

Processus - Création

Programmation Système — R3.05

C. Raïevsky



Département Informatique
BUT Informatiques 2^{ème} année

Définition Définition

Processus — Définition(s)

Processus

- ▶ Structure de donnée qui contient toute l'information nécessaire à l'exécution d'un programme
- ▶ Instance d'un programme en cours d'exécution

Contient (entre autre) :

- ▶ Code exécutable et données du programme
- ▶ Attributs de sécurité (propriétaire, permissions)
- ▶ État des ressources système qu'il utilise (fichiers, etc.)
- ▶ Table des pages
- ▶ Informations relatives à l'ordonnement
- ▶ Un état du processeur

2 / 15

Création État initial de la mémoire

Structuration initiale de la mémoire

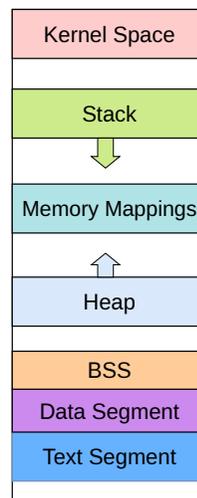
Zones mémoire prédéfinies par l'OS :

“Statiques” :

- ▶ Text segment (ro)
- ▶ Data segment
- ▶ BSS segment

“Dynamiques” :

- ▶ Heap (Tas, free-store)
- ▶ Stack (Pile)
- ▶ Memory Mapping segment



3 / 15

Création Heap & Stack

Heap != Stack

Tas – Heap ou Free Store

- ▶ Allocation dynamique de mémoire
- ▶ Au cours de l'exécution
- ▶ Explicitement par le programme
- ▶ en utilisant `malloc` et autres

Pile – Stack

- ▶ Utilisée pour les appels de fonction
- ▶ Pas de manipulation explicite par le programmeur (du moins en C)
- ▶ Instructions processeur dédiées

4 / 15

Outils pour examiner la mémoire d'un processus

Avant lancement du processus :

- ▶ nm
- ▶ objdump
- ▶ readelf

Une fois le processus lancé :

- ▶ pmap
- ▶ /proc/n°process
- ▶ ps pstree

Démo :

hello.c

Création d'un processus en C sous Linux

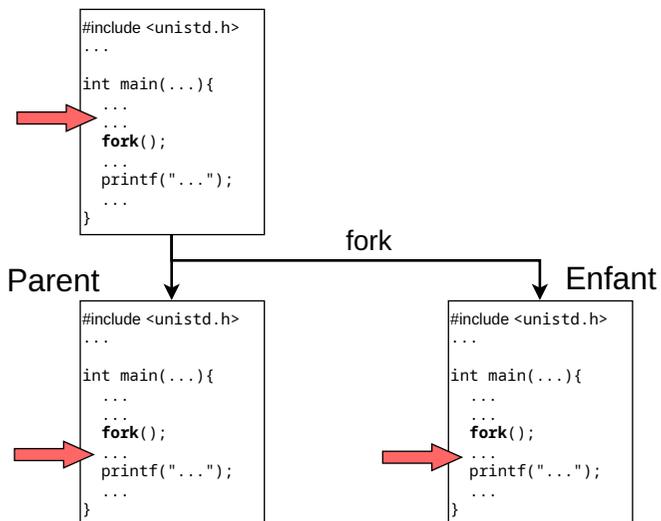
fork();

fork ne crée pas un nouveau processus **vide**

fork **duplique** le processus courant

- ▶ Le nouveau processus est appelé *processus enfant*
- ▶ Le processus qui a appelé fork est le *processus parent*

Création d'un processus en C sous Linux



Création d'un processus en C sous Linux

fork duplique dans le processus enfant :

- ▶ Tout l'espace mémoire du parent
- ▶ Les descripteurs de fichier
- ▶ Les "memory mappings"
- ▶ Et beaucoup d'autres chose (cf. man 2 fork)

Réplication de l'espace mémoire, lourd ?

"Copy on write" pour les pages du processus créé ⇒ mécanisme efficace

Attribution d'un identifiant **UNIQUE** au nouveau processus par l'OS

Exemple de création de processus sous UNIX

:O{ :|:& }::

```

1  int main(int argc, char* argv[]) {
2      pid_t id;
3      id = fork();
4      /** From here, 2 processes execute the code **/
5
6      if (id == 0) {
7          /* Only the *child* process executes this code */
8          printf("Child_process.\n");
9
10     } else {
11         /* Only the *parent* process executes this code */
12         printf("Parent_process.\n");
13     }
14
15     return EXIT_SUCCESS;
16 }

```

9 / 15

Création d'un processus en C sous Linux

Comment savoir si on est dans l'enfant ou le parent ?

Que se passe-t-il dans les cas suivants

- ▶ Qu'arrive-t-il aux variables globales ?
 - ▶ Si un des processus les modifie, que se passe-t-il pour l'autre ?
- ▶ Qu'arrive-t-il à l'enfant si le parent meurt ?
 - ▶ Meurt-il de chagrin ?
 - ▶ Qui l'adopte ?

10 / 15

Exemple de remplacement du processus courant par un autre programme

Comment lancer un **nouveau programme** ?

Alors que fork ne fait que **dupliquer** un processus existant.

```

1  int main(void)
2  {
3      execlp("hello", "hello", "arg1", "arg2", NULL);
4
5      return EXIT_FAILURE; /* ? */
6  }

```

Démo : sepuku.c

12 / 15

Lancement d'un programme

Utilisation du processus enfant pour exécuter un autre programme

Après un `fork` le code des deux processus reste **identique**

Pour exécuter un autre programme : `execlp`

- ▶ Remplacement de toutes les zones mémoire (text, data, bss, et stack)
- ▶ Suppression des Memory mappings

- ▶ `execlp` ne retourne pas lors d'un succès
- ▶ puisque le code appelant est écrasé

Pour un processus, appeler `execlp` c'est se suicider

Exemple : Exécution d'un programme par le processus enfant

```

1 int main(void)
2 { pid_t ident = fork();
3
4   switch (ident) {
5     case -1: perror("fork");
6             return EXIT_FAILURE;
7
8     case 0: execlp("unExecutable", "unExecutable", "arg1", "arg2", NULL);
9             return EXIT_FAILURE; /* ? */
10
11    default: codeParent();
12  }
13  return EXIT_SUCCESS;
14 }

```

Démo : launchHello.c

Lancement d'un nouveau programme

