Programación Concurrente y Distribuida

Práctica 6 Curso 2018/19 3º Curso

Comunicación entre Procesos

Introducción

El sistema operativo *Linux* proporciona un mecanismo de comunicación indirecta implementando paso de mensajes a través de buzones tipados unidireccionales o bidireccionales. Es decir, cada mensaje se clasifica según su tipo, de manera que el receptor puede seleccionar mensajes de un tipo concreto. Además, cada buzón se crea en base a una clave y con permisos de lectura y/o escritura, lo que permite generar enlaces lógicos unidireccionales y/o bidireccionales entre múltiples procesos compartiendo un mismo buzón. Por otro lado, dos procesos pueden comunicarse a través de diferentes enlaces lógicos compartiendo varios buzones.

Dicho mecanismo se incluye como parte de los *IPCs* proporcionados por el sistema operativo definiendo un conjunto de llamadas para su utilización similar a las descritas en la práctica 3 para memoria compartida, cuyas declaraciones se encuentran en los ficheros de cabecera <*sys/ipc.h*> y <*sys/msg.h*>.

El comando `ipcs -q` muestra información relativa a las colas de mensajes existentes en un momento dado, incluyendo: la *clave* con la que se creó; su *identificador;* el propietario; los permisos de lectura/escritura; el número de mensajes almacenados; y el número de bytes que ocupan dichos mensajes.

Para eliminar una cola de mensajes en la línea de comandos, basta con utilizar `ipcrm msg` seguido del identificador de la cola de mensajes a eliminar.

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Utilizar paso de mensajes como mecanismo de comunicación entre procesos.
- Adquirir experiencia en el desarrollo de soluciones a problemas de concurrencia usando paso de mensajes.

Entorno

Los ordenadores del laboratorio ya están preparados para realizar las prácticas. Si el alumno desea realizar las prácticas en su ordenador deberá tener instalado Linux (en el laboratorio está instalada la distribución OpenSuse 12.3 y es sobre la que se deberá garantizar el correcto funcionamiento de las prácticas).

Creando Colas de Mensajes

Antes de poder enviar o recibir mensajes es necesario crear e inicializar una cola de mensajes (*buzón)* mediante la llamada `msgget()`. La función `msgctl()` permite realizar diferentes operaciones de control sobre una cola de mensajes, en particular eliminar dicha cola.

Ejercicio 1 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `msgget`.

Escriba un proceso `p1` que cree una cola de mensajes. Compruebe con el comando `ipcs -q` que la cola se crea correctamente según los permisos indicados.

Ejercicio 2 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `msgctl`.

Escriba un proceso `p2` que elimine la cola de mensajes cuyo *id* recibe como parámetro. Compruebe con el comando `ipcs -q` que la cola se elimina correctamente.

Enviando y recibiendo Mensajes

Las funciones `msgsnd()` y `msgrcv()` permiten depositar y extraer, respectivamente, un mensaje de un *buzón* previamente creado, y pueden ejecutarse en modo bloqueante y/o no bloqueante.

Ejercicio 3 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `msgsnd`.

Escriba un proceso `p3` que envíe un mensaje a la cola de mensajes creada con el proceso `p1` del Ejercicio 1. Compruebe con el comando `ipcs -q` que el mensaje se ha almacenado en la cola correctamente.

Ejercicio 4 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `msgrcv`.

Escriba un proceso `p4` que extraiga el mensaje enviado por el proceso `p3` del Ejercicio 3. Compruebe con el comando `ipcs -q` que el mensaje se ha retirado correctamente.

Modifique los procesos `p3` y `p4` y haga diferentes pruebas combinando envíos y recepciones en modo bloqueante y no bloqueante.

Ejercicio 5 (trabajo previo a la sesión de Laboratorio):

Escriba una solución al problema de los filósofos utilizando como mecanismo de sincronización una única cola de mensajes para los cinco filósofos. Cada proceso `filósofo` recibirá como parámetro su posición en la mesa (`1 < i < 5`) y todos deberán coger primero el tenedor de su izquierda (`i`).

Escriba un proceso `inic_filosofos` encargado de inicializar la cola de mensajes.

Ejercicio 6:

El hecho de utilizar una única cola de mensajes en el Ejercicio 5 impide que dos filósofos puedan coger de manera concurrente dos tenedores diferentes. Escriba una nueva solución, utilizando el número de colas de mensajes que considere, que permita el mayor grado de concurrencia posible entre los filósofos.

Ejercicio 7:

Escriba, utilizando colas de mensajes, una solución al problema clásico de *Los Fumadores*:

"En torno a una mesa hay cuatro personas: tres fumadores y un proveedor. Cada fumador fuma cigarrillos uno detrás de otro. Para poder fumarse un cigarrillo se necesitan tres ingredientes: papel, tabaco y fósforos. Uno de los fumadores tiene papel, el otro tabaco y el otro fósforos. El proveedor tiene una cantidad infinita de los tres ingredientes. Inicialmente, el proveedor coloca dos de los ingredientes en la mesa, escogidos al azar. El fumador que tiene el ingrediente que falta toma lo que hay en la mesa, lía un cigarrillo y se lo fuma. Cuando termina, el proveedor coloca otros dos ingredientes y el ciclo se repite una y otra vez."

Resumen

Los principales resultados esperados de esta práctica son:

- Adquirir experiencia en la programación concurrente.
- Aprender a utilizar mecanismos de comunicación entre procesos como *colas de mensaje*.
- Resolver problemas de concurrencia utilizando paso de mensajes.

Como trabajo adicional del alumno, se proponen las siguientes líneas:

- Resolver los ejercicios de las prácticas anteriores utilizando *colas de mensajes*.
- Implementar otros problemas clásicos de concurrencia con *colas de mensajes*.
- Examinar otros mecanismos de comunicación entre procesos como los *pipes*.