241 fork et création de processus

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Principes de fork

`fork(2)`

- Crée une **copie** de l'appelant
- Parent et enfant auront le **même code**
- Ils continueront leur exécution **indépendamment** l'un de l'autre
- Le parent reconnaît son enfant nouvellement créé

Question

- Comment l'enfant reconnaît son parent ?

Algorithme de fork

si *ressources système insuffisantes* **alors**
    positionner errno;
    retourner -1;
fin
obtenir nouvelle entrée dans la table des processus;
obtenir nouveau numéro de processus;
initialiser table[enfant];
marquer état enfant en cours de création;
« copier » segments mémoire du parent dans l'espace du nouveau;
incrémenter le décompte des fichiers ouverts;
marquer état enfant prêt;
si *processus en cours est le parent alors* **alors**
    retourner numéro enfant;
sinon
    retourner 0;
fin

Points clés de fork

- Parent et enfant partagent le même code
- Enfant a une copie des données du parent
- Parent et enfant partagent les fichiers ouverts
- La valeur de retour de `fork` permet de différencier parent et enfant

Exemple de fork

```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main() {
  pid_t pfils;

  printf("Je suis %d, je commence\n", getpid());
  pfils = fork();
  if (pfils == -1) {
    perror("Echec du fork");
  } else if(pfils == 0) {
    printf("Je suis %d, le fils de %d\n", getpid(), getppid());
  } else {
    printf("Je suis %d, le pere de %d\n", getpid(), pfils);
  }
  printf("Je suis %d, je finis\n", getpid());
}
```

Plusieurs forks

```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main() {
  printf("Gen 1\n");
  fork();
  printf("Gen 2\n");
  fork();
  printf("Gen 3\n");
  fork();
  printf("Gen 4\n");
}
```

Copie de la mémoire

La mémoire est « copiée » telle quelle

Il y a des optimisations

- La copie de la mémoire est paresseuse
- Et souvent, elle est juste partagée

→ Mais le système fait semblant que oui

Il n'y a pas de sémantique particulière

- Les zones sont toutes « copiées »
- code, pile, tas, bibliothèques, etc.

→ Un processus est responsable de l'organisation de sa mémoire

Attention aux tampons en espace utilisateur

- Les effets peuvent être surprenants

→ Pensez à `fflush(3)` avant

Forkbomb

Dénis de service

- Demande infinie de création de processus

→ **Famine** CPU, mémoire, table des processus

- Le nombre de demandes croît exponentiellement

→ Chaque processus en engendre 2

- Il est difficile de guérir

→ Plus assez de ressource pour lancer un processus qui nettoie tout ça

→ Dès qu'un processus est tué, un autre prend sa place

Exemples

- En C « `for(;;){fork();}` »
- En shell « `:(){ :|:& };:` »

Prévention forkbomb

Limiter le nombre maximal de processus par utilisateur (ou autre)

- `ulimit -u` commande interne du shell
- `/etc/security/limits(conf)` configuration globale (PAM)
- `setrlimit(2)` appel système sous-jacent
- `/proc/PID/limits` voir les limites de chaque processus

Et si jamais...

- `pkill -STOP -u john` puis `pkill -KILL -u john`
- ou redémarrer la machine
  (et mettre des limites pour la prochaine fois!)

Question

- Pourquoi `killall nomcommande` ne fonctionne pas directement ?
- Comment encore c'est possible en 2020 ?

Appel système Linux `clone`

- `clone(2)` appel système similaire à fork
- Évite certaines limitations de l'API de `fork`
- `clone3(2)` version moderne de clone avec une encore meilleure API
- Utilisé pour processus, threads et autres bêtes hybrides

Contexte d'exécution

Contrôle très précis du contexte d'exécution

- Partages mémoires
- Espaces de noms
- Cgroups
- En particulier utilisé pour les conteneurs Linux