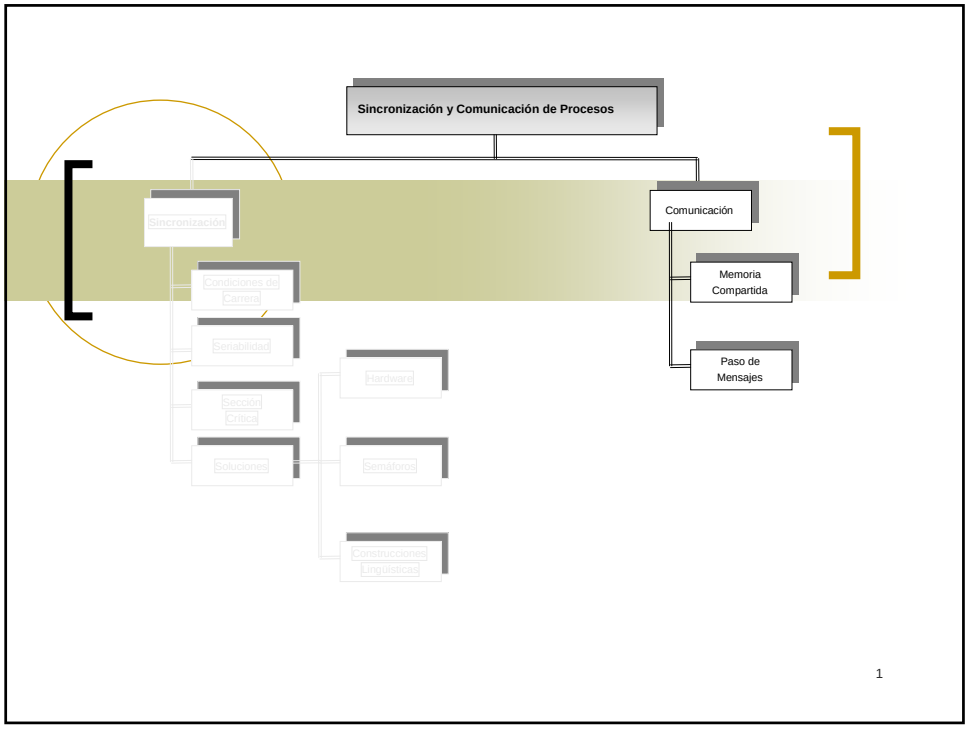


1

Comunicación entre Procesos

- Librería ***IPC*** (*Inter-Process Communication*) /POSIX
- Dos modelos complementarios de comunicación
  - **Memoria compartida**: los procesos comparten variables mediante las cuales comunican la información
    - Existe un espacio de memoria común
    - La responsabilidad de la comunicación recae sobre los programadores
    - El SO sólo proporciona la memoria compartida
  - **Paso de mensajes**: permite a los procesos intercambiar mensajes
    - Los procesos tienen espacios de direcciones distintos
    - Las operaciones básicas de comunicación las proporciona el SO
      send (destino, mensaje)
      receive (origen, mensaje)
    - Es necesario definir las características de los enlaces

Memoria Compartida (POSIX)

- Pertenecen al estándar POSIX.1 y utilizan un entero como descriptor
- Cabeceras:

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

- Funciones:

```
key_t ftok(const char *path, int id);
int shmget (key_t key, size_t size, int shmflg);
void *shmat (int shmid, const void *shmaddr, int shmflg );
int shmdt (const void *shmaddr);
int shmctl (int shmid, int cmd, struct shmid_ds *buf);
```

- Para ver las áreas de memoria compartida activas:

```
ipcs –ma
```


Memoria Compartida: Funciones (I)

- Inicialización: la inicialización crea un segmento de memoria compartida *shmget*
- La creación es similar a la de los semáforos nombrados:
  - Uno de los procesos debe crear el segmento de memoria y el resto lo utilizan

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

- Parámetros:
  - ***key***: identificador único de la región de memoria compartida
  - Para crearlo se utiliza la función:

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

    - ***path***: es el path de un fichero accesible por todos los procesos
    - ***id***: identificador para asociar diferentes claves al mismo fichero.

Memoria Compartida: Funciones (II)

- ***size***: tamaño en bytes de la región de memoria compartida
- ***shmflag****:* determina si se accede a una región de memoria ya creada o se debe crear una nueva
  - Si *shmflag* es 0 se indica que se quiere acceder a una región de memoria compartida ya creada (si no existe, *shmget* devuelve -1)
  - Si *smhflag* tiene el valor de IPC_CREATse crea la región de memoria compartida si no existe. Si ya existe devuelve el manejador a la región ya asignada
  - Los permisos se indican añadiéndolos en octal al parámetro IPC_CREAT
    - *Ejemplo:* IPC_CREAT | 0777

- **Valor Devuelto:**
  - Manejador de la Memoria Compartida (entero positivo)
  - -1 en caso de error

Memoria Compartida: Funciones (III)

- Acceso a la Memoria Compartida (*attach*):

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

- Nos devuelve un puntero a la región de memoria compartida inicializada previamente.
- Parámetros:
  - ***shmid***: identificador devuelto por *shmget()*
  - ***smhaddr***: dos opciones
    - 0: asigna la dirección de memoria automáticamente (Recomendado)
    - Puntero dentro del área de memoria del proceso.
  - ***shmflg***: modo de acceso
    - 0: lectura y escritura
    - SHM_RDONLY: acceso solo en modo lectura

Memoria Compartida: Funciones (IV)

- Liberación de la memoria compartida (*detach*):

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

  - Libera la dirección de memoria (*shmaddr*) asignada previamente. El segmento de memoria compartido con otros procesos no se libera
  - Devuelve 0 si es correcto y -1 en caso de error

- Eliminación de la memoria compartida (último proceso):

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

  - *shmid:* identificador de memoria compartida
  - *cmd:* IPC_RMID
  - *buff:* NULL

  - Devuelve 0 si es correcto y -1 en caso de error

EJEMPLO: memoria compartida

- Crear una región de memoria compartida.
- El Proceso hijo escribe en el buffer de memoria (libera la CPU entre cada escritura
- El proceso padre lee y muestra en contenido del buffer
- Semáforo para acceso al buffer de memoria

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>

#define SHM_SIZE 1024  /* 1K segmento de memoria compartida */

sem_t *sem_1;          // puntero para el identificador de los semáforos
int shmid;             // manejador de memoria compartida

int main(void)
{
   int i,j;
   key_t key;
   pid_t hijo;
   int *data;

   printf("Creando semáforos .....\n");
   /* comprueba si ya existe el semáforo sino lo crea desbloqueado (1)*/
   if((sem_1=sem_open("/sem_ipc1", O_CREAT, 0644, 1))==(sem_t *)-1)
            perror("Error creando semáforo 1");

// Continua ->
```

Valor Inicial 1 (Desbloqueado)

Ejemplo: crear memoria compartida

```
printf("Creando Región de memoria compartida .....\n");

 /* Crea la clave: */
if ((key = ftok("ipc1", 0)) == -1) {
        perror("Error creando clave con ftok");
                                        exit(1);
}

/* Crear/conectar el segmento de memoria: */
 if ((shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0644)) == -1) {
        perror("Error creando segmento memoria con shmget");
        exit(1);
 }

 /* Enlaza (attach) el segmento de memoria para obtener un puntero al mismo: */
data = (int *) shmat(shmid, (void *)0, 0);
if (data == (int *)(-1)) {
        perror("Ernor obteniendo el puntero con shmat");
        exit(1);
}

            // Continua ->
```

```
printf("Creando proceso hijo .....\n");
hijo=fork() ;
switch(hijo)
{
        case -1:    printf("error creando proceso hijo\n");
                    exit(-1);
        case 0:     /* estamos en el hijo, escribir valores en segmento de memoria */
            printf("Soy el hijo con PID (escribo en el buffer):%d\n", getpid());
           for (i=0;i<10;i++)
           {
                    sem_wait(sem_1);      /* Sección de entrada */
                    /* Sección Crítica */
                       data[i]=i;
                    /* Fin de sección Crítica */
                    sem_post(sem_1); /* Sección de salida  */
                    sleep(2);
           }
        /* libero semáforos */
        sem_close(sem_1);

          /* Deconecta (detach) el segmento de memoria compartida: */
        if (shmdt((char *)data) == -1) {
                perror("Error desconectado segmento de memoria con shmdt");
                exit(1);
        }
        printf("Soy el hijo y termino.....\n");
        break;
// Continua ->
```

Bifurcación


```
default: /*estamos en el padre, leer valores del segmento compartido */
        printf("Soy el padre con PID (leo del buffer):%d\n", getpid());
        /* muestra el contenido del buffer */
        for (j=0;j<20;j++)
        {
                sem_wait(sem_1);              /* Sección de entrada */

                /* Sección Crítica: muestra los 10 valores */
                        for (i=0;i<10;i++)
                                printf("%3d ",data[i]);
                        printf("\n");
                /* Fin de sección Crítica */
                sem_post(sem_1); /* Sección de salida  */
                sleep(1);
        }
        /* libero semáforos */
        sem_close(sem_1);

        /* Desconecta (detach) el segmento de memoria compartida: */
        if (shmdt((char *)data) == -1) {
                perror("Error desconectado segmento de memoria con shmdt");
                exit(1);
        }
// Continua ->
```

Ejemplo: eliminar memoria compartida

```
    printf("Soy el padre espero que termine el hijo .....\n");
    wait(0); /* Esperar que acabe el hijo */

    printf("Soy el padre destruyo los semáforos/memoria compartida y termino\n");

    /* destruyo semaforos (solo el padre) */
    sem_unlink("/sem_1");

    /* elimino segmento de memoria compartida */
    if (shmctl(shmid, IPC_RMID , NULL) == -1) {
            perror("Error eliminando segmento de memoria con shmctl");
            exit(1);
    }
  }
  exit(0);
}
```


Ejemplo: memoria compartida

```
Creando semáforos .....
Creando Región de memoria compartida .....
Creando proceso hijo .....
Soy el hijo con PID:4488
Soy el padre con PID:4487
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   1   0   0   0   0   0   0   0   0
  0   1   0   0   0   0   0   0   0   0
  0   1   2   0   0   0   0   0   0   0
  0   1   2   0   0   0   0   0   0   0
  0   1   2   3   0   0   0   0   0   0
  0   1   2   3   0   0   0   0   0   0
  0   1   2   3   4   0   0   0   0   0
  0   1   2   3   4   0   0   0   0   0
  0   1   2   3   4   5   0   0   0   0
  0   1   2   3   4   5   0   0   0   0
  0   1   2   3   4   5   6   0   0   0
  0   1   2   3   4   5   6   0   0   0
  0   1   2   3   4   5   6   7   0   0
  0   1   2   3   4   5   6   7   8   0
  0   1   2   3   4   5   6   7   8   0
  0   1   2   3   4   5   6   7   8   9
Soy el padre espero que termine el hijo .....
Soy el hijo y termino.....
Soy el padre destruyo los semáforos/memoria compartida y termino.....
```

```
[/u0/sitr/sitr001]ipcs -m
IPC status from <running system> as of Wed Nov  5 11:58:17 2008
T         ID      KEY        MODE       OWNER    GROUP
Shared Memory:
m          0   0x50000444 --rw-r--r--      root      root
m        901   0x000050f2 --rw-r--r--   sitr001       alu
[/u0/sitr/sitr001]
```

Comunicación por Paso de Mensajes

- Permite a los procesos intercambiar mensajes mediante colas (*Messages Queues*) (*el descriptor es un entero*)
- Las operaciones básicas de comunicación las proporciona el SO:

  send (destino, mensaje)
  receive (origen, mensaje)
- *Cabeceras:*

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

- *Funciones (IPC):*

```
key_t ftok(const char *path, int id);
int msgget (key_t key, size_t size, int msqflg);
int msgctl (int msqid, int cmd, struct msqid_ds *buf);

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


Colas de mensajes POSIX

- POSIX implementa una especificación más potente de las colas de mensajes
- *Cabecera:*

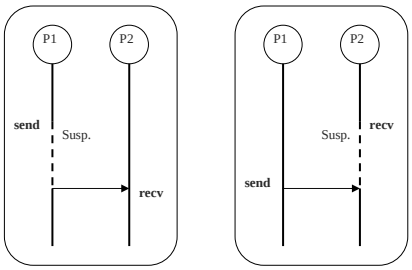
```
#include <mqueue.h>
```

- *Funciones POSIX:*

```
mqd_t mq_open (const char *name, int oflag);
mqd_t mq_open (const char *name, int oflag,
                        unsigned long mode, mq_attr attr );

int mq_close (mqd_t mqdes);
int mq_unlink (const char *name);

int mq_send (mqd_t mqdes, const char *msg_ptr,
                size_t msg_len, unsigned int msg_prio);
ssize_t mq_receive (mqd_t mqdes, char *msg_ptr,
           size_t msg_len, unsigned int *msg_prio);

int mq_notify (mqd_t mqdes, const struct sigevent *notification);
int mq_setattr (mqd_t mqdes, const struct mq_attr *mqstat,
                        struct mq_attr* omqstat);
```

Características del enlace

- ***Capacidad del enlace***: capacidad de almacenamiento de mensajes en el canal (nº de mensajes que pueden permanecer en el puerto sin ser leídos)
  - Capacidad cero ⇒ comunicación **síncrona**
    - La comunicación sólo se lleva a cabo cuando emisor y receptor han invocado *send* y *receive* respectivamente.
    - Los procesos se sincronizan mediante el modelo "*Rendez-vous*" o cita (el primero que llega se bloquea)



Características del enlace (VI)

- **Capacidad n>0** ⇒ comunicación **asíncrona** (*n* finito)
  - Una operación *receive* de un canal vacío suspende al proceso invocante
  - Una operación *send* de un canal lleno puede:
    - Suspender al proceso invocante
    - Perder el mensaje
    - Devolver un mensaje de error
  - Para notificar la lectura de de un mensaje y así simular un sistema síncrono se usan mensajes de reconocimiento ACK

```
P                              Q

send(Q, mensaje)               receive(P,mensaje)
receive(Q,ACK)                 send(P,ACK)
```

Características del enlace (VII)

- En enlaces de capacidad limitada, cuando los productores son muy rápidos es posible desbordar la capacidad del enlace y es necesario utilizar un *protocolo de control de flujo* para reducir la velocidad de los productores
- En comunicaciones asíncronas no se garantiza el orden correcto de los mensajes recibidos y es necesario adoptar un protocolo (*timestamps)* para asegurar el orden