Laboratorio 5 - Comunicación entre procesos y sincronización

Responder las preguntas en el archivo `respuestas.txt`.

Ejercicio 1

En este ejercicio usaremos la librería de mensajes POSIX. El manual `mq_overview` presenta una introducción general al API de colas de mensajes.

En este ejercicio vamos a usar las siguientes funciones:

- `mq_open()`: crea una nueva cola de mensajes, o abre una ya existente.
- `mq_send()`: envía un mensaje a la cola de mensajes.
- `mq_receive()`: recibe un mensaje.
- `mq_close()`: cierra el descriptor de una cola de mensajes.
- `mq_unlink()`: elimina una cola de mensajes.
- `mq_getattr()`: recupera los atributos de una cola de mensajes.

Deben completar los siguientes programas, en el directorio `mq`:

- `mq_open.c`: crea una cola de mensajes.
- `mq_send.c`: envia un mensaje a travez de la cola de mensajes especificada.
- `mq_receive.c`: lee el mensaje de mayor prioridad en la cola de mensajes indicada.
- `mq_attr.c`: muestra información acerca de la cola de mensajes especificada.
- `mq_unlink.c`: elimina la cola de mensajes indicada.

Una vez completados, deben poder crear colas de mensajes y envíar y recibir mensajes por medio de las mismas.

Responder también las siguientes preguntas:

1. ¿Que sucede si se ejecuta `mq_receive` sobra una cola de mensajes vacía?
2. Enviar varios mensajes, con `mq_send`, algunos con distinta prioridad y otros con la misma prioridad. Luego, recuperarlos con `mq_receive`. ¿En qué orden son recuperados de la cola de mensajes? ¿Cómo se ordenan los mensajes con la misma prioridad?

Ejercicio 2

En este ejercicio usaremos el API de POSIX para crear y utilizar segmentos de memoria compartida. Mediante estos segmentos, diferentes procesos pueden intercambiar datos de una manera más rapida que mediante el uso de mensajes. El manual `shm_overview` tiene una introducción al API de memoria compartida de POSIX.

Las principales funciones que vamos a usar en el ejercicio son:

- `shm_open()`: crea un nuevo objeto de memoria compartida, o abre uno ya existente.
- `ftruncate()`: cambia ("trunca") el tamaño del segmento de memoria compartida.

- `mmap()`: mapea el segmento de memoria compartida indicado dentro del espacio de direcciones del proceso.
- `close()`: cierra el descriptor de un segmento de memoria compartida.
- `shm_unlink()`: elimina el segmento de memoria compartida indicado.

Completar los siguientes programas en el directorio `shm` haciendo uso del API de memoria compartida de POSIX, que utilizan memoria compartida para escribir y leer una serie de datos:

- `shm_create.c`: crea un segmento de memoria compartida.
- `shm_write.c`: escribe una serie de datos en el segmento de memoria compartida indicado.
- `shm_read.c`: lee los datos que se encuentren el segmento de memoria compartida especificado.
- `shm_unlink.c`: elimina el segmento de memoria compartida.

Responder también la siguiente pregunta:

1. ¿Cómo sabe `shm_read` cuanto datos puede leer del segmento de memoria compartida?

Ejercicio 3

El programa `eco.c` crea dos procesos hijos que se comunican por medio de una tubería. Uno de los procesos lee una línea desde la *entrada estándar*, y la envía por la tubería. El segundo proceso lee esta línea de la tubería y la imprime por la *salida estándar*. El programa termina cuando se ingresa una linea en blanco (osea, un `\n`).

Modificar el programa de manera que ambos procesos se comuniquen mediante paso de mensajes, en lugar de una tubería.

¡Fin del Laboratorio 5!