Headers

Para el funcionamiento completo del programa, se han considerado tres headers o interfaces, además de las ya existentes pertenecientes a la librería estándar ANSI C. Estas son: "memoriaI.h", "definitions.h" y "semaforoI.h".

La primera define los métodos necesarios para gestionar el funcionamiento de un objeto de memoria compartida: métodos de creación y obtención, así como métodos para destruir, modificar y consultar la información almacenada en el objeto.

La segunda define las rutas de los procesos telefono y linea, y sus clases, y también define los semáforos y la memoria compartida. También define una estructura de proceso (process_id y clase).

La tercera y última presenta una estructura similar a la primera, solo que el método de creación, obtención y destrucción se aplica a un semáforo POSIX. Presenta también un método `signal()`, el cual incrementa el semáforo, y otro de espera `wait()`, el cual decrementa el semáforo.

Manager.c

En esta clase es donde se va a ejecutar todo lo necesario para obtener el resultado deseado del programa, es decir, contiene desde la creación de semáforos y de la memoria compartida hasta las llamadas a las clases linea.c y telefono.c, pasando por la instalación de la señal de interrupción (Control+C) y la inicialización de las correspondientes tablas de procesos. Acabará finalmente con la espera de la finalización de las líneas, cierre de los procesos hijos y liberación de recursos.

En este punto vamos a indagar concretamente en aquellos métodos que han supuesto cierta dificultad una vez comenzada la fase de programación, los cuales son: *`lanzar_proceso_telefono()`* y *`terminar_procesos_específicos()`*.

- **lanzar_proceso_telefono()**. Para lanzar un proceso telefono necesitaremos declarar un id de proceso. Tras declararle vamos a poner un `switch` donde se dan dos casos al crear un proceso a partir del "pid":
  - `case -1`: trata cuando se ha producido un **error** al crear el proceso. Aquí lanzamos un aviso y procedemos a llamar a los métodos terminar_procesos() y liberar_recursos() y finaliza el programa con código de error.
  - `case 0`: aquí se da el caso de que sí se haya **creado correctamente** el proceso. Entonces, llamamos a ejecutar a la clase telefono dentro de la condición de un if para comprobar si la ejecución se ha realizado con éxito. En caso de error, es decir, que la ejecución devuelva -1, imprime en pantalla un mensaje de error y finaliza el programa con código de error.

  Tras esto, una vez fuera de la sentencia del `switch`, se procede a almacenar en la tabla de procesos del telefono (g_process_telefono_table) el "pid" y la clase (su tipo de proceso).

- **terminar_procesos_especificos()**. Este método es el que hemos empleado para finalizar los procesos "telefono" y "linea" por separado, ya que está preparado para ambos.
  Para detener los procesos comenzamos con un bucle `for`, el cual va a iterar para poder acceder al proceso de la tabla y así matarlo. Después comprobamos con un `if` si en la tabla de procesos el pid del proceso sobre el que estamos en ese momento es **distinto de 0**, es decir, que **no esté terminado**.
  En caso afirmativo vamos a imprimir un aviso de que se está terminando ese proceso. Tras esto vamos a comprobar con un bucle `if` si el intento de matar ese proceso nos devuelve un -1, donde en caso afirmativo estamos ante un error y se nos notificará que ha ocurrido un problema al emplear la sentencia `kill()` en dicho proceso.

Linea.c

Como la línea consistirá en un proceso de un solo uso, todas las instrucciones estarán en el método `main()`, cerrando con un `return EXIT_SUCESS`.

Lo primero que hará el proceso "línea" será establecer la semilla de generación de números aleatorios, con su id de proceso, a través de la primitiva `srand()`. Después, tomará los semáforos de la variable compartida, teléfono y línea, todos de la línea de argumentos. También se tomará la variable compartida, almacenada en un manejador.

Para modificar el valor de la variable compartida, nos apoyaremos en una variable local que almacenará la consulta a esta, y su posterior modificación (incremento prefijo). Todo esto regulado con su semáforo "mutex" correspondiente, evitando que otros procesos línea y teléfono la manipulen a la vez.

Esta modificación se efectuará cuando la línea haya terminado de esperar una llamada (simulado con una instrucción de espera del proceso, de duración aleatoria de entre 1 y 30 segundos). Posteriormente la línea esperará a que exista un teléfono disponible, expresado como un decremento del semáforo de teléfonos. Cuando el proceso teléfono incremente su semáforo (equivalente a disponible o esperando llamada), desbloqueará este proceso, permitiendo desviar la línea su llamada al teléfono en cuestión (señalizado con incremento de semáforo de línea). De esta manera, al teléfono le llegará que se ha desviado una llamada y podrá tener la conversación.

Posteriormente, el proceso "línea" finaliza su ejecución (proceso "manager" lo notificará por terminal).

Telefono.c

A diferencia del proceso "línea", el proceso "teléfono" será de ejecución infinita, por lo que hemos definido un método `void()` llamado `telefono`, el cual está compuesto en su esencia por un bucle infinito, representado como `while(1)`, implicando en una inexistente condición de parada; su única finalización será forzada por parte del proceso "manager".

A este método le pasamos los semáforos "línea", "teléfono" y "mutex" para la variable compartida, el manejador de la variable compartida y la variable entera auxiliar para almacenar su valor, para consultas y modificaciones.

Una vez este proceso entra en el bucle, el teléfono establece que está en espera de que le desvíen una llamada, lo que implica incrementar su semáforo, y decrementar el de las líneas. De esta manera, cuando una línea ha detectado un teléfono disponible, podrá desviar su llamada a él, desbloqueando este proceso por el semáforo de las líneas. Acto seguido, se decrementa en una unidad el valor de llamadas en espera (variable compartida, regulado por semáforo "mutex").

Por último, se simulará una conversación telefónica, con una parada del proceso teléfono de una duración aleatoria de entre 10 y 20 segundos (**gestión de aleatorios similar a `linea.c`**). Después, se vuelve al principio del bucle, cuando termina la conversación y el teléfono vuelve a estar disponible.

semaforoI.c y memoriaI.c

Estos dos archivos desarrollan las funciones declaradas en sus respectivos *headers*. Para ello, se emplean funciones ya existentes a partir de librerías propias de ANSI C, con tal de simplificar su uso:

semaforoI.c

- crear_sem() = incremento del semáforo = sem_open() con opción O_CREAT
- get_sem() = obtención semáforo = sem_open() con opción O_RDWR
- destruir_sem() = cierre y eliminaciój semáforo = sem_close() + sem_unlink
- signal_sem() = modificación semáforo = sem_post()
- wait_sem() = decremento del semáforo = sem_wait()

memorial.c

- crear_var() = apertura de objeto memoria compartida, definición de su tamaño y mapeo/desmapeo de dicho objeto.
- obtener_var() = obtención de objeto memoria compartida a partir de un descriptor.
- close_var() = cierre del descriptor y *unlink* (destrucción) de la variable.
- modificar_var() = remapeo de objeto memoria compartida con valor nuevo
- consultar_var() = remapeo de objeto memoria compartida con obtención del contenido del mapeo en un puntero valor

Compilación y ejecución con Makefile

Una vez hemos terminado de programar las clases, es hora de comprobar si funciona lo que hemos escrito. Para ello usaremos un archivo MAKEFILE. Los archivos MAKEFILE contienen reglas que especifican las acciones a realizar por el comando *make* y directivas que indican a éste algunas formas especiales de interpretación del contenido del archivo MAKEFILE.

El comando *make* funciona basándose en las reglas especificadas en el archivo MAKEFILE. En esas reglas se expresa, para cada archivo que deba construirse, de qué otros archivos dependen.

En este caso, cuando escribamos *make* por terminal, nos va a crear directorios nuevos:

- obj/ Aquí se alojarán los archivos objeto (extensión .o), necesarios para la compilación
- exec/ En este directorio se guardan los ejecutables resultantes de la compilación

En la ejecución de make, se crearán 5 archivos objeto: semaforoI.o, memoriaI.o, manager.o, telefono.o y linea.o, con opción para depuración (ggdb), usando como includes la carpeta include del proyecto (por eso creamos archivo objeto de la lista) y visualización de todas las advertencias (Wall), y 3 ejecutables: manager, telefono y linea.

Por último, para ejecutar el resultado de hacer make empleamos `./<directorio_ejecutable> <argumento1> <argumento2>`. En el caso de este programa debemos escribir en la línea de comandos la siguiente sentencia de ejecución: `./exec/manager <nº telefonos> <nº lineas>`.

En el archivo MAKEFILE también hay definida una última regla de borrado, denominada clean, para borrar las carpetas creadas anteriormente, incluyendo su contenido. Para llevarlo a cabo, sólo debemos escribir en nuestro espacio de trabajo donde esté MAKEFILE el comando *make clean*.