi que les *constantes*.
- le *tas*, où sont stockées les variables allouées dynamiquement.
- la *pile*, une zone de mémoire de taille variable où sont stockés
  - les *blocs d'activation* pour chaque appel de fonction contenant les variables locales aux fonctions
  - un *pointeur d'instruction* vers le segment de code qui contient l'adresse de la prochaine instruction dans le *segment de code*
- le *segment de code*, où sont stockés les isntructions du programme

Remarque 8.2

Deux processus distincts ne partagent pas leur zones de mémoire.

Remarque 8.3

Pour les faire communiquer, on a besoin de faire des *appels système*. Par exemple, on utilise un ficher utilisé par deux processus en lecture/écriture.

Remarque 8.4

On dit que le *contexte d'un processus* est *lourd* car sa création est son activation sont coûteux.

Définition 8.5 - *instruction atomique*

Une *instruction atomique* est une instruction qui s'effectue sans interruption.

Définition 8.5 - *fil d'exécution (thread)*

Un *fil d'exécution* (dit *thread*) est une séquence d'*instructions atomiques* : instructions s'effectuant sans interruption.
Généralement, un *fil d'exécution* est matérialisé par une fonction en cours d'exécution.

Remarque 8.6 - *création d'un fil d'exécution*

Pour créer un fil d'exécution, on définit sa *tâche* sous la forme d'une fonction et on lui associe une pile listant les instructions à traîter.

Remarque 8.7

Un processus peut contenir plusieurs fils d'exécutions :

- un désigné *principal*
- tous ceux créés en plus : on a autant de piles dans la mémoire que de fils d'exécution.

voir figure

Plusieurs fils d'exécution peuvent partager des données stockées dans le tas ou le segment de données.

Théoriquement, ils ont accès avec les adresses mémoires aux piles des autres fils, mais ce n'est pas utilisé en pratique.

On dit que le *contexte* des fils d'exécution est *léger* car la création, l'activation et la communication est facilitée et peu coûteuse.

Les fils d'exécutions s'exécutent indépendamment les uns les autes.

Un fil seul est *séquentiel*. On dit qu'une programme est *concurrent* lorsqu'il a plusieurs fils d'exécution (concurrent au sens de "se passe en même temps").

L'exécution repose sur un *entrelacement non déterministe* des fils d'exécution.