Sistemas Operativos II

Práctica 4 - Sincronización de procesos con paso de mensajes

Objetivo

Conocer el funcionamiento básico de pase de mensajes como mecanismo de solución al problema de las carreras críticas y desarrollar habilidades en su manejo.

1. Descripción de la práctica

- Repasar la solución vista en clase al problema del productor-consumidor con pase de mensajes.
- Programar la solución al problema del productor-consumidor utilizando pase de mensajes entre dos procesos según la plantea Tanenbaum. El código del productor debe llamarse *prod.c* y el del consumidor *cons.c*. Ten en cuenta lo siguiente:
  - El tamaño de cada buffer debe ser N=5.
  - Los items serán de tipo *char*.
  - El productor debe llamar a una función para producir items (su tarea principal). Esta tarea consistirá en leer los carácteres de un fichero de texto plano, que será el primer argumento de la línea comando al ejecutar el proceso. Deben leerse todos los carácteres del fichero, que constituirán los items enviados al cosumidor. La lectura de cada carácter debe ir acompañada por una espera de un número aleatorio de entre 0 y T microsegundos usando la función *usleep*. El valor de T debe ser el segundo de los argumentos de la línea comando al ejecutar el proceso.
  - El consumidor debe llamar a una función para consumir items (su tarea principal). Esta tarea consistirá en almacenar en un fichero, cuyo nombre será el primer argumento de la línea comando al ejecutar el proceso, todos los carácteres recibidos. La lectura de cada carácter debe ir acompañada por una espera de un número aleatorio de entre 0 y T microsegundos usando la función *usleep*. El valor de T debe ser el segundo de los argumentos de la línea comando al ejecutar el proceso.
  - El buffer de items debe funcionar como una cola FIFO (First In First Out), de manera que el primer item enviado sea el primero consumido.
  - Ambos procesos deben ejecutarse en terminales diferentes.
  - Al final del enunciado hay una plantilla para el código del productor. Ten en cuenta que:
    - Falta la función *productor()*, en la que se deben usar las funciones *mq_send* y *mq_receive* para enviar y recibir mensajes.
    - Completa el código del productor, y escribe el del consumidor.
    - Para compilar debes usar la librería -lrt (Realtime Extensions library).

2. Ejercicio opcional

- Resuelve el mismo problema considerando que las colas funcionan como pilas LIFO (Last In First Out).

Entrega

- Deben subirse los códigos en C con comentarios exhaustivos sobre su funcionamiento y manejo.
- La fecha de entrega es la indicada en el Campus Virtual.

```
#define MAX_BUFFER 5          /* tamaño del buffer */

/* cola de entrada de mensajes para el productor */
mqd_t almacen1;
/* cola de entrada de mensajes para el consumidor */
mqd_t almacen2;

void main(void) {
    struct mq_attr attr;        /* Atributos de la cola */

    attr.mq_maxmsg = MAX_BUFFER;
    attr.mq_msgsize = sizeof (char);

    /* Borrado de los buffers de entrada
       por si existían de una ejecución previa*/
    mq_unlink("/ALMACEN1");
    mq_unlink("/ALMACEN2");

    /* Apertura de los buffers */
    almacen1 = mq_open("/ALMACEN1", O_CREAT|O_WRONLY, 0777, &attr);
    almacen2 = mq_open("/ALMACEN2", O_CREAT|O_RDONLY, 0777, &attr);
    if ((almacen1 == -1) || (almacen2 == -1)) {
        perror ("mq_open");
        exit(EXIT_FAILURE);
    }

    productor();

    mq_close(almacen1);
    mq_close(almacen2);

    exit(EXIT_SUCCESS);
}

productor(void) {
    /* ToDo */
}
```