Sistemas Operativos II [G4012227] [2022/2023]

Práctica 4 - Sincronización de procesos con paso de mensajes

Resumen

El objetivo de esta práctica es el de conocer el funcionamiento básico de pase de mensajes como mecanismo de solución al problema de las carreras críticas. Desarrollar habilidades en su manejo.

1. Descripción de la práctica

- Programa el problema del **productor-consumidor** utilizando **paso de mensajes entre procesos**.
- Debes implementar la solución propuesta por Tanenbaum vista en clase de teoría usando buffers en productor y consumidor.
- El consumidor debe **llenar el buffer del productor** antes de empezar a consumir.
- El **tamaño** de cada **buffer** debe ser **N=5**.
- **Prueba** que se soluciona el problema del productor-consumidor **forzando el llenado y vaciado de los buffers** con llamadas a **sleep** de esperas adecuadas.
- La función **producir_elemento()** debe generar un elemento de la forma: **'a' + (iter % MAX_BUFFER)**, siendo iter igual a la iteración en la que se genera el item en el productor desde 0 hasta DATOS_A_PRODUCIR-1.
- Debes **asegurarte que no quedan mensajes** en las colas de mensajes, ni en la del consumidor ni en la del productor.
- El código debe tener **dos versiones** (implementadas por separado).
  - Versión LIFO: El consumidor debe consumir los items en Last-In-First-Out orden.
  - Versión FIFO: El consumidor debe consumir los items en First-In-First-Out orden.
- Implementa también el **código** del productor y del consumidor por **separado** y usa los siguientes **nombres para cada implementación**:
  - Versión LIFO: productor_LIFO.c y consumidor_LIFO.c
  - Versión FIFO: productor_FIFO.c y consumidor_FIFO.c
- Al final del enunciado tienes un **ejemplo de manejo de cola de mensajes**.
  - Falta implementar la función productor(), en la que se deben usar las funciones mq_send para enviar mensajes a una cola y mq_receive para recibirlos. **Estudia** el código y el uso de estas funciones.
  - **Completa** el código del productor, y **escribe** el del consumidor.
  - Para compilar debes usar la **librería -lrt** (Realtime Extensions library).

2. Formato y fecha de entrega

- Hacer un breve **informe (máximo 2 páginas)** incluyendo comentarios y las conclusiones obtenidas. Sube al Campus Virtual un **único archivo comprimido llamado Apellido1_Apellido2-Apellido1_Apellido2.zip** incluyendo el código con comentarios exhaustivos sobre su funcionamiento/compilación y el informe.
- En la nota obtenida se tendrá en cuenta la estructura y claridad tanto del informe como del código.
- La evaluación de esta práctica puede estar sujeta a una revisión individual.
- Fecha de entrega: Hasta las 23:59 del 6º día contando desde el último día de prácticas del grupo que te corresponda (ver entrega en el Campus Virtual).

3. Código adjunto: ejemplo de manejo de cola de mensajes

El código adjunto es parte del productor, en el que se utilizan las funciones de POSIX:

- mq_open para crear o abrir una cola de mensajes en los dos procesos.
- mq_close para cerrar la cola de mensajes.
- mq_unlink para borrar la cola de mensajes.

Falta implementar la la función productor(), en la que se deben usar las funciones:

- mq_send para enviar mensajes a una cola.
- mq_receive para recibirlos.

```
#define MAX_BUFFER 5 /* tamaño del buffer */
#define DATOS_A_PRODUCIR 100 /* número de datos a producir/consumir */

/* 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 */
}
```