

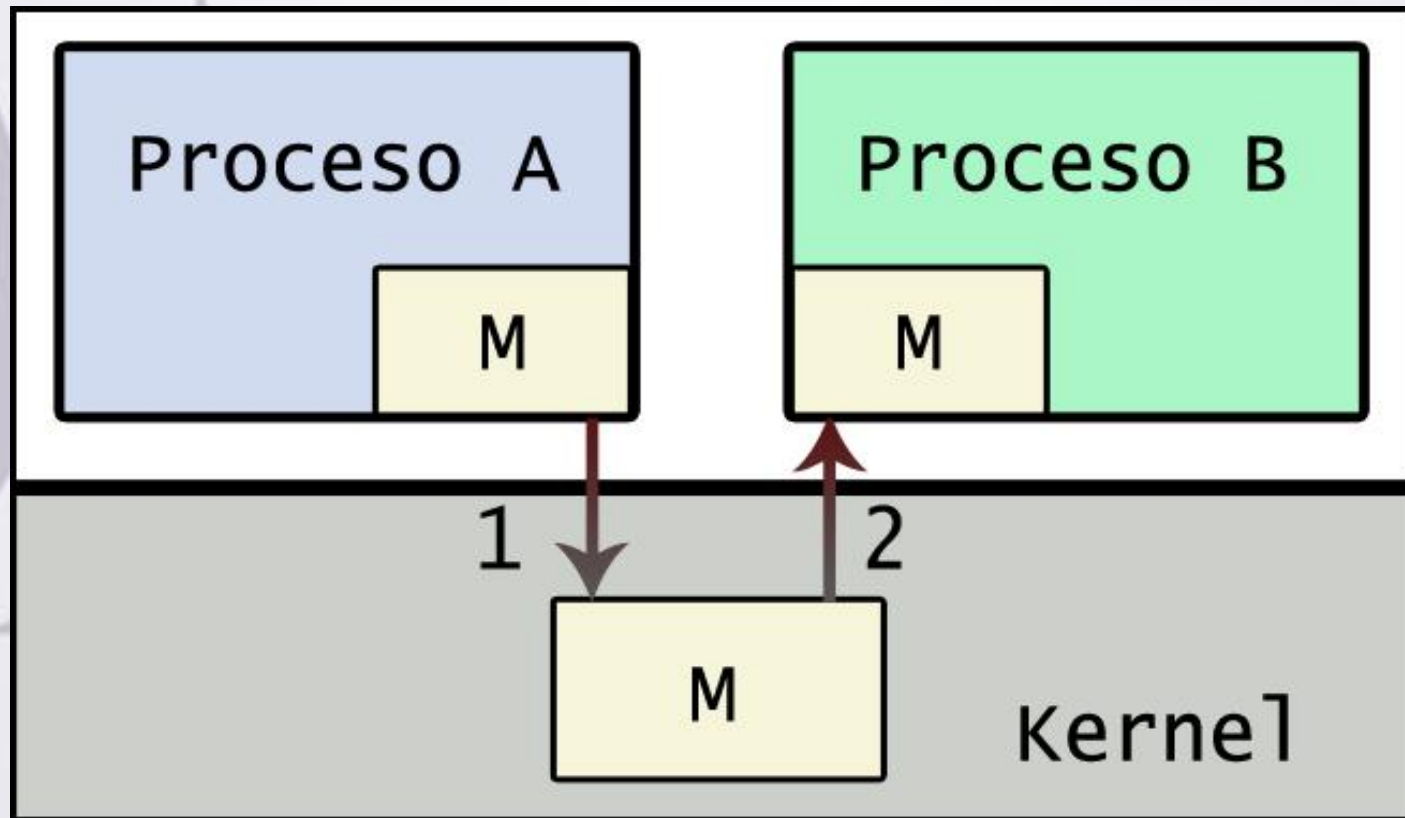


COLA DE MENSAJES POSIX



Cola de mensajes

Permite comunicar unidades de mensajes entre procesos





Cola de mensajes

- Permite escribir mensajes para ser leídos por diferentes procesos que conocen su identificador, por lo que podemos utilizarla en procesos no relacionados.
- Permiten a los procesos intercambio de datos en forma de mensajes.
- Los procesos introducen mensajes y se van almacenando.
- Cuando un proceso extrae un mensaje, este mensaje se borra.
- Los mensajes se ordenan por prioridad y luego para la misma prioridad por antigüedad.
- Se pueden abrir configuradas como bloqueantes o no bloqueantes.
- Las colas de mensajes tiene persistencia de kernel.
- Para compilar un proceso con cola de mensajes utilizamos la opción:
-lrt (library realtime). Ejemplo `gcc -o mq01 mq01.c -lrt`



Apertura de una cola de mensajes

La función `mq_open()` crea una nueva cola de mensajes o abre una existente.

```
#include <mqueue.h>
mqd_t  mq_open(const char *name, int oflag, ... mode_t mode,
struct mq_attr *attr );
```

```
mq_open(nombre, banderas, permisos, attr );
```

Devuelve un descriptor de cola de mensajes en caso de éxito, o `(mqd_t) -1` en caso de error.

Para abrir una cola de mensajes existente, se requieren como mínimo solo dos argumentos (`name`, `oflag`).



Cola de mensajes

El argumento `oflag` es una máscara de bits que controla varios aspectos de la operación de **`mq_open()`**.

| Flag | Description |
|-------------------------|---------------------------------|
| <code>O_CREAT</code> | Crea una cola si esta no existe |
| <code>O_RDONLY</code> | Abre para lectura solamente |
| <code>O_WRONLY</code> | Abre para escritura solamente |
| <code>O_RDWR</code> | Abre para lectura y escritura |
| <code>O_NONBLOCK</code> | Abrir en modo de no bloqueo |

El argumento `modo` es una máscara de bits que especifica los permisos que se colocan en la cola de un nuevo mensaje.

El argumento `*attr` es un puntero a una estructura `mq_attr` que especifica los atributos de la nueva cola de mensajes. Si `*attr` es `NULL`, la cola se crea con atributos por defecto definidos por la aplicación.



Cierre de una cola de mensajes:

La función `mq_close()` cierra el descriptor de cola de mensajes `mqdes`.

```
#include <mqueue.h>  
int mq_close(mqd_t mqdes);
```

Devuelve 0 si tiene éxito, o `-1` en caso de error.

El cierre de una cola de mensajes no la elimina.



Eliminar una cola de mensajes:

Para eliminar la cola de mensajes utilizamos `mq_unlink()`.

```
#include <mqueue.h>  
int mq_unlink(const char *name);
```

Devuelve 0 si tiene éxito, o -1 en caso de error.

Marca a la cola de mensaje para ser destruida cuando todos los procesos dejen de usarla.



Envío de mensajes:

La función `mq_send()` añade el mensaje a la cola de mensajes a la que hace referencia el descriptor `mqdes`.

```
#include <mqueue.h>
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t
msg_len, unsigned int msg_prio);
```

`mq_send(descriptor, mensaje, tamaño del mensaje, prioridad);`

Devuelve 0 si tiene éxito, o -1 en caso de error.

El argumento `msg_len` especifica la longitud del mensaje apuntado por `msg_ptr`.

Cada mensaje tiene una prioridad dada por un número entero no negativo, especificado por el argumento `msg_prio` (0 es la más baja prioridad).



Recepción de mensajes:

La función `mq_receive()` elimina el mensaje más antiguo con la más alta prioridad de la cola de mensajes a que se refiere el `mqdes` y devuelve el mensaje en el buffer apuntado por `msg_ptr`.

```
#include <mqueue.h>
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t
msg_len, unsigned int *msg_prio);
```

`mq_receive(descriptor, mensaje, tamaño máximo, prioridad);`

Devuelve el número de bytes recibidos si tuvo éxito, o `-1` si hay error.

El argumento `msg_len` es utilizado por el proceso que recibe para especificar el número de bytes de espacio disponible. Este valor debe ser mayor o igual que el especificado en `mq_msgsize` (atributo).



Consulta de atributos:

La función `mq_getattr()` devuelve una estructura `mq_attr` que contiene información acerca de la descripción de la cola de mensajes asociada con el descriptor `mqdes`.

```
#include <mqueue.h>
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
```

Devuelve cero si tuvo éxito y -1 en caso de error.

`attr.mq_curmsgs` : cantidad de mensajes que están actualmente en la cola.
`attr.mq_maxmsg` : número máximo de mensajes.
`attr.mq_msgsize` : tamaño máximo de mensaje (en bytes).
`attr.mq_flags` : devuelve las banderas para la descripción de la cola de mensajes abierta (0 o `O_NONBLOCK`), asociada al descriptor `mqdes`.



El rincón de C

Puntero a una estructura

El argumento `*attr` es un puntero a una estructura `mq_attr` que especifica los atributos de la nueva cola de mensajes.

```
struct mq_attr {  
    long mq_flags;      /* Flags: 0 or O_NONBLOCK */  
    long mq_maxmsg;     /* Max. # of messages on queue */  
    long mq_msgsize;    /* Max. message size (bytes) */  
    long mq_curmsgs;    /* # of messages currently in queue */  
};
```

La función,

```
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
```

espera un puntero a la estructura. Como se explicó anteriormente, una función puede recibir valores por referencia (punteros) o por valores. Cuando el dato a suministrar implica cargar una cantidad importante de bytes en la pila del proceso haciendo un *push*, es más eficiente solo suministrar un puntero a la estructura que toda la estructura en sí.



Bibliografía

Kerrisk, Michael. *The linux programming Interface*. 2011. **Capítulo 52.**