M2101 - Architecture et programmation bas niveau

Cours n°3 : Processus

Victor Poupet

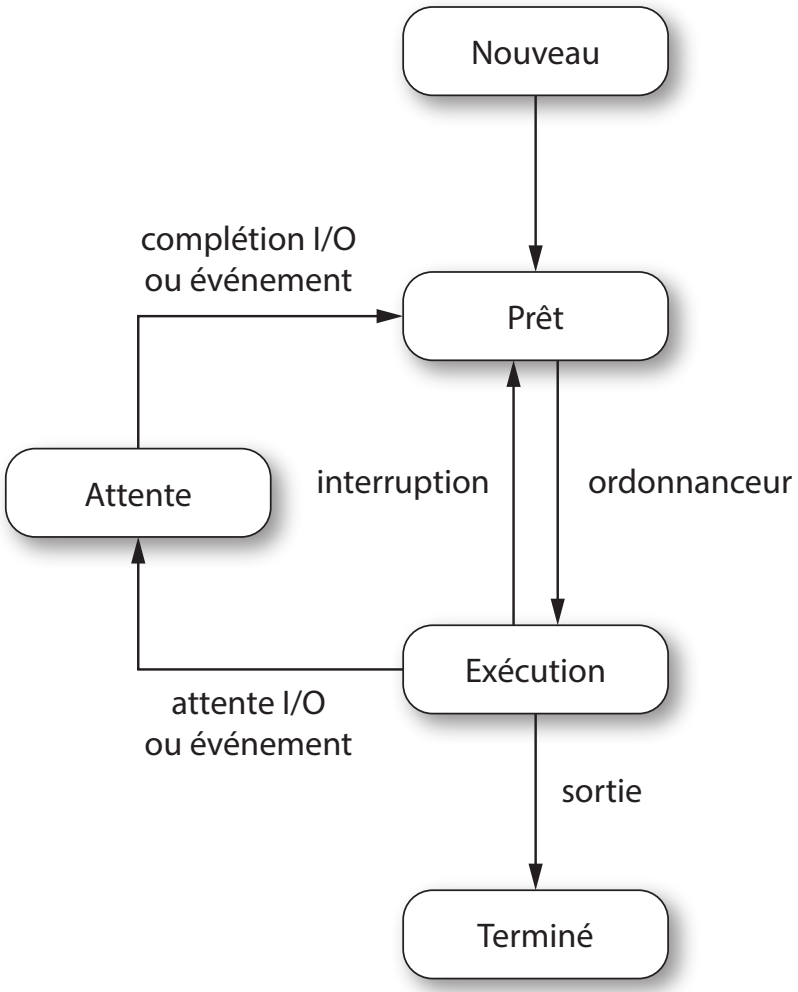
26 Février 2018

Parallélisme

Les OS modernes sont tous capables d'exécuter plusieurs tâches en parallèle

- chaque tâche est réalisée par un *processus*
- en réalité un processeur ne peut exécuter qu'un seul processus à la fois à un instant donné
- pour les faire avancer en parallèle, le processeur alloue de petits intervalles de temps à chaque processus, en alternance (ordonnancement)

Cycle d'un processus



Les processus ont 5 états possibles d'exécution :

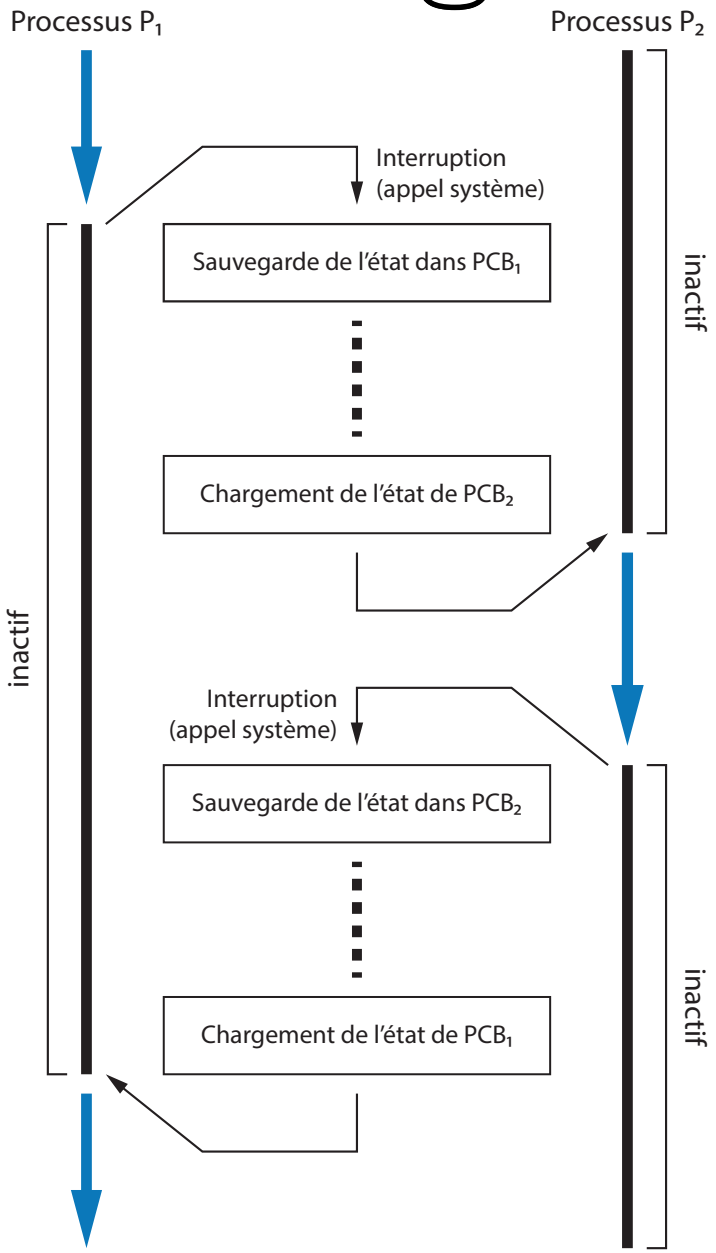
- **Nouveau :** en cours de création
- **Prêt :** toutes les ressources nécessaires sont disponibles mais pas actuellement exécuté par le CPU
- **Exécution :** en cours d'exécution par le CPU
- **En attente :** ne peut pas poursuivre son exécution (attente clavier, fichier, message d'un autre processus, etc.)
- **Terminé :** l'exécution est finie

Bloc de contrôle

Le *bloc de contrôle* d'un processus (*process control bloc* ou PCB) est une structure de données utilisée par l'OS pour représenter ses informations :

- Identification
  - identifiant du processus (PID)
  - identifiant du processus parent (PPID)
  - identifiant de l'utilisateur et du groupe exécutant le processus (UID et GID)
- État du processeur
  - registres
  - pointeurs de pile et exécution
- Contrôle du processus
  - état d'exécution (prêt, suspendu, etc.)
  - descripteurs de fichiers ouverts
  - privilèges
  - comptabilité (temps CPU, date d'exécution, *etc.*)

Changement de processus



Pour changer le processus en cours d'exécution, le processeur doit :

- sauvegarder le PCB du processus en cours
- charger le PCB du nouveau processus
- reprendre l'exécution du nouveau processus

Ce transfert peut être réalisé après une interruption de l'exécution du premier processus à l'aide d'un appel système.

Anatomie d'un processus

La mémoire occupée par un processus est divisée en plusieurs sections :

- une section `text` contenant les instructions (en langage machine) à exécuter par le processus
- une section `data` contenant les variables globales et statiques, allouées avant d'exécuter la fonction `main`
- le tas (*heap*) utilisé pour l'allocation dynamique des variables, géré à l'aide de fonctions d'allocation (`malloc`, `free`, *etc.*)
- la pile (*stack*) utilisée pour stocker les variables locales, les arguments et la valeur de retour des procédures exécutées par le programme

En plus de ces données, si le programme est interrompu, il faut également sauvegarder le compteur d'exécution ainsi que les valeurs contenues dans les registres du processeur

size

```c
int tab[5000];
int tab2[1000] = {1};

int main() {
  int tab2[20000];
  return 0;
}
```

```
$ size a.out
text data bss   dec   hex  filename
1072 4276 20032 25380 6324 a.out
```

La fonction **size** permet d'obtenir (entre autres) la taille des sections **text** et **data** correspondant à un exécutable

- La section **data** d'un exécutable est divisée en deux parties :
  - **data** correspondant aux variables globales ou statiques initialisées
  - **bss** correspondant aux variables globales ou statiques non initialisées ou initialisées à 0 (qui ne prennent donc pas de place dans l'exécutable)
- L'espace des variables globales est préparé par le compilateur, et réservé à la création du processus (la zone correspondant à **bss** est initialisée à 0)
- Les variables globales ne peuvent pas avoir une taille variable (définie à l'exécution)

text

La section `text` de la mémoire contient la séquence d'instructions à exécuter

- Chargée au démarrage du programme et protégée en écriture
- Un registre du processeur contient l'adresse de l'instruction en cours (*program counter* ou PC)

Limites

```
$ ulimit -a

core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
file size (blocks, -f) unlimited
max locked memory (kbytes, -l)
   unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 256
pipe size (512 bytes, -p) 1
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 709
virtual memory (kbytes, -v) unlimited
```

La fonction **ulimit** permet d'obtenir les limitations de taille imposées par le système d'exploitation.

Allocation dynamique

```
int *fibo (int n) {
  int *t = malloc(sizeof(int)*(n+1));
  t[0] = 1;
  t[1] = 1;
  for (int i=2; i<=n; i++) {
    t[i] = t[i-1] + t[i-2];
  }
  return t;
}

int main() {
  int *t = fibo(100);
  int a = t[10];
  int b = t[100];
  free(t);
  printf("10: %d, 100: %d\n", a, b);
}
```

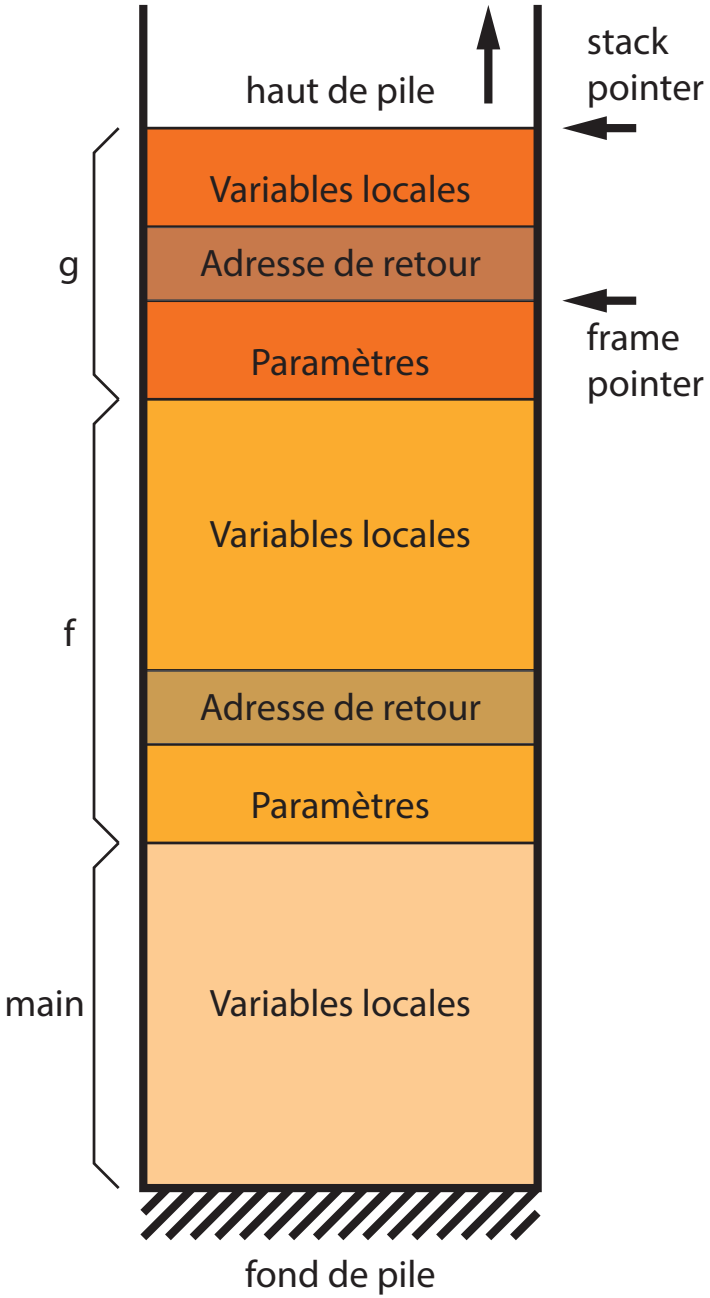
Les fonctions de la famille **malloc** permettent de réserver de l'espace dynamiquement (dans le tas)

- La fonction renvoie un pointeur indiquant où se trouve le bloc de mémoire libéré

La fonction **free** permet de marquer la mémoire allouée par **malloc** comme étant libre

La taille du tas disponible pour un processus n'est en général pas limitée, et c'est donc là que doivent être placés les objets de grande taille.

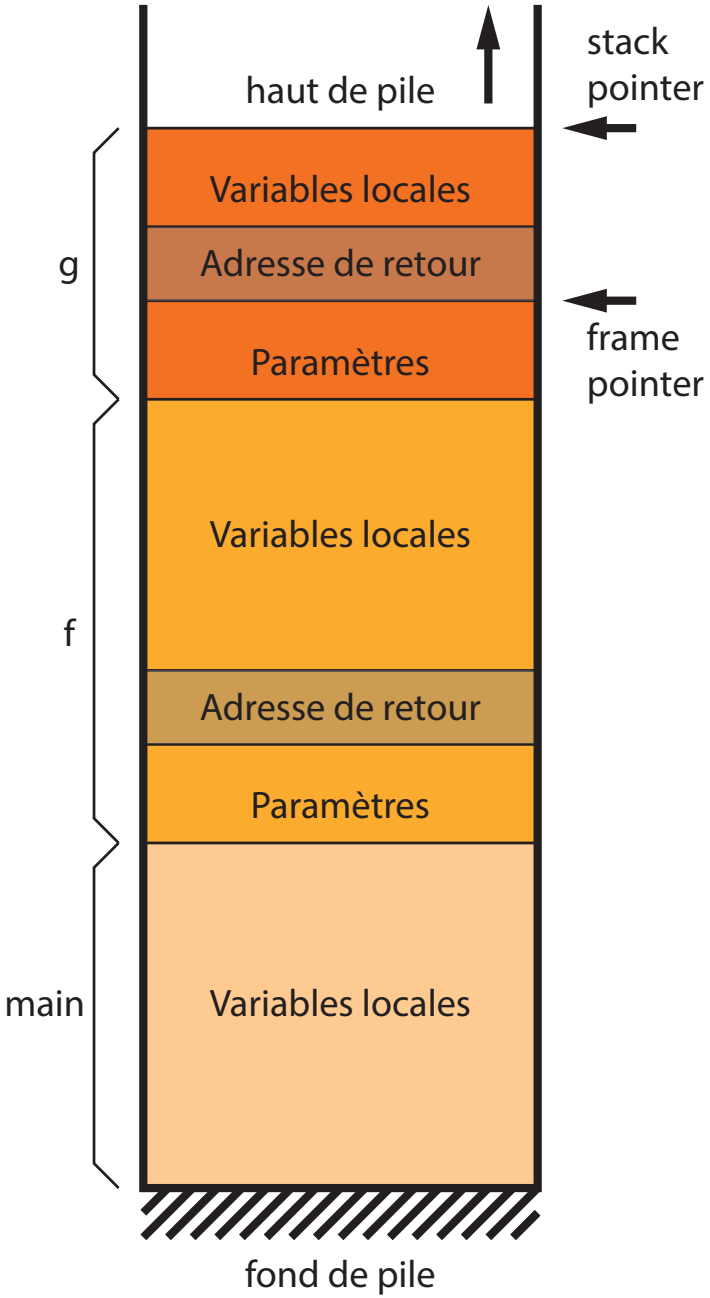
La pile



La pile d'exécution est une structure LIFO (*last in, first out*)

- divisée en blocs (*frames*) correspondant aux fonctions en cours
- lorsqu'une sous-fonction est appelée, l'état de la fonction en cours est sauvegardé, et un nouveau bloc est placé sur la pile pour la sous-fonction
- lorsqu'une fonction termine, son bloc est supprimé de la pile et la fonction qui l'a appelée reprend son exécution

La pile



Pour chaque fonction en cours d'exécution, la pile contient :

- l'adresse de retour qui correspond à l'instruction où reprendre l'exécution de la fonction précédente lorsque la fonction courante termine
- les variables locales de la fonction
- les arguments passés à la fonction
- environnement à restaurer lorsque la fonction termine (privilèges, états de certains registres, etc.)

Lorsque la fonction termine, le résultat est placé sur la pile pour être traité par la fonction précédente.

Pile

- Fréquemment utilisée donc placée sur de la mémoire plus rapide
- Variables libérées automatiquement
- Connexe : ne se fragmente pas et facile d'accès
- Gestion automatique de la portée des variables
- Récursivité facile

Tas

- Espace illimité
- Objets accessibles globalement
- Doit être libéré explicitement
- Objets peuvent être redimensionnés (`realloc`)