Creación de procesos en UNIX

Introducción

fork - wait

Introducción

- fork
  - Permite subdividir el proceso en dos procesos idénticos con la diferencia del valor de retorno de la función.
- wait
  - Permite sincronizar el proceso padre y el proceso hijo.
- Manejo básico de señales (signals)
  - Mecanismo para manejo de excepciones.

2

headers

Se debe incluir

```
#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>
```

3

fork()

Crear un proceso hijo

pid_t fork (void);

Resultado:

- Identificador del proceso hijo para el proceso padre
- 0 para el proceso hijo
- -1 si hubo error, dónde errno especifica el tipo de error

4

fork()

```
childpid = fork();
if (childpid == -1)
   {
    perror("fork");
    exit(1);
   }
else if (childpid == 0)
   {
    /* PROCESO HIJO */
    ...
    exit(0);
   }
else
   {
    /* PROCESO PADRE */
    ...
   }
```

5

wait()

Suspender hasta la muerte de un proceso hijo

pid_t wait(int * stat_loc);

- stat_loc: estado con que terminó el proceso hijo

Resultado:

- identificador del proceso hijo
- -1 si hubo error, dónde errno especifica el tipo de error

6

waitpid()

Suspender hasta la muerte de un proceso hijo

pid_t waitpid(pid_t pid, int * stat_loc, int * options);

- pid: identificador del proceso hijo por el que espero (-1 si espero cualquiera)
- stat_loc: estado con que terminó el proceso hijo
- options: bandera de opciones
  - WNOHANG: no se suspende la ejecución del proceso si no hay información de un proceso hijo de forma inmediata.

Resultado:

- identificador del proceso hijo
- -1 si hubo error, dónde errno especifica el tipo de error

7

SIGCHLD

§ SIGCHLD es una señal (signal) que el sistema operativo envía a los procesos cuando uno de sus hijos muere.

§ Cuando trabajamos con un esquema fork – wait, debemos capturar esta señal, para que el proceso padre no cancele.

§ Para esto o definimos un handler asociado a la señal, o declaramos de forma explícita que debe ignorarse.

8

SICHLD

Definición de handler

```
...
int catchChild(int sig_num)
{
 /* cuando estamos aquí, sabemos que hay un proceso hijo
    “zombie” esperando */

 int      childStatus;

 wait(&childStatus);
 printf(“Hijo terminado.\n”);
}
...
/* Define el handler de la señal para la señal SIGCHLD */
signal(SIGCHLD, catchChild);
...
```

9

SICHLD

Declaración de ignorar la señal

```
...

/* Define explícitamente ignorar la señal SIGCHLD

  */

sigignore(SIGCHLD);

...
```

10

System V IPC

Introducción

Memoria Compartida

Introducción

§ Objetos IPC
- Colas de mensajes
- Semáforos
- Memoria compartida

§ Identificadores IPC

§ Claves IPC
- key_t ftok(char *pathname, char proj);

```
key_t mykey;
mykey = ftok(".", 'a');
```

12

Comandos Generales

§ ipcs

- ipcs -q: muestra sólo colas de mensajes
- ipcs -s: muestra sólo semáforos
- ipcs -m: muestra sólo memoria compartida
- ipcs --help: argumentos adicionales

§ ipcrm

- ipcrm <msg | sem | shm> <IPC ID>

13

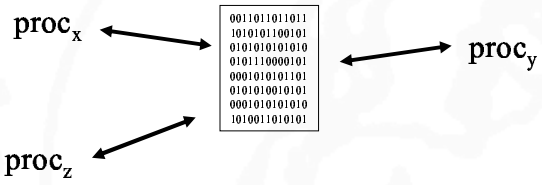
Comandos Generales

```
------ Shared Memory Segments --------
shmid      owner      perms      bytes      nattch     status
------ Semaphore Arrays --------
semid      owner      perms      nsems      status
------ Message Queues --------
msqid      owner      perms      used-bytes      messages
0          root       660        5               1
```

14

Memoria Compartida

§ La memoria compartida puede describirse como el mapeo de un área o segmento de memoria que puede ser direccionado y compartido por más de un proceso.



15

headers

Se debe incluir

```
#include <sys/ipc.h>

#include <sys/shm.h>
```

16

shmget()

Obtener el identificador del segmento de memoria compartida

int shmget (key_t key, int size, int shmflg);

- key : resultado de invocación a ftok()
- size : tamaño del segmento solicitado.
- shmflg: bandera de opciones
  - IPC_CREAT : crea el segmento si no existe aún en el kernel.
  - IPC_EXCL : cuando se utiliza con IPC CREAT, falla si el segmento ya existe.
  - 0ddd: especifica los permisos a asignar sobre el segmento en octal.

Resultado:

- Identificador de segmento de memoria compartida si hubo éxito
- -1 si hubo error, dónde errno especifica el tipo de error

17

shmget()

```
int abrir_segmento(key_t keyval, int segsize)
{
 int        shmid;
 shmid = shmget(keyval, segsize, IPC_CREAT |
  0660);
 if (shmid == -1)
    {
     return(-1);
    }
 return(shmid);
}
```

18

shmat()

Obtener un puntero al segmento de memoria compartida

void * shmat (int shmid, void *shmaddr, int shmflg);

- shmid: identificador resultado de shmget()
- shmaddr : cero para que el sistema operativo hubique libremente el segmento
- shmflg: bandera de opciones
  - SHM_RDONLY : para indicar sólo lectura.

Resultado:

- dirección del segmento de memoria compartida si hubo éxito
- -1 si hubo error, dónde errno especifica el tipo de error

19

shmat()

```
void * mapear_segmento(int shmid)
{
 return(shmat(shmid, 0, 0));
}
```

20

shmctl()

Modificar / destruir el segmento de memoria compartida

int shmctl (int shmid, int cmd, struct shmid_ds *buf);

- shmid: identificador resultado de shmget()
- cmd: bandera de opciones
  - IPC_STAT: carga la estructura shmid_ds asociada al segmento en buf.
  - IPC_SET: define los permisos del segmento tomando los valores de buf.
  - IPC_RMID: marca el segmento para borrado.
  - SHM_LOCK : bloquea el segmento. (solo root)
  - SHM_UNLOCK: desbloquea el segmento. (solo root)
- buf: estructura con la información del segmento

Resultado:

- 0 si hubo éxito
- -1 si hubo error, dónde errno especifica el tipo de error

21

shmctl()

```
int borrar_segmento(int shmid)
{
 return(shmctl(shmid, IPC_RMID, NULL));
}
```

22

shmdt()

Liberar el segmento de memoria compartida

int shmdt (void *shmaddr);

- shmaddr : segmento de memoria direccionado

Resultado:

- 0 si hubo éxito
- -1 si hubo error, dónde errno especifica el tipo de error

23

shmdt()

```
int liberar_segmento(void * addr)
{
 return(shmdt(addr));
}
```

24

System V IPC

Introducción

Semáforos

Introducción

§ Cada semáforo toma un valor mayor o igual a cero.

§ Un id de objeto IPC proporciona acceso a un array de semáforos.

§ Podemos leer, incrementar o decrementar el valor de un semáforo.

§ Podemos realizar n operaciones a la vez. Dichas operaciones serán aplicadas si todas ellas tienen éxito.

26

Llamada al sistema semget

Obtener el identificador del recurso (array de semáforos)

int semget (key_t key, int nsem, int semflg);

- key : resultado de invocación a ftok()
- nsem : cantidad de semáforos en el recurso.
- semflg: bandera de opciones
  - IPC_CREAT : usado para crear en nuevo recurso.
  - IPC_EXCL : cuando se utiliza con IPC CREAT, falla si el recurso ya existe.
  - 0ddd: especifica los permisos a asignar sobre el recurso en octal.

Resultado:

- Identificador de recurso si hubo éxito.
- -1 si hubo error, dónde errno especifica el tipo de error.

27

Ejemplo semget

Creando un recurso con 10 semáforos

```
#include <sys/ipc.h>
#include <sys/sem.h>

key_t key;
int semid;

key = ftok(“.", 'E');
semid = semget(key, 10, 0666 | IPC_CREAT);
```

28

Llamada al sistema semop

Permite realizar operaciones sobre los semáforos del array

int semop (int id, struct sembuf *sops, unsigned nsops);

- id : identificador del recurso (se obtiene utilizando la llamada semget).
- sops : array de operaciones a realizar sobre el recurso.
- nsops: cantidad de operaciones a realizar (tamaño de sops).

Resultado:

- 0 si fue exitoso (se aplicaron todas las operaciones).
- -1 si hubo error, dónde errno especifica el tipo de error.

29

Estructura sembuf

```
struct sembuf
    ushort  sem_num;        /* semaphore index in array */
    short   sem_op;         /* semaphore operation */
    short   sem_flg;        /* operation flags */
```

§ sem_num: semáforo sobre el cual quiero actuar.

§ sem_op: operación a realizar.
- Positivo: el valor de sem_op se suma al valor del semáforo.
- Negativo: si el valor de sem_op es mayor al valor del semáforo el proceso se bloquea hasta que el valor del semáforo alcance el valor de sem_op. Luego resta sem_op al valor del semáforo.
- Cero: el proceso espera a que el semáforo alcance el valor cero.

§ sem_flg: bandera de opciones
- IPC_NOWAIT (EAGAIN).
- SEM_UNDO: recuerda los cambios realizados sobre el semáforo.

30

Ejemplo semop

Realizando un p

```
struct sembuf sb = {0, -1, 0};
...
result = semop(semid, &sb, 1);
```

Realizando un v

```
struct sembuf sb = {0, 1, 0};
...
result = semop(semid, &sb, 1);
```

31

Llamada al sistema semctl

Modificar / destruir recurso

int semctl (int semid, int semnum, int cmd, union semun arg);

- semid: identificador resultado de semget()
- semnum: semáforo sobre el cual voy a realizar la operación.
- arg: parámetro requerido para realizar la operación indicada.

Resultado:

- 0 si hubo éxito
- -1 si hubo error, dónde errno especifica el tipo de error

32

Llamada al sistema semctl (cont.)

Estructura semun

```
union semun
       int val;              /* used for SETVAL only */
       struct semid_ds *buf; /* for IPC_STAT and IPC_SET */
       ushort *array;        /* used for GETALL and SETALL */
```

33

Llamada al sistema semctl (cont.)

- cmd: operación a realizar.
  - IPC_STAT: carga la estructura semid_ds asociada al recurso en arg.buf
  - IPC_SET: define los permisos del recurso tomando los valores de arg.buf
  - GETPID: retorna el identificador del proceso que realiza la última operación.
  - GETVAL : retorna el valor del semáforo número semnum.
  - GETNCNT: cantidad de procesos esperando que se incremente el semáforo.
  - GETZCNT : cantidad de procesos esperando que el semáforo alcance el valor cero.
  - SETVAL: carga el valor del semáforo con arg.val.
  - GETALL: carga el valor de todos los semáforos en arg.array
  - SETALL: carga el valor de todos los semáforos con arg.array
  - IPC_RMID: elimina el recurso compartido.

34

Ejemplo shmctl

```
union semun temp;
int semid;
...
semid = semget(...);
...
semctl(semid, 0, IPC_RMID, temp);
```

35