Système d'exploitation

Processus

Florent Devin

Ecole Internationale des Sciences du Traitement de l'Information

Plan

Rappel

Processus systèmes

- Pas contrôlé par un terminal
- Propriétaire *root*
- Résident en mémoire centrale en attente de demande
- Assure des services pour tous les utilisateurs
- Peuvent être lancé au démarrage, ou par l'administrateur

Création d'un processus

Création

- Création : appel système `int fork ();`
- Crée tous les processus
- Notion de père/fils
- Notion de PID (unique)
- PID 0 : créé au démarrage avant `init`

Contexte

Contexte d'un processus

- Environnement utilisateur
  - zone programme (code à exécuter) : partageable entre plusieurs processus
  - zone données (variables) : Possibilité d'utilisation de segment de mémoire partagée
  - zone pile utilisateur (pour les variables locales et les appels de fonctions) : modifiable dynamiquement

Contexte

Contexte d'un processus

- Environnement matériel
  - compteur ordinal
  - pointeur de pile
  - registres de travail
  - registres de données
  - registres pour la mémoire virtuelle
  - …

Contexte

Contexte d'un processus

- Environnement système
  - Table des processus
  - Zone u : donnée privé au processus uniquement manipulable par le noyau
  - Table des régions (permet l'utilisation de mémoire partagée)

Mode

Mode d'exécution

- Mode utilisateur : Accès uniquement à son espace d'adressage
- Mode noyau ou système (*kernel mode*) : Exécution d'instructions propre au système, possibilité d'accéder à d'autre espace d'adressage

États possibles

État d'un processus sous Unix

- Naissance d'un processus : ni prêt, ni endormi; état initial de tous processus
- Prêt mais le swappeur doit le transférer en mémoire centrale pour le rendre éligible. (ce mode est différent dans un système à pagination).
- Exécution en mode utilisateur
- Exécution en mode noyau
- Éligible (prêt à s'exécuter)

États possibles

État d'un processus sous Unix

- Endormi en mémoire centrale
- Endormi en zone de swap (sur disque par exemple)
- Passage du mode noyau au mode utilisateur
- Zombie : réalisation d'un exit; uniquement dans la table des processus où il est conservé le temps pour son processus père de récupèrer le code de retour et d'autres informations de gestion (coût de l'exécution sous forme de temps, et d'utilisation des ressources)

Explication de la commande `ps`

`ps -l`

- F : localisation du processus
  - 0 : Hors de la mémoire centrale
  - 1 : En mémoire
  - 2 : Processus système
  - 4 : Verrouillé (en attente d'E/S)
  - 8 : Vidage
- `UID` : User `IDentifier`
- `PID` : Process `IDentifier`
- `PPID` : Parent Process `IDentifier`

Explication de la commande ps

ps -l

- S : État du processus
  - O : non existant
  - S : sleeping
  - W : waiting
  - R : running
  - Z : zombie
  - X : en évolution
  - P : pause
  - I : attente d'entrée au terminal ou input
  - L : attente d'un fichier verrouillé ou locked)

Explication de la commande ps

ps -l

- PRI : Priorité du processus
- NI : Valeur de Nice
- SZ : Taille du processus
- WCHAN : Attente du processus
- TTY : Terminal associé
- TIME : Temps d'exécution cumulé
- CMD : Commande lancée

Gestion du swap

Rôle du swap

- Gérer par le processus 0
- Allocation du swap
- Transfert vers la mémoire
- Transfert hors de la mémoire

Allocation du swap

Allocation

- Gestion différente d'un disque
- Ensemble contigu de blocs de taille fixe
- Utilisation des fonctions
  - `swapalloc (int taille);` : demande d'unités d'échange
  - `swapfree (int adresse, int taille);` : demande de libération d'unités
- Possibilité d'avoir plusieurs unités d'echange
- Création et destruction dynamique

Transfert vers la mémoire

Transfert

- Examen des différents processus prêt présents sur le disque
- Choix du plus ancien
- Transfert de celui-ci : allocation de mémoire, lecture depuis le disque, libération de l'espace utilisé en swap
- Exécution jusqu'à
  - Plus de processus prêt sur le disque
  - Plus de place en mémoire

Transfert hors de la mémoire

Transfert

- Transfert en priorité des processus bloqué, puis les prêts
- Aucun transfert de processus Zombie, ni verrouille par le SE
- Transfert d'un processus prêt : uniquement si il est présent en mémoire depuis un certain temps (en général 2s)
- Si pas de possibilité : "re-scan" toutes les secondes

Problème

Étreinte fatale

- Tous les processus en mémoire sont “endormi”
- Tous les processus “prêt” sont transférés en swap
- Plus de place en mémoire
- Plus de place en échange sur le disque

Deux autres possibilités

Autres types de transfert

- Création d'un fils : possibilité de créer e fils en swap, lorsqu'il n'y a plus d'espace
- Accroissement de la taille d'un processus : Transfert du processus en swap avec l'allocation adéquate; au retour dans la mémoire le processus occupe plus de place

Exécution interactive

Exécution en avant plan

1. Lecture de la commande (ex : `./bidon`)
2. Duplication du shell via la commande `fork`. Existence de deux shells
3. Recouvrement du segment de texte (programme), par le code approprié
4. Récupération du numéro du processus fils par le shell initial
5. Attente de la fin d'exécution du processus fils (`wait (etat)`)

Exécution en arrière plan

Exécution en arrière plan

1. Lecture de la commande (ex : `./bidon`)
2. Duplication du shell via la commande `fork`. Existence de deux shells
3. Recouvrement du segment de texte (programme), par le code approprié
4. Récupération du numéro du processus fils par le shell initial
5. Émission d'un prompt, puis reprise de l'activité

Variables

Portée des variables

- Processus shell : pas de transmission de variable (exception pour les variables d'environnement)
- Possibilité de transmettre des variables vers le fils : `export`
- Possibilité de masquer une variable du père dans le fils : `unset`
- Impossibilité "physique" d'exporter une variable fils vers le père

pid_t fork (void)

Création d'un processus

- Processus fils
  - partage le segment de texte du père
  - dispose d'une copie de son segment de donnée
  - hérite du terminal de contrôle
  - hérite d'une copie des *file descriptor* ouverts
  - n'hérite pas du temps d'exécution, ni de la priorité, mais du *nice*
  - n'hérite pas des signaux en suspens
- Appel
  - Retourne 0 au fils
  - Retourne le PID du fils au père, ou -1

`void exit(int status)`

Terminaison d'un processus

- Provoque la terminaison du processus
- Code de retour : `status`
  - O : fin correcte
  - autre : fin incorrecte
  - Récupération avec `$?` pour le shell
- Rattachement de tous les fils encore actifs à `init`
- Libération des ressources allouées
- Fermeture des fichiers
- Attente du père : réveil de celui-ci

`pid_t wait(int *status)`

Attente de terminaison

- Suspend le processus jusqu'à la terminaison de l'un de ses fils
  - Possibilité pour le fils d'être zombie
- Retourne le pid du fils, ou -1 (dans le cas ou il n'y a pas de fils)
- Récupère les zombies en premier
- Achèvement du père : fils pris en charge par `init`
- `status` : valeur de `exit` ou autre (dans le cas d'un signal)

Exemple

Exemple

voir fichier `exemple_fork.c`

Fonctions de recouvrement (exec)

exec

- Permet de recouvrir la zone de texte (programme) par une autre
- Pas de création d'un nouveau processus