COMMUNICATION ET SYNCHRONISATION ENTRE PROCESSUS UNIX

Michel Riveill – riveill@unice.fr

Le menu du jour

- La vie des processus Unix → rappel
  - Clonage
  - Création par remplacement
- Communication entre processus Unix
  - Les tubes (pipes) → rappel
  - Les tubes nommés
  - Les fichiers couplé en mémoire → cf. TD
  - Les sockets → pas dans ce cours
  - La mémoire partagée
  - Les files de messages
- Synchronisation entre processus UNIX
  - Les signaux → rappel
  - Les verrous
  - Les sémaphores

3

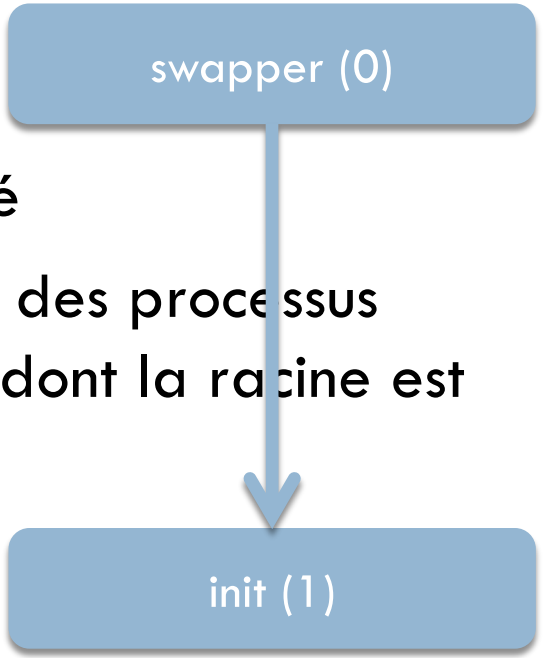
La vie des processus Unix

Clonnage

Remplacement

La vie des processus Unix

- Lors de sa création, tout processus reçoit un numéro unique (entier positif)qui est son identificateur (pid).
- Tout processus est créé par un autre processus, excepté le processus initial, de nom swapper et de pid 0, créé artificiellement au chargement du système
- Le swapper crée alors un processus appelé init, de pid 1, qui initialise le temps-partagé
- Par convention, on considère que l'ensemble des processus existants à un instant donné forme un arbre dont la racine est le processus initial init.



La vie des processus

- Un processus qui s'exécute lâche le processeur de manière
  - Volontaire → multi-tâche coopératif
    - L'ordonnancement dépend de l'application
    - Possibilité de monopoliser le système
  - Forcé → multi-tâche préemptif
    - Protection de l'OS
    - Permet d'assurer un équilibre entre les processus (et donc les utilisateurs)
- Typiquement, un processus est interrompu (préempté)
  - Après un certain temps (time slice)
  - En cas de terminaison d'une entrée/sortie
  - Si un autre processus à une plus haute priorité

La vie des processus – changement de contexte

- Le basculement d'un processus à un autre est géré par le noyau
  - Suspendre le processus P0
  - Sauvegarder le contexte du processus P0
  - Restaurer le contexte du processus P1
  - Reconfigurer l'espace mémoire
    - En particulier reconfiguration du MMU (Memory Management Unit)
    - Rappel : *le MMU gère la correspondance adresse virtuelle / adresse physique*
  - Démarrer le processus P1
- Cette opération est généralement décidée et réalisée par l'ordonnanceur (scheduleur)
  - Opération coûteuse

15

Communication entre processus

Tubes (pipe)

Tubes nommés

Fichiers couplés

Sockets

Mémoire partagée

File de messages

Communication entre processus – mmap ()

- L'appel système mmap () permet de projeter le contenu d'un fichier en mémoire
  - Le contenu de la mémoire est synchronisé automatiquement avec le contenu du fichier (et vice-versa).
    - L'option MAP_SHARED est requise pour garantir la synchronisation.
  - Un fichier couplé en mémoire (mappé) peut être partagé par plusieurs processus
    - Addr = mmap (NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fr, 0)
    - NULL : le fichier est coupler à une adresse choisie par le système
    - size : taille de la zone mémoire
    - La zone mémoire peut être lue et écrite (autre protection possible : exécutée)
    - Les modifications sont partagées
    - Descripteur du fichier utilisé
    - Offset

Communication entre processus – IPC

- Mécanismes IPC (Inter Process Communication) regroupent
  - Mémoire partagée
  - File de messages
  - Sémaphore
- Tous ces mécanismes survivent au processus qui l'a créés
  - Il faut donc les détruire explicitement
- Pour tous ces mécanismes
  - Les droits d'accès sont définis à la création
  - La commande ipcs liste les ressources IPC allouées
  - La commande ipcrm pemet de libérer des ressources

Communication entre processus – les clés IPC

- Tous les mécanismes IPC utilisent une clé permettant d'identifier la famille de processus pouvant utiliser le même segment mémoire partagé, le même sémaphore ou la même file de message
- Les procédures suivantes permettent de trouver une clé dynamiquement
  - key_t ftok (char *pathname, char project);
  - Exemple
    - cle =ftok ("/", 'A'); // clé absolue
    - cle =ftok (".", 'A'); // clé dépendant du répertoire du processus
- Autres solutions : définir statiquement la clé
  - Exemple
    - cle = 123

Communication entre processus – mémoire partagée

- Permet à plusieurs processus processus tout à fait quelconques (pas nécessairement affilié) de partager des segments en mémoire. Il s'agit d'un partage de mémoire qui n'induit aucune recopie de données …
- Les segments mémoires peuvent être couplés à des adresses différentes
- Principales opérations
  - shmget () : création
  - shmat () : couplage du segment mémoire dans l'espace d'adressage virtuel du processus
  - shmctl () : modification des droits et du propriétaire, destruction
  - shmdt () : détachement du segment mémoire

Mémoire partagée – création (1ère étape)

- #include <sys/shm.h>
- Création d'une mémoire partagée
  - shmid = shmget (key t key, int size, int shmflg);
- shmget retourne l'identificateur (int shmid) du segment de mémoire partagée ayant la clé key.
- Un nouveau segment de taille size octets est créé si :
  - Indicateur de shmflg contient IPC_CREAT;
  - Par exemple IPC_CREAT|IPC_EXCL indiquent que le segment ne doit pas exister au préalable.
- Les bits de poids faible de shmflg indiquent les droits d'accès (rwxrwxrwx).
- Exemple :
  - id = shmget (cle, sizeof(int), IPC_CREAT|0666);

Mémoire partagée – liaison/couplage (2de étape)

- Lier un segment à un processus lui permet l'accès aux données contenues dans ce segment à l'aide d'un pointeur :
  - mem addr = shmat (int shmid, char *shmaddr, int shmflg);
- retourne l'adresse (char *mem addr) où le segment identifié par (shmid) a été placé en mémoire :
  - Placement automatique (et conseillé) si shmaddr = NULL
  - Si shmaddr != NULL, le segment est couplé à l'adresse indiqué (si c'est possible)
- shmflg spécifie quels sont les droits d'accès du processus au segment : SHM_R, SHM_W, ...
- Exemple :
  - addr = shmat (id, 0, 0);

Mémoire partagée – déliaison/découplage

- Délier un segment de mémoire partagée d'un processus
  - ret = shmdt (char *mem addr);
- Détache le segment du processus et retourne (-1) en cas d'erreur (0 sinon).
- Exemple :
  - If (shmdt (addr) == -1) {
    fprintf (stderr ,"segment indétachable\n"); exit(-1);
    }

Mémoire partagée - contrôle

- int shmctl (int shmid, int cmd, struct asmid ds *buf);

- permet diverses opérations
  - Destruction du segment (IPC_RMID)
    - A priori : le segment est détruit quand plus aucun processus ne le lie (ce n'est pas toujours le cas)
  - Verrouillage en mémoire (SHM_LOCK)
    - Le segment n'est plus swappé

- Exemples :
  - shmctl (shmid, SHM_LOCK, NULL); // verrouille mémoire partagées
  - shmctl (shmid, IPC_RMID, NULL); // détruit une mémoire partagée
  - If (shmctl (id, IPC_RMID, NULL)==-1) {
    fprintf (stderr, "segment indestructible\n"); exit(-1);
    }

Un exemple d'utilisation : producteur-consommateur

- Un processus producteur lit une valeur au clavier puis l'incrémente à la valeur d'une variable commune
  - Si la valeur lut au clavier est 1, le producteur s'arrête
- Un processus consommateur lit la valeur de la variable commune puis l'affiche
- Il n'y a pas de synchronisation…

Le producteur


```
void abandon(char message[]) { perror(message); exit(EXIT_FAILURE); }
int main(void) {
    key_t cle;
    int id; *seg_part; reponse;
    if (cle = ftok (getenv("HOME"), 'A') == -1) abandon("ftok");
    if (id = shmget (cle, sizeof(int), IPC_CREAT | IPC_EXCL | 0666) == -1)
            if (errno == EEXIST) abandon ("Note: le segment existe déjà\n")
            else abandon ("shmget");
    if (seg_part = (int *) shmat(id, NULL, SHM_R | SHM_W)  == NULL) abandon ("shmat");
    *seg_part = 0;
    while (scanf("%d", &reponse) != 1) *seg_part += reponse;
    if (shmdt ((char *) seg_part) == -1) abandon ("shmdt");
    if (shmctl (id, IPC_RMID, NULL) == -1) abandon ("shmctl(remove)");
    return EXIT_SUCCESS;
}
```

Le consommateur


```
void abandon(char message[]) { perror(message); exit(EXIT_FAILURE); }
int main(void) {
    key_t cle;
    int id, *commun;
    struct sigaction a;
    if (cle = ftok (getenv("HOME"), 'A') == -1) abandon("ftok");
    if (id = shmget (cle, sizeof(struct donnees), 0) == -1)
            if (errno == ENOENT) abandon ("pas de segment\n")
            else abandon ("shmget");
    if (commun = (int *) shmat (id, NULL, SHM_R) == NULL) abandon("shmat");
    while (*commun < 1000) { sleep(2); printf("sous-total %d\n", *commun;}
    if (shmdt((char *) commun) == -1) abandon("shmdt");
    return EXIT_SUCCESS;
}
```

Synchronisation entre processus

Signaux

Verrous

Sémaphore

Files de messages

Synchronisation entre processus Unix – verrouillage de fichier

- La commande système flock () permet de mettre des verrous partagés ou exclusifs sur des fichiers
  - Verrou partagé (**LOCK_SH**) : autorise l'accès simultanée à plusieurs processus
  - Verrou exclusif (**LOCK_EX**) : un seul accès simultanée
  - Libération d'un verrou précédemment acquis (**LOCK_UN**)
  - On peut passer d'un verrou partagé à un verrou exclusif
  - Les verrous portent sur le fichier (pas son descripteur)
    - Si on duplique le descripteur, le verrou concerne toujours le même fichier
- Exemple d'utilisation

```
fp = fopen ("/tmp/lock.txt", "w+");
if (flock (fp, LOCK_EX)) {    // pose un verrou exclusif
    fwrite (fp, "Écrire dans un fichier\n");
    flock (fp, LOCK_UN);                    // libère le verrou
} else {  printf ( "Impossible de verrouiller le fichier !\n";  }
fclose (fp);
```

Synchronisation entre processus - sémaphore

- Fait parti des mécanismes IPC (Inter Process Communication)
- Permet de résoudre le problème des accès concurrents à une même ressource telle que, par exemple, un segment de mémoire partagé entre plusieurs processus
- Les sémaphores IPC sont gérés sous forme d'un tableau, on effectue les opérations équivalente à P () et V() sur les éléments du tableau
  - Création du tableau : semget ()
  - Manipulation du tableau : semctl ()
  - Opération Down () et Up () : réservation ou libération de N unité de ressources

Utilisation des sémaphores Unix à la Dijsktra


```
typedef int semaphore;
void abandon(char message[]) { perror(message); exit(EXIT_FAILURE); }
semaphore creer_sem (key_t key, int val_init) {
    /* création d'un tableau de 1 sémaphore initialisé à val_init */
    semaphore sem;
    int r;
    if (sem = semget (key, 1, IPC_CREAT | 0666) < 0) abandon ("creer_sem");

    if (r = semctl (sem, 0, SETVAL, val_init) < 0)
            abandon ("initialisation sémaphore");
    return sem;
}
void detruire_sem(semaphore sem) { if (semctl (sem, 0, IPC_RMID, 0) != 0)
    abandon("detruire_sem"); }
```

Utilisation des sémaphores Unix à la Dijsktra


```
void changer_sem(semaphore sem, int val) {
    struct sembuf sb[1];
    sb[0].sem_num = 0;
    sb[0].sem_op = val;
    sb[0].sem_flg = 0;
    if (semop (sem, sb, 1) != 0) abandon("changer_sem");
}
void down(semaphore sem) { changer_sem(sem, -1); }
void up(semaphore sem) { changer_sem(sem, 1); }
```

Contrôle d'une section critique avec des sémaphores


```
// mutex
// → sémaphore
   initialisé à 1

down (mutex)
/* je suis
   * en
   * section
   * critique
   */
up (mutex)
```

```
semaphore sem;
key_t cle;

if (cle = ftok(getenv("HOME"), 'A')
                    == -1)
    abandon("ftok");
sem = creer_sem (cle, 1);
down (sem);
/* je suis en section critique */
up (sem);
detruire_sem (sem);
```

Mise en place d'une barrière avec des sémaphores

- Chaque processus i se bloque sur un sémaphore attendre initialisé à 0
- Le nombre de processus à attendre est N
- Pour protéger l'utilisation de la variable n initialisée à N qui compte le nombre de processus arrivée à la barrière, on utilise un sémaphore mutex

```
down(mutex)
n = n-1
Si (n > 0)
Alors
   up (mutex)
   down(attendre)
   down (mutex)
Finsi
n = n+1
Si (n < N)
   up(attendre)
Finsi
up(mutex)
```

TD : ‘autour du parking’

Utilisation :

- Mémoire partagée
- Sémaphores