Clase 7
Message queues
Shared memory
Semaphores

Autor: Esp. Ing. Ernesto Gigliotti. UTN-FRA

Message queues

- Similar a las MemFIFO
- Los mensajes tienen un tipo
- Se puede quitar de la queue un mensaje intermedio
- Un proceso puede crear una msgqueue o conectarse a una existente

Creación de Message queue

- Dos argumentos
  - **key**: Identificador único en el sistema que describe la queue. Se necesita para conectarse a la queue.
  - **msgflg**: Indica qué se quiere hacer con la queue (crearla por ejemplo)
- Devuelve el ID de la queue o -1

```
int msgget(key_t key, int msgflg);
```

Creación de la key

- Dos argumentos
  - **path**: Un archivo que el proceso pueda leer.
  - **id**: Generalmente se usa un caracter cualquiera (por ejemplo 'A')

- Devuelve la key (es un long)

```
key_t ftok(const char *path, int id);
```

Creación de la queue

- Creamos la message queue

```c
#include <sys/msg.h>

int main(void)
{
   key_t key = ftok("main.c", 'b');
   int msqid = msgget(key, 0666 | IPC_CREAT);

   return 0;
}
```

Creación de la queue

- Ejecutamos comando *ipcs*

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x620546a4 0          ernesto    666        0            0
```

Enviando un mensaje

- Formato del mensaje

```
struct msgbuf {
  long mtype;
  char mtext[20];
};
```

Enviando un mensaje

```
int main(void)
{
   key_t key = ftok("main.c", 'b');
   int msqid = msgget(key, 0666 | IPC_CREAT);

   struct msgbuf msg = {2,{'h','o','l','a','\0'}};
   int size = sizeof(msg.mtext);
   printf("size:%d\r\n",size);
   printf("msg:%s\r\n",msg.mtext);

   msgsnd(msqid,&msg,size,0);

   return 0;
}
```

Mensaje enviado

- Ejecutamos comando *ipcs*

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x620546a4 0          ernesto    666        20           1
```

Recibir mensaje

- Utilizamos msgrcv()

```
int msgrcv(int msqid, void *msgp, size_t msgsz,
                      long msgtyp, int msgflg);
```

msgtyp	Descripcion
Cero	Lee el próximo msg sin importar el tipo
Positivo	Lee el próximo msg del tipo especificado en msgtyp (valor de campo mtype = valor de argumento msgtyp)
Negativo	Lee el próximo msg tal que : su campo mtype <= mod(msgtyp)

Recibir mensaje

```
key_t key = ftok("sender.c", 'b');
int msqid = msgget(key,0);

struct msgbuf msg;
int size = sizeof(msg.mtext);

While(1) {
    printf("Leo proximo mensaje...\r\n");
    if(msgrcv(msqid,&msg,size,0,0)==-1) {
        perror("msgrcv");
        exit(1);
    }
    printf("lei de la queue: %s\r\n",msg.mtext);
}
```

Eliminar queue

- Ejecutamos comando *ipcrm*

  ipcrm -q msqid

- Ejecutamos función *msfctl*

```
msgctl(msqid, IPC_RMID, NULL);
```

Shared memory

- Segmento de memoria compartido entre procesos.
- Se obtiene un puntero a memoria, mediante el que se puede leer y escribir.
- Los cambios serán visibles por otros procesos.

Shared memory

- Creación

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

```
key_t key;
int shmid;

key = ftok("writer.c", 'R');
shmid = shmget(key, 1024, 0644 |
                          IPC_CREAT); //1k segment
```

Shared memory

- Attach

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

- Dejamos que el OS genere la dirección

```
char* pData = shmat(shmid, (void *)0, 0);
if (pData == (char *)(-1))
    perror("shmat");
```

Shared memory

- Leer y escribir.

```
fgets(pData,1024,stdin);

pData[3] = 27;



printf("%s\n", pData);

char a = pData[7];
```

Shared memory

- Detaching y eliminación

```
int shmdt(void *shmaddr);
```

- La eliminación queda pendiente si hay attachs

```
shmdt(pData);

shmctl(shmid, IPC_RMID, NULL);
```

Semaphores

- Creación de un *set* de semaforos

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

key_t key;
int semid;

key = ftok("main.c", 'E');

semid = semget(key, 10, 0666 | IPC_CREAT);
```

Semaphores

- Inicialización:
  - Si el proceso pudo crear el set, le debe dar valores inicialiales.

```
struct sembuf {
   ushort sem_num;
   short sem_op;
   short sem_flg;
};
```

```
struct sembuf sb;

sb.sem_op = 1; // unlock. val:1
sb.sem_flg = 0;
semop(semid, &sb, 1); // 1 op
```

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

Semaphores

- Si el proceso no lo pudo crear (ya estaba creado), debe esperar que se inicialice.

```
for(i = 0; i < MAX_RETRIES && !ready; i++) {

    semctl(semid, nsems-1, IPC_STAT, arg);

    if (arg.buf->sem_otime != 0) {
        ready = 1;
    } else {
        sleep(1);
    }

}
```

Semaphores

- Inicialización

SETVAL
GETVAL
IPC_STAT

```
int semctl(int semid, int semnum,
                      int cmd, ... /*arg*/);

union semun {
   int val; /* used for SETVAL only */

   struct semid_ds *buf; /* used for IPC_STAT and
                                    IPC_SET */
   ushort *array; /* used for GETALL and SETALL */
};
```

Bibliografía

- Uresh Vahalia. (1996). UNIX Internals. The New Frontiers. New Jersey, Prentice Hall.
- Robert Love. (2013). Linux System Programming 2nd Edition. USA, O'Reilly Media.