Création de processus

dimanche 25 décembre 2022

1. GitHub Code

2. Définition du processus : un processus est entité dynamique

   -1- Commande *pstree* : afficher les processus (arbre de prosessus)

   -2- Deux ensembles à concernés

      1- Identification/Identité : PID, PPID

      2- Contexte : Segment de Code, celui-ci de Stack, Queue

3. ***pid_t fork(void)*** : opération de création

   -1- Si retour du $pid = 0$, le processus fils

   -2- Si retour du $pid > 0$, le processus père

   -3- Il est rappelé par une fois et retourne par deux fois (fils 0 et père)

   -4- Deux raisons possibles pour **une erreur de fork** :

      1- Le nombre actuel de processus a atteint la limite supérieure spécifiée par le système, alors la valeur de *errno* est définie sur EAGAIN.

      2- La mémoire système est insuffisante et la valeur de *errno* est définie sur ENOMEM à ce moment.

   -5- On ne sait pas d'ordre fixe pour l'exécution des deux processus, qui dépend de la politique de **planification des processus du système**. Donc, on utilise souvent *wait*() en évitant l'incertitude.

4. ***exec***() : opération de recouvrement 覆盖操作

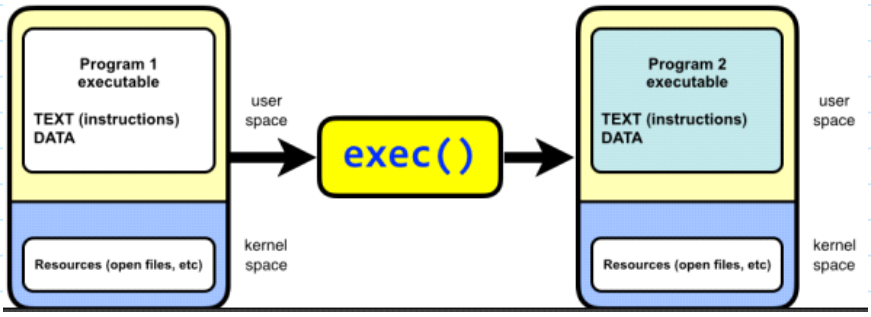
   -1-
   - int execl(const char *path, const char *argv, ...);
   - int execv(const char *path, const char *argv[]);
   - int execle(const char *path, const char *argv, const char *envp[]);
   - int execlp(const char *file, const char *argv, ...);
   - int execvp(const char *file, const char *argv[]);

      1- $int\ execv("/bin/ls", "-l", NULL);$

   -2- Un processus fils peut remplacer complètement (à partir du début de main) le code de père par un nouveau processus (mais **il ne crée pas de processus**)

   -3- Après l'appel de *exec*(), le père s'arrête et s'enregitre, et PID, PPID du fils sont équales à celui-ci du père.

   -4- Si le processus fils est créé pour appeler *exec*() afin d'exécuter un nouveau programme, $vfork()$ doit être utilisé

-5- 

5. ***int system(const char * string)*** *: lancer un nouv*eau processus/commande

-1- Il retourne jusqu'à la fin de ce processus (possibilité en bloquant)

6. ***int waitpid(pid_t pid, int* status, int options)***

-1- Options

1- WNOHANG : attente non bloquante si pas de fils

2- WUNTRACED : attente terminaison ou processus stoppé

3- WCONTINUED : attente réception SIGCONT par le fils

4- $waitpid(pid\_fils, \&st\_fils, WUNTRACED \mid WCONTINUED)$

7. Terminaison d'un processus

-1- Introduction

1- Interaction交互 des processus

1> Ouverture, modification

2> Envoi de messages

2- Terminaison inopinée意外的

1> Terminaison initiée par l'utilisateur du programme

2> Terminaison par une commande *Shell* $(kill)$

3> Terminaison par le système

3- Terminaison propre

1> Gestion du changement d'état des ressources

2> Garde la cohérence协调 des fichiers

4- Veille de père sur son fils

1> Ne pas laisser de processus fils à l'état zombi

2> Sinon encombrement堵塞 des tables du système

-2- Caractéristiques

1- $int\ atexit\left(void\ (*function)(void)\right)$ *: installer* une fonction qui sera exécutée sur

terminaison normale

1> Il appel cette fonction, lorsque le processus se termine normalement.

2> Si $atexit()$ est appelé plus d'une fois, ils seront exécutées **par la pile** des $atexit()$.

-3- Processus zombis

1- Pour éviter l'état zombi, le délais de père soit supérieur à celui-ci de fils.

-4- Code de retour

1- $sigaction()$: Détection de la fin d'un fils et après exécute la fonction de handler ⇒ Récupération de l'état d'un processus fils

1> $SIGCHLD$: Signal de l'intercepter拦截信号

2> $void\ intercepter()\ \{printf("Hello\backslash n");\}$

3> $struct\ sigaction\ S;$

4> $S.sa\_handler = intercepter;$

5> $sigaction(SIGCHLD, \&S, NULL)$

8. Pipes : communication inter-processus

-1- Mécanismes de communication entre processus

1- $pipe()$: anonymes匿名

1> Permettant la communication entre deux processus ayant **un ancêtre commun**

2- $mkfifo()$: nommés

1> Permettant la communication entre **n'importe quels processus** en passant par le système de **fichier** en mode FIFO.

2> Lorsque des processus s'échangent des données à travers le pipe nommé, le noyau fait passer les données en interne sans écriture sur le système de fichier

3- IPC Système V

4- Les sockets : (non abordés dans ce cours)

-2- Caractéristiques

1- En mode FIFO, la communication est unidirectionnelle

2- En mode flot continu d'octets

3- Un synchronisation de type producteur/consommateur entre lecteurs et rédacteurs :

1> un lecteur peut parfois attendre qu'il y est quelque chose d'écrite avant de lire

2> un écrivain peut attendre qu'il y ait de la place dans le tube avant de pouvoir y écrire

-3- Procédure de fonctionnement

1- $int\ pipe(int\ fd[2])$ crée un pipe

1> S'il réussit, il renvoie 0, sinon -1.

2- $fork()$ processus créateur crée un processus fils

3- Le père et le fils choisissent le sens de communication dans le pipe commun

4- Ils communiquent par $read(p[0],\ char*, int\ n)$ et $write(p[1], char*, int\ n)$ sur le pipe commun, mêmes appels systèmes que sur les fichiers

-4- Héritage de données (pipe)

9. Facilite la mise en place d'une communication ente père et fils

10.