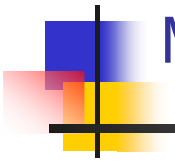
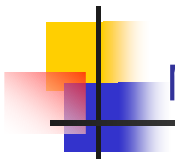


Mecanismos IPC System V

Ampliación de Sistemas Operativos (prácticas)
E.U. Informática en Segovia
Universidad de Valladolid



Mecanismos IPC System V: generalidades (1)

- Existen tres tipos de mecanismos IPC (*Inter-Process Communication*) disponibles:
  - Colas de mensajes (*message queues*): permiten el intercambio de mensajes entre cualquier proceso o servidor (local)
  - Semáforos: permiten sincronizar la ejecución de procesos (locales) no relacionados
  - Memoria compartida (*shared memory*): permiten compartir memoria entre procesos (locales) no relacionados
- El acceso a cualquier recurso se permite o no, en función de los permisos establecidos cuando se creó éste
- Se entenderá por recurso o bien una cola de mensajes, o bien un array de semáforos, o un segmento de memoria compartida
- Antes de utilizar un recurso IPC es necesario crearlo. El usuario que lo crea se dice que es el creador (*creator*), si bien éste puede ceder su control a otro usuario, que se conoce como poseedor (*owner*). Después, el recurso sólo puede ser destruido por el creador o por el poseedor

Mecanismos IPC System V: generalidades (2)

- Un recurso IPC se identifica mediante un identificador numérico (*id*). Habitualmente, el creador define una clave (*key*), la cual debe usarse para acceder al recurso. El proceso de usuario puede utilizar entonces esta clave en la llamada al sistema "**`getxxx`**" para obtener el identificador numérico del recurso. Este identificador será el que deba utilizar, a partir de entonces, cada vez que se realice una operación sobre el recurso. La función de librería **`ftok()`**, proporciona un medio para traducir nombres completos de fichero o cadenas a claves numéricas (*keys*)
- Existen límites, definidos tanto por el sistema como por la implementación, que se refieren al número y tamaño de los recursos de cada tipo.
- Para cada tipo de recurso: cola de mensaje, array de semáforos o segmento compartido, existe una estructura asociada (**`msquid_ds`**, **`semid_ds`** y **`shmid_ds`**, respectivamente) que soporta las propiedades específicas de cada recurso. Además, cada recurso IPC tiene asociada una estructura **`ipc_perm`** que define el creador, poseedor y permisos de acceso para el recurso

Mecanismos IPC System V: permisos (1)

- Cada recurso tiene asociado una estructura del tipo `ipc_perm` que define el creador (*creator*), poseedor (owner) y permisos de acceso para el recurso

```
struct ipc_perm
    key_t key;    /* establecido por el creador */
    ushort uid;   /* euid y egid  del poseedor*/
    ushort gid;
    ushort cuid;  /* euid y egid del creador*/
    ushort cgid;
    ushort mode;  /* modos de acceso en los 9 bits más bajos */
    ushort seq;   /* número de secuencia */
```

- El proceso creador es por defecto el poseedor. El poseedor puede ser reasignado por el creador y tiene los mismos permisos que éste. Sólo el poseedor, creador o root pueden eliminar el recurso

Mecanismos IPC System V: permisos (1)

- Los nueve bits más bajos del parámetro `flags` proporcionado por el usuario en la llamada al sistema del tipo `get`, se comparan con los valores almacenados en el campo `ipc_perms.mode` para determinar si el acceso solicitado está permitido. En el caso de que la llamada al sistema del tipo `get` sea para crear el recurso, estos bits son inicializados con los valores proporcionados por el usuario en la llamada
- Al igual que para ficheros, los permisos de acceso se especifican para la lectura, escritura y ejecución del usuario, grupo y resto del usuario (el permiso de ejecución carece de sentido). Por ejemplo, los permisos `0624` concede permisos de lectura/escritura para el poseedor, escritura para el grupo y lectura para el resto
- En el caso de la memoria compartida, nótese que los accesos de lectura/escritura para los segmentos se determinan por una bandera separada que no se almacena en el campo `mode`. Los segmentos de memoria compartida que se pueden escribir, por definición, también se pueden leer (independientemente del permiso de lectura)
- Los campos `cuid`, `cgid`, `key` y `seq` no pueden ser modificados por el usuario

Mecanismos IPC System V: llamadas al sistema (1)

- Cada recurso IPC proporciona una llamada al sistema de tipo **`get`**, otra de tipo **`ctl`** y, al menos una llamada al sistema de tipo **`op`**, para realizar operaciones sobre el recurso, que permiten al usuario crear o utilizar un recurso (**`get`**), definir el comportamiento o destruir del recurso (**`c`**tl) y manipular recursos (**`op`**)
- Las llamadas al sistema de tipo **`get`**:
  - Habitualmente, **`get`** tiene como parámetro una clave de recurso (**`key`**) y devuelve un identificador numérico (**`id`**), útil para sucesivas operaciones sobre el recurso. El identificador **`id`** es un índice (o descriptor) de la tabla de recursos. El sistema mantiene un número de secuencia que es incrementado cuando se destruye un recurso de modo que es probable que falle el acceso a un recurso utilizando un identificador obsoleto
  - El usuario también especifica los permisos y otras características de comportamiento para el acceso actual. Las banderas son establecidas mediante operaciones **`OR`** a nivel de bit con los permisos del recurso cuando se hace la llamada, por ejemplo:

```
msg_flags = IPC_CREAT | IPC_EXCL | 0666;
id = msgget (key, msg_flags);
```

Mecanismos IPC System V: llamadas al sistema (2)

- Las llamadas al sistema de tipo **`get`** (continuación)
  - **`key`** :
    - **`IPC_PRIVATE`** => se inicializa una nueva instancia del recurso especificado
  - **`flags`** :
    - **`IPC_CREAT`** : el recurso es creado para la nueva clave, si no existía previamente
    - **`IPC_CREAT | IPC_EXCL`** : la llamada falla si el recurso ya existe con la clave especificada
  - Valor devuelto
    - Un identificador que será el que hay que utilizar para futuros accesos al recurso
  - Notas
    - Nótese que **`IPC_PRIVATE`** no es una bandera sino una clave especial que asegura (cuando la llamada se realiza con éxito) que se cree un nuevo recurso
    - El uso de **`IPC_PRIVATE`** no significa que el recurso sea inaccesible a otros procesos de usuario. Para ello deben establecerse adecuadamente los permisos de acceso
    - No hay forma de garantizar que un proceso gane en exclusiva el acceso al recurso. **`IPC_CREAT | IPC_EXCL`** simplemente garantiza (en caso de éxito) que se inicializa un nuevo recurso, pero no que se accede a éste con exclusividad
    - Las diferentes versiones de la llamada al sistema **`get`** son: **`msgget`**, **`semget`** y **`shmget`**.

Mecanismos IPC System V: llamadas al sistema (3)

- Llamadas al sistema de tipo **`ctl`**:
  - Proporcionan o alteran la información almacenada en la estructura que describe el recurso y que está indexado por el identificador numérico **`id`**

```
#include <sys/msg.h>
...
struct msqid_ds buf;
...
err = msgctl (id, IPC_STAT, &buf);
if (err)
          /* Fallo al hacer la operación ctl */
else
        printf ("creator uid = %d\n", buf.msg_perm.cuid);
  ....
```

  - Operaciones soportadas por las llamadas de tipo **`ctl`**:
    - **`IPC_STAT`** : permite leer y almacenar en el buffer indicado, los atributos del recurso. El usuario debe tener acceso de lectura al recurso
    - **`IPC_SET`**: permite modificar los atributos del recurso con la información existente en el buffer indicado. El usuario debe ser propietario, creador o **`root`**
    - **`IPC_RMID`**: destruye el recurso. El usuario debe ser el propietario, creador o **`root`**
      - La orden **`IPC_RMID`** es inmediata para el caso de las colas de mensaje o arrays de semáforos. Sin embargo para segmentos de memoria compartida, el recurso sólo se destruye una vez que se realiza la última operación **`detach`** sobre dicho segmento
    - Las operaciones de tipo **`ctl`** disponibles son **`msgctl`**, **`semctl`** y **`shmctl`**
      - La llamada a **`semctl`** proporciona un número de órdenes adicionales que permiten al usuario determinar o establecer los valores de los semáforos dentro del array

Mecanismos IPC System V: llamadas al sistema (4)

- Llamadas al sistema de tipo **op**:
  - Permiten enviar o recibir mensajes de una cola de mensajes (**msgsnd** y **msgrcv**, respectivamente), consultar o alterar el valor de un semáforo (**semop**), e incluir en el espacio de memoria de un proceso o excluirlo de éste, un segmento de memoria compartida (**shmat** y **shmdt**, respectivamente)
  - El flag **IPC_NOWAIT** hace que la operación especificada falle, generando un error **EAGAIN**, si el proceso debe bloquearse como consecuencia de la ejecución de la llamada. En otro caso la llamada es bloqueante, es decir, si por ejemplo se realiza una operación consistente en la recepción de un mensaje de una cola de mensajes y el mensaje no está disponible, el proceso se bloqueará hasta que haya un mensaje disponible
  - flags : **IPC_NOWAIT** => no bloquea al proceso y retorna con un error si se requiere esperar como consecuencia de la llamada

Colas de mensajes (1)

- Una cola de mensajes está soportada por una estructura del tipo **msquid_ds** que se reserva e inicializa cuando se crea el recurso IPC. Algunos de los campos de la estructura se pueden modificar (si se desea) mediante la llamada a **msgctl**. La memoria utilizada por el recurso se libera cuando éste se destruye mediante la llamada a **msgctl**

```
struct msqid_ds
    struct ipc_perm msg_perm;
    struct msg *msg_first;  /* primer mensaje en cola (interno) */
    struct msg *msg_last;   /* último mensaje en cola (internal) */
    time_t msg_stime;       /* instante del último msgsnd */
    time_t msg_rtime;       /* instante del último msgrcv */
    time_t msg_ctime;       /* último cambio de tiempo */
    struct wait_queue *wwait; /* escritores esperando (interno) */
    struct wait_queue *rwait; /* lectores esperando (interno) */
    ushort msg_cbytes;      /* número de bytes usados en la cola */
    ushort msg_qnum;        /* número de mensajes en cola */
    ushort msg_qbytes;      /* máximo número de bytes de la cola */
    ushort msg_lspid;       /* pid del último msgsnd */
    ushort msg_lrpid;       /* pid del último msgrcv */
```

Colas de mensajes (2)

- Para enviar o recibir un mensaje, el usuario debe reservar un área de memoria con una estructura parecida a **`msgbuf`**, pero con el array **`mtext`** del tamaño deseado (pudiendo ser incluso 0, si no hubiera campo de datos en el mensaje)
  - Todos los mensajes tienen un tipo (entero positivo) asociado, de modo que un lector de mensajes de la cola pueda seleccionar qué mensajes leer examinando este tipo

```
struct msgbuf
    long mtype;      /* tipo de mensaje (véase msgrcv) */
    char mtext[1];   /* área de datos del mensaje (contigua) */
```

- El usuario debe tener permisos de escritura para enviar un mensaje y permisos de lectura para recibir un mensaje de la cola
- Cuando se llama a **`msgsnd`**, el mensaje del usuario se copia en una estructura interna del tipo **`msg`** y se añade a la cola. Una operación **`msgrcv`** leerá el mensaje y liberará el espacio asociado a la estructura **`msg`** asociada

Colas de mensajes: msgget

- **`msgget`**: inicializa una cola de mensajes

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
msqid = msgget (key_t key, int msgflg);
```

  - Parámetros
    - **`key`** : un entero, habitualmente obtenido de **`ftok()`** o **`IPC_PRIVATE`**
    - **`msgflg`** :
      - **`IPC_CREAT`** : utilizado para crear un nuevo recurso, si éste no existe aún
      - **`IPC_EXCL | IPC_CREAT`** : utilizado para crear una cola y para asegurar fallo si ya existiese
      - **`rwxrwxrwx`** : permisos de acceso (en octal).
  - Valor devuelto
    - **`msqid`** (un identificador de tipo entero) si hay éxito ó -1 en caso de error
  - Se reserva espacio para una nueva cola de mensajes si no existe ya una cola para la clave dada. Los permisos de acceso especificados se copian en la estructura **`msg_perm`** y se inicializan los campos de **`msquid_ds`**. El usuario puede utilizar el flag **`IPC_CREAT`** o la clave **`IPC_PRIVATE`**, si debe reservarse espacio para una nueva cola de mensajes. Si la cola correspondiente a la clave ya existe, se verifican los permisos de acceso

Colas de mensajes: msgsnd

- **msgsnd**: envía un mensaje a la cola de mensajes

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd (int msqid, struct msgbuf *msgp, int msgsz, int msgflg);
```

  - Parámetros
    - **msqid** : el identificador numérico obtenido en la llamada a msgget
    - **msgp** : apuntador al área de datos que contiene el mensaje a enviar (**msgp->mtype** debe ser positivo)
    - **msgsz** : el tamaño en bytes del campo de datos del mensaje (campo **mtext**)
    - **msgflg** : 0 (invocación bloqueante si no hay espacio en la cola para albergar el nuevo mensaje) o **IPC_NOWAIT** (en caso de que no haya sitio en la cola para el nuevo mensaje, se retorna inmediatamente, pero con un código de error **EAGAIN**)
  - Valor devuelto
    - 0 si no hay error ó –1 si ha ocurrido un error
  - El área de datos del mensaje se almacena en la estructura interna msg y los campos msg_cbytes, msg_qnum, msg_lspid y msg_stime de la estructura msquid_ds se actualizan. El sistema desbloquea a uno de los lectores que estuviera esperando por uno de los mensajes del tipo especificado en la llamada

Colas de mensajes: msgrcv

- **msgrcv**: lee un mensaje del tipo especificado de la cola de mensajes

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv (int msqid, struct msgbuf *msgp, int msgsz, long msgtyp,int msgflg);
```

  - Parámetros
    - **msqid**: el id obtenido en la llamada a **msgget**
    - **msgp**: dirección del espacio de memoria (del proceso de usuario) donde copiar el mensaje leído de la copia
    - **msgsz**: tamaño máximo del mensaje a recibir (no puede superar el tamaño máximo del área de datos de usuario donde se guardará)
    - **msgtyp** :
      - 0 => obtiene el primer mensaje disponible en la cola
      - > 0 => obtiene el primer mensaje que coincida con el tipo **msgtyp** especificado
      - < 0 => obtiene el primer mensaje cuyo tipo sea menor o igual que **abs(msgtyp)**
    - **msgflg** :
      - **IPC_NOWAIT**: Retorna inmediatamente, si no hay mensajes en la cola del tipo solicitado
      - **MSG_NOERROR**: El mensaje se trunca si es mayor que **msgsz**.
      - **MSG_EXCEPT**: Tiene sentido cuando **msgtyp > 0**, ya que se recibe cualquier mensaje a excepción del los del tipo especificado por **msgtyp**
  - Valor devuelto
    - La cantidad de bytes efectivamente leídos de la cola si no hay error ó –1 si ha ocurrido un error
  - Se mantiene una disciplina FIFO, el primer mensaje que satisfaga la especificación dada por **msgtyp** se devuelve. Cuando **msgtyp <0,** se busca en toda la cola y se devuelve el primer mensaje del tipo más pequeño
  - Si su longitud es menor que **msgsz** o si se especifica la bandera **MSG_NOERROR**, su texto y tipo se copian en **msgp->mtext** y **msgp->mtype**, y el mensaje se elimina de la cola
  - Los campos **msg_cbytes**, **msg_qnum**, **msg_lrpid** y **msg_rtime** son modificados. Se despierta a los escritores esperando en la cola

Colas de mensajes: msgctl

- **`msgctl`**: operación de control sobre una cola

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl (int msqid, int cmd, struct msqid_ds *buf);
```

  - Parámetros
    - **`msqid`**: el id obtenido en la llamada a **`msgget`**
    - **`buf`**: espacio de memoria del usuario para almacenar o obtener la información leída o escrita en la operación **`msgctl`**
    - **`cmd`** : **`IPC_STAT`**, **`IPC_SET`**, **`IPC_RMID`** (véase mecanismos IPC System V: llamadas al sistema (3)).
  - Valor devuelto
    - 0 si no hay error ó –1 si ha ocurrido un error
  - **`IPC_STAT`** hace una copia de los datos de la estructura que soporta la cola de mensajes en el área de datos proporcionado en la invocación
  - En el caso de **`IPC_SET`**, el tamaño de la cola de mensajes (**`msg_qbytes`**) y el **`uid`** , **`gid`** , modo (9 bits bajos) de la estructura **`msg_perm`** son establecidos a partir de los valores proporcionados en la invocación. El campo **`msg`**_ctime es actualizado por el sistema
  - Nótese que sólo el usuario **`root`** puede incrementar el limite del tamaño de la cola de mensajes por encima de **`MSGMNB`**
  - Cuando se destruye al cola (**`IPC_RMID`**), el número de secuencia se incrementa y todos los lectores y escritores bloqueados son despertados. Estos procesos retornan de las invocaciones previas que les mantenían bloqueados, pero con una condición de error **`EIDRM`**

Ejemplo colas de mensaje: emisor.c

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

#define LON 200

struct my_msgbuf {
   long mtype;
   char mtext[LON];
};
```

```
int main(void) {
   struct my_msgbuf buf;
   int msqid;
   key_t key;

   if ((key = ftok("emisor.c", 'B')) == -1) {
       /* Debe coincidir con el utilizado en receptor.c */
       perror("ftok");
       exit(1);
   }
   if ((msqid = msgget(key, 0644)) == -1) { /* conexión a la cola */
       perror("msgget");
       exit(1);
   }
   printf("Teclea lineas de texto, ^D para terminar:\n");
   buf.mtype = 1; /* el tipo de mensaje, no importa en este caso */
   while(fgets(buf.mtext, LON, stdin) && !feof(stdin)) {
      if (msgsnd(msqid, (struct msgbuf *)&buf, sizeof(buf), 0)==-1)
          perror("msgsnd");
   }
   return 0;
}
```

Ejemplo colas de mensaje: receptor.c

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

#define LON 200
struct my_msgbuf {
   long mtype;
   char mtext[LON];
};
```

```
int main(void) {
   struct my_msgbuf buf;
   int msqid;
   key_t key;

   if ((key = ftok("aso_pr05_ej01_receptor.c", 'B')) == -1) {
      perror("ftok"); exit(1);
   }
   if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
       perror("msgget"); exit(1);
   }
   printf("receptor: Dispuesto a recibir mensajes.\n");
   for (;;) { /* El receptor nunca termina */
      if (msgrcv(msqid,(struct msgbuf *)&buf,sizeof(buf),0,0)==-1) {
          perror("msgrcv");
          exit(1);
      }
      printf("receptor: \"%s\"\n", buf.mtext);
    }
    if (msgctl(msqid, IPC_RMID, NULL) == -1) {
       perror("msgctl"); exit(1);
   }
   return 0;
}
```