Session 3

Gestion des threads

Plan

- La gestion des threads
- La communication par les segments de mémoire partagés

Processus Unix

- Un processus
  - un espace d'adressage et
    - Le programme
    - Les données
    - La pile d'exécution
  - une unité d'exécution unique
    - Actions produites par l'exécution du processus
    - Ressources physiques mises en œuvre
- Contexte associé à un processus Unix:
  - pid,
  - répertoire de travail,
  - descripteurs,
  - propriétaires,
  - implantation en mémoire,
  - registres,
  - priorité d'ordonnancement,
  - masque des signaux,
  - pile d'exécution
- Cela rend coûteuse la création d'un processus

Descripteur des processus Unix

- Bloc de contrôle (utilisé par Linux) comporte :
  - État du processus
  - Priorité du processus
  - Signaux en attente
  - Signaux masqués
  - Pointeur sur le processus suivant dans la liste des processus prêts
  - Numéro du processus
  - Numéro du groupe contenant le processus
  - Pointeur sur le processus père
  - Numéro de la session contenant le processus
  - Identificateur de l'utilisateur réel
  - Identificateur de l'utilisateur effectif
  - Politique d'ordonnancement utilisé
  - Temps processeur consommé en mode noyau et en mode utilisateur
  - Date de création

Thread

- Processus léger
- Permet de multiplier l'implémentation d'activités caractérisées par des contextes plus légers que ceux des processus

- Contexte d'exécution d'une thread Unix:
  - registres,
  - priorité d'ordonnancement,
  - masque des signaux,
  - pile d'exécution.

- L'espace virtuel d'une thread :
  - code exécuté par la thread,
  - données définies dans la thread et
  - une pile d'exécution propre à la thread.

Entre Processus et Thread

- Un processus
  - C'est l'exécution d'un programme par le processeur. Un même programme (passif) peut donner lieu à plusieurs processus (actifs).
  - Il se compose de l'environnement (variables héritées), le tas (variables globales), la pile (variables et exécution locales) et le code (avec le compteur d'instruction).
  - On peut avoir plusieurs processus actifs qui sont dans l'un des états suivants: prêt, en attente, en traitement. On ne peut avoir qu'un seul processus en traitement à la fois.

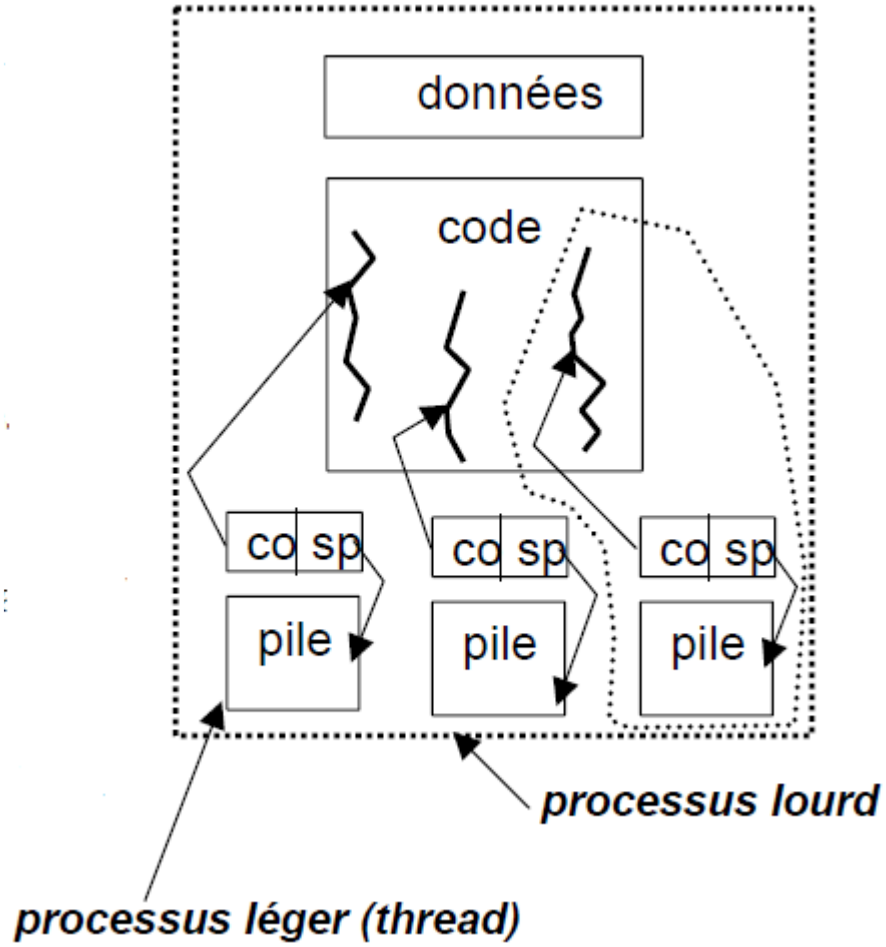
- Un thread
  - C'est une abstraction de l'exécution.
  - Chaque thread a son propre compteur de programme qui maintient l'historique des instructions.
  - Plusieurs threads partagent le même espace mémoire.

Processus Léger ou Thread

- Partage les zones de code, de données, de tas + des zones du PCB (Process Control Block) :
  - liste des fichiers ouverts, comptabilisation, répertoire de travail, userid et groupid, des handlers de signaux

- Chaque thread possède :
  - un mini-PCB (son CO + quelques autres registres),
  - sa pile,
  - attributs d'ordonnancement (priorité, état, etc.)
  - structures pour le traitement des signaux (masque et signaux pendants)

Thread et Processus



Caractéristiques des Threads

- Avantages
  - Création plus rapide
  - Partage des ressources
  - Communication entre les threads est plus simple que celle entre processus
    - communication via la mémoire : variables globales.
  - Solution élégante pour les applications client/serveur :
    - une thread de connexion + une thread par requête

- Inconvénients
  - Programmation plus difficile (mutex, interblocages)
  - Fonctions de librairie non multi-thread-safe

Primitives de manipulation de threads Unix

- Création et activation d'une thread:
  - int pthread_create(
  - pthread_t *idptr, /* ptr sur la zone où sera retournée l'identité de la thread */
  - pthread_attr_t attr,/* attributs de la thread: pthread_attr_default ou 0 */
  - void *(*fonc) (void *), /* pointeur sur la fonction exécutée par la thread */
  - void *arg/* pointeur sur le(s) argument(s) passé(s) à la thread */
  -);
- Terminaison d'une thread:
  - void pthread_exit(
  - void *status /* ptr sur le résultat retourné par la thread */
  -);
- Libération des ressources d'une thread:
  - int pthread_detach(
  - pthread_t *idptr/* pointeur sur l'identité de la thread */
  -);
- Attente de la terminaison d'une thread:
  - int pthread_join(
  - pthread_t id, /* identité de la thread attendue */
  - void *status/* pointeur sur le code retour de la thread attendue */
  -);

Exemple 1 de threads Unix

```
/* Trois_Th.c */
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int compteur[3];
/* fonction executee par chaque
thread */
void *fonc_thread(void *k) {
   printf("Thread numero %d :
mon tid est %d\n",(int) k,
pthread_self());
   for(;;) compteur[(int) k]++;
}
main() {
int i, num; pthread_t tid[3];
 /* creation des threads */
for(num=0;num<3;num++)
{pthread_create(tid+num, 0,
fonc_thread,(void *) num);
 printf("Main: thread numero %d
creee: id = %d\n",num,
tid[num]);
  }
 usleep(10000); /* attente de
10 ms */
printf("Affichage des
compteurs\n");
for(i=0;i<20; i++) {
  printf("%d \t%d \t%d\n",
compteur[0], compteur[1],
compteur[2]);
  usleep(1000);
/* attente de 1 ms entre 2
affichages */
 }
exit(0); }
```

7

Exemple 2 de threads Unix

```
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
typedef struct {
                  int x, y;
                  } data;
/* fonction executee par  chaque thread
*/
void *mul(data *ptrdata)
{
pthread_exit((void *)((ptrdata->x) *
(ptrdata->y)));
}
void main(int argc, char *argv[]){
int i;
int a, b, c, d;
pthread_t pth_id[2];
data donnees1;
data donnees2;
int res1, res2;
if(argc < 5) { perror("\007Nbre
d'arguments incorrect"); exit(1);}
a=atoi(argv[1]); b=atoi(argv[2]);
c=atoi(argv[3]); d=atoi(argv[4]);
donnees1.x=a; donnees1.y=b;
donnees2.x=c; donnees2.y=d;
/* creation des threads */
   pthread_create(pth_id,   0, (void
*(*)())mul, &donnees1);
   pthread_create(pth_id+1, 0, (void
*(*)())mul, &donnees2);
/* Attente de la fin de la thread 1 */
   pthread_join(pth_id[0], (void **)
&res1);
/* Attente de la fin de la thread 2 */
   pthread_join(pth_id[1], (void **)
&res2);
/* Affichage du resultat a*b + c*d */
   printf("Resultat =%d\n", res1+res2);
/* Suppression des ressources des
threads */
   pthread_detach(&pth_id[0]);
   pthread_detach(&pth_id[1]);
exit(0);
}
```

Thread

compilation avec :

```
gcc thread_exo1.c -o exo1 -lpthread
```

Cancel

- Moyen donné à une thread de tenter d'arrêter l'exécution d'une autre thread
- L'autre thread peut accepter de subir ce cancel ou le refuser en le masquant

Exercices

- Ecrire un programme qui crée trois threads qui exécutent toutes le même code qui consiste à incrémenter un compteur particulier

- La thread initiale se contentant de visualiser périodiquement les valeurs des différents compteurs