Programmation système avancée

Mémoire partagée anonyme

mémoire partagée anonyme

mémoire partagée anonyme

Deux ou plusieurs processus partagent une région de la mémoire créée avec `mmap()` `sans aucun fichier à partager.`

Utiliser mmap() avec les paramètres:

fd == -1 la mémoire partagée n'est pas associée à un fichier. Plus besoin de faire open() pour ouvrir le fichier.

`flag == MAP_ANON | MAP_SHARED`

`MAP_ANON <--> MAP_ANONYMOUS en fonction de système`

`ou  flag == MAP_ANON | MAP_PRIVATE`

Mémoire partagée créée par un processus est partagée après fork() par tous les processus descendant (mais seulement jusqu'à exec())

```
adr = mmap(0, len, PROT_READ|PROT_WRITE,
           MAP_SHARED|MAP_ANON, -1, 0);
```

mémoire partagée anonyme - exemple de Michael Kierrisk

```
#define _BSD_SOURCE
int
main(int argc, char *argv[]){
    int *addr;                  /* Pointer to shared memory region */
    /* Parent creates mapped region prior to calling fork() */
    addr = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED) PANIC_EXIT("mmap");
    *addr = 1;                  /* Initialize integer in mapped region */
    switch (fork()) {           /* Parent and child share mapping */
    case -1:  PANIC_EXIT("fork");
    case 0:                     /* Child: increment shared integer and exit */
        printf("Child started, value = %d\n", *addr);
        (*addr)++;
        if (munmap(addr, sizeof(int)) == -1)
            PANIC_EXIT("munmap");
        exit(EXIT_SUCCESS);
    default:                    /* Parent: wait for child to terminate */
        if (wait(NULL) == -1)
            PANIC_EXIT("wait");
        printf("In parent, value = %d\n", *addr);
        if (munmap(addr, sizeof(int)) == -1)
            PANIC_EXIT("munmap");
        exit(EXIT_SUCCESS);
```

l'enfant augmente la valeur int dans la mémoire partagée anonyme, parent attend la terminaison de l'enfant et lit la valeur écrite par l'enfant

mémoire partagée anonyme - exemple de Michael Kierrisk

```
/****************** panic.h ********************/
#ifndef PANIC_H
#define PANIC_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define PANIC_EXIT( msg )  do{        \
   fprintf(stderr,\
     "\n %s : error \"%s\" in file %s in line %d\n",\
       msg, strerror(errno), __FILE__, __LINE__); \
   exit(1);\
  } while(0)
#endif
Rappel : les macro-fonctions doivent être écrite sur une seule ligne,
pour continuer la ligne taper contre-oblique suivi de RETURN
```

mémoire partagée

Problème de synchronisation de l'accès à la mémoire partagée.

Si deux processus accèdent à la mémoire partagée en même temps et au moins un d'entre eux modifie le contenu de la mémoire partagée le résultat est indéfini. Conclusion : nécessité de synchroniser les accès à la mémoire partagée.

Une technique simple avec le verrou de fichier. Dans cette méthode le fichier sert juste à coordonner l'accès à la mémoire partagée, le fichier ne contient aucune information, c'est un fichier vide (et ce n'est pas le fichier projeté en mémoire).

Créer un fichier vide connu par tous les processus communiquant par la mémoire partagée.

- Avant d'écrire dans la mémoire partagée le processus demande le verrou fcntl (POSIX) exclusif sur le premier octet du fichier.
- Après écriture le processus libère le verrou
- Avant de lire dans la mémoire partagée le processus pose le verrou partagé fcntl sur le premier octet de fichier.
- Après la lecture de la mémoire partagée le processus libère le verrou.