Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 3. Programación multi-hilo. Control de procesos de fabricación.

Grado de Ingeniería en Informática

Curso 2024/2025

Índice

1. Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con los servicios para la gestión de procesos que proporciona POSIX.

Para la gestión de procesos ligeros (hilos), se utilizarán las llamadas al sistema *pthread_create, pthread_join, pthread_exit,* y para la sincronización de los mismos se utilizarán ***mutex*** y ***variables condicionales***:

- **Pthread_create:** crea un nuevo hilo que ejecuta una función que se le indica como argumento en la llamada.
- **Pthread_join:** realiza una espera por un hilo que debe terminar y que está indicado como argumento de la llamada.
- **Pthread_exit:** finaliza la ejecución del proceso que realiza la llamada.

El alumno deberá diseñar y codificar, en lenguaje C y sobre el sistema operativo Linux, un programa que actúe como gestor de procesos de fabricación incluyendo varios threads encargados de gestionar distintas fases de la fábrica, y un thread que realiza la planificación de las fases.

2. Descripción de la práctica

El objetivo de esta práctica es codificar una simulación del funcionamiento de una factoría en la que hay diferentes roles para los trabajadores. Los roles deberán trabajar de forma concurrente permitiendo que los elementos utilizados en la factoría tengan una gestión y movimiento correctos.

Los roles son:

- **Factory manager:** Es el thread destinado a poner en marcha a los threads encargados de la producción de la factoría. Su función principal es la de iniciar el proceso de fabricación poniendo en marcha los threads *process_manager*. El *factory_manager* puede iniciar tantos threads *process_manager* como se le indique en el fichero de carga.

- **Process manager:** Es el thread encargado de poner en marcha los procesos ligeros que van a funcionar junto a cada cinta de transporte para que se genere un sistema *productor-consumidor*. Los elementos que intervienen en este subsistema son:

    - **Productor:** Es un proceso ligero que se encargará de producir tantos elementos como le indique el *process_manager*, y los pondrá a disposición de un *consumidor* en una cinta de transporte entre ellos dos.

    - **Consumidor:** Es un proceso ligero que se encargará de recoger tantos elementos como le mande su compañero *productor* mediante la cinta de transporte.

Por lo tanto, el *factory_manager* es el jefe de la factoría, que comunica a sus *n* gestores *process_manager* cuántos elementos debe producir cada uno. Finalmente, cada *process_manager* mandará a dos trabajadores (*producer* y *consumer*) que realicen las tareas encomendadas por el jefe.

NOTA: Las cintas de transporte serán implementadas mediante **colas circulares**.

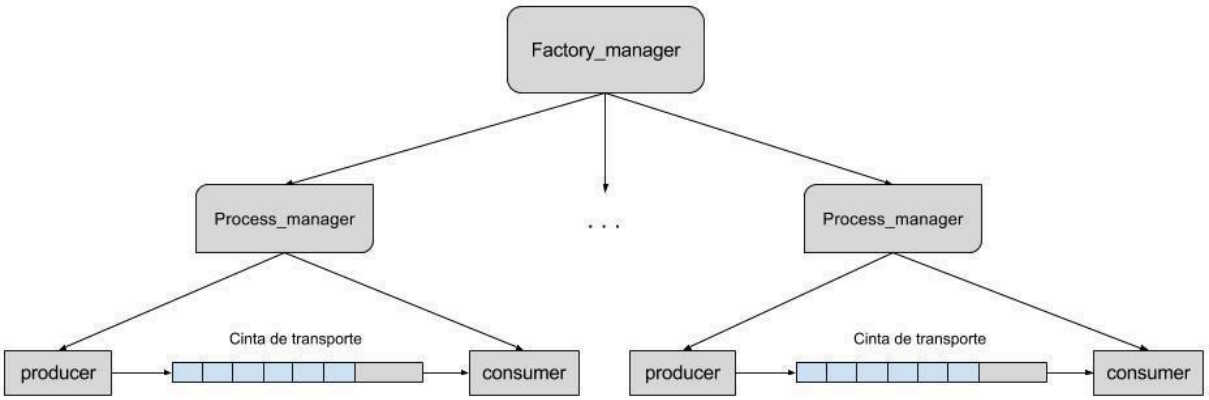

Ilustración 1. Ejemplo de funcionamiento con 2 cintas.

2.1. Gestor de procesos de fabricación

El gestor de procesos de fabricación se corresponde con las funciones del *factory_manager*, y debe implementarse en un fichero denominado **factory_manager.c**. Las tareas de este rol son: capturar los parámetros de entrada de la factoría, que son **introducidos mediante un fichero**, y lanzar los *n process_manager* que le indican los parámetros de entrada. El programa únicamente tendrá un argumento: la ruta de un fichero, que será el fichero de carga (descrito en el apartado 2.3).

Cuando el *factory_manager* crea un thread de tipo *process_manager*, le debe indicar los siguientes datos: **<Id> <tamaño máximo de la cinta><Nº de productos a generar>**, dónde:

- **Id:** es el identificador de la cinta que se le asigna (viene determinado por los datos incluidos en el fichero de carga).
- **Tamaño máximo de la cinta:** es el máximo número de elementos que pueden ser almacenados en la cinta asignada (tamaño del buffer circular).
- **Nº de productos a generar:** cada grupo de trabajo *productor-consumidor* tiene asignado un número de productos que tiene que generar, indicado por el fichero de entrada y asignado a los *process_manager* por parámetro desde el proceso padre *factory_manager*.

La sincronización entre el factory_manager y los process_manager se realiza mediante semáforos controlados por el *factory_manager*. La función de los semáforos que van a manejar los procesos es la de controlar el funcionamiento de un número máximo de procesos de fabricación y su orden. La factoría tendrá un tamaño máximo, definido en el fichero de carga, y **no podrá haber más procesos de fabricación de los que entran en la factoría (pero sí puede haber menos)**. Además, los procesos de fabricación deberán esperar en un semáforo a ser despertados por el *factory_manager*.

2.2. Procesos de fabricación

Los threads de los procesos de fabricación son los que se van a encargar de realizar las tareas de lanzamiento de los trabajadores de la factoría, y sus funciones son las de *process_manager*. Se debe

implementar la funcionalidad correspondiente a este punto en un fichero denominado **process_manager.c**. Sus argumentos serán los definidos en el apartado 2.1.

Cada *process_manager* creado debe generar un sistema *productor-consumidor* implementado con hilos (*threads)*. El proceso productor será el encargado de insertar elementos en una cinta de transporte, mientras que el proceso consumidor será el encargado de recoger elementos de la cinta.

Este problema que se pide implementar es un ejemplo clásico de sincronización de procesos: al compartirse una cinta de transporte (**buffer circular**), hay que realizar un control sobre la concurrencia a la hora de depositar objetos en la cinta y a la hora de extraerlos.

2.2.1. Cola sobre un buffer circular

La comunicación entre los productores y los consumidores se realizará mediante las cintas de transporte. Estas cintas se corresponden con colas sobre buffers circulares. Debe crearse una cola circular para cada productor-consumidor. Dado que constantemente se van a producir modificaciones sobre este elemento, se deben implementar mecanismos de control de la concurrencia para los procesos ligeros.

La cola circular y sus funciones deben estar implementadas en un fichero denominado **queue.c**, y debe contener, al menos, las siguientes funciones:

- **int queue_init (int num_elements)**: función que crea la cola y reserva el tamaño especificado como parámetro.
- **int queue_destroy (void)**: función que elimina la cola y libera todos los recursos asignados.
- **int queue_put (struct element * ele)**: función que inserta elementos en la cola si hay espacio disponible. Si no hay espacio disponible, debe esperar hasta que pueda ser realizada la inserción.
- **struct element * queue_get (void)**: función que extrae elementos de la cola si hay elementos en ella. Si la cola está vacía, se debe esperar hasta que haya un elemento disponible.
- **int queue_empty (void)**: función que consulta el estado de la cola y determina si está vacía (return 1) o no (return 0).
- **int queue_full (void)**: función que consulta el estado de la cola y determina si está llena (return 1) o aún dispone de posiciones disponibles (return 0).

La implementación de esta cola debe realizarse de forma que no haya problemas de concurrencia entre los threads que están trabajando con ella. Para ello se deben utilizar los mecanismos propuestos de ***mutex* y *variables de condición*.**

El objeto que debe almacenarse y extraerse de las distintas cintas debe corresponderse con una estructura definida con los siguientes campos:

- **int num_edition**: representa el orden de creación dentro de la cinta.
- **int id_belt**: representa el ID de la cinta en la que se crea el objeto (el *factory_manager* se lo indica al proceso por parámetro).
- **int last**: será 0 si el elemento que se inserta no es el último, y será 1 si el objeto es el último que creará el productor.

2.3. Fichero de entrada a Factory Manager

El programa principal (*factory_manager*) debe ser capaz de reconocer los ficheros de entrada que tengan el formato indicado a continuación. La información contenida se representa de la siguiente forma: **<N.º max cintas> [<ID cinta> <Tamaño de cinta> <N.º elementos>]**$^{+}$, donde:

- **N.º max cintas:** representa el número máximo de *process_managers* que se pueden crear. Si tras la lectura del fichero se detectan más procesos que el máximo declarado, el fichero es inválido y el programa debe finalizar retornando -1.
- **ID cinta:** se asigna un número de cinta a cada proceso *process_manager* para identificar sus productos en las trazas.
- **Tamaño de cinta:** es el valor máximo de elementos que puede contener una cinta de transporte (tamaño del buffer circular que hay entre *productor-consumidor*).
- **N.º elementos:** Número de elementos que debe generar la cinta asignada.

La carga de los datos del fichero se debe realizar en el proceso *factory_manager*, realizando una comprobación de la integridad de los datos (por ejemplo, no se puede tener un número de cintas menor o igual que 0).

Un ejemplo de fichero de entrada válido sería:

```
4 5 5 2 1 2 3 3 5 2
```

- Se podrán crear, como mucho, **4** ***process_manager***.
- El primer *process_manager*, con ID ***5***, creará una cinta con un *tamaño máximo para* ***5*** elementos, y tendrá que *producir* ***2*** elementos.
- El segundo *process_manager* con ID *1*, creará una cinta con *tamaño máximo para 2* elementos, y tendrá que *producir 3* elementos.

- Finalmente, el tercer *process_manager*, con ID ***3***, creará una cinta con *tamaño máximo para **5*** elementos, y deberá *producir **2*** elementos.

NOTA: los ID de los *process_manager* no tienen por qué ser consecutivos. El orden debe ser traducido por el *factory_manager* y preservarlo utilizando mecanismos de sincronización.

2.4. Integración y ejemplos de ejecución

Un ejemplo del flujo de ejecución de la factoría podría ser el siguiente:

- Ejecución del *factory_manager*:
  - Lee el fichero con la especificación de la factoría, cuyo nombre se pasa como parámetro.
  - Abre el fichero, obtiene la información y cierra el fichero.
  - Crea las estructuras de sincronización necesarias para el correcto funcionamiento de la factoría.
  - Creación de los threads *process_manager*.
  - Para cada thread, controlar que se ejecuten en el mismo orden que se especifica en el fichero de entrada.
  - Cuando todos los threads *process_manager* han finalizado, se liberan los recursos y se finaliza el programa.
- Ejecución del *process_manager*:
  - Obtención de los parámetros de entrada.
  - Esperar hasta que el *factory_manager* da la orden de comenzar, mediante un semáforo.
  - Crear e inicializar la cinta de transporte.
  - Crear los dos procesos ligeros: *producer* y *consumer*. El primero crea los elementos y los inserta en la cinta hasta que no tenga más elementos que fabricar. El consumidor obtendrá los elementos de la cinta hasta que no se vayan a elaborar más.
  - Cuando los hilos han finalizado, liberará los recursos utilizados y terminará el programa.

Para asegurar el correcto funcionamiento de las operaciones, el programa debe imprimir las siguientes trazas (**a través de la salida estándar, a no ser que se especifique lo contrario**) en cada momento especificado:

- *Factory_manager*:

- Cuando ocurre un error en la apertura, lectura, cierre o análisis del fichero de entrada, se debe devolver el siguiente mensaje por la **salida estándar de error** y retornar el valor -1:
  - "[ERROR][factory_manager] Invalid file."
- Cuando un thread process_manager es creado:
  - "[OK][factory_manager] Process_manager with id <id> has been created."
- Cuando un process_manager ha terminado correctamente:
  - "[OK][factory_manager] Process_manager with id <id> has finished."
- Cuando un process_manager ha terminado con errores, utilizar la **salida de error:**
  - "[ERROR][factory_manager] Process_manager with id <id> has finished with errors."
- Antes de que finalice el programa:
  - "[OK][factory_manager] Finishing."

- *Process_manager*:
  - Cuando ocurre un error leyendo los argumentos, se utiliza la **salida de error** y se retorna -1:
    - "[ERROR][process_manager] Arguments not valid."
  - Cuando los argumentos se han procesado:
    - "[OK][process_manager] Process_manager with id <id> waiting to produce <number of elements> elements."
  - Cuando la cinta es creada:
    - "[OK][process_manager] Belt with id <id> has been created with a maximum of <maximum number of elements> elements."
  - Cuando todos los elementos se han producido:
    - "[OK][process_manager] Process_manager with id <id> has produced <number of elements> elements."
  - Si ha ocurrido algún error (threads con errores de terminación, problemas de inicialización o destrucción de la cinta, etc.), utilizando la **salida de error**:
    - "[ERROR][process_manager] There was an error executing process_manager with id <id>."
- *Queue:*
  - Cuando un elemento es insertado en la cinta:

        - "[OK][queue] Introduced element with id <num_edition> in belt <id cinta>."
    - Cuando un elemento es extraído de la cinta:
        - "[OK][queue] Obtained element with id <num_edition> in belt <id cinta>."
    - Si ocurre algún error, se debe mostrar por la **salida de error** y retornar -1:
        - "[ERROR][queue] There was an error while using queue with id: <id>."

En general, si el programa detecta un error, debe inmediatamente retornar -1. Si la ejecución es correcta, debe finalizar con retorno 0.

Un **ejemplo** de ejecución podría ser (tomado del fichero de ejemplo de la sección anterior):

```
shell>./factory_manager input_file.txt
[OK][factory_manager] Process_manager with id 5 has been created.
[OK][factory_manager] Process_manager with id 1 has been created.
[OK][factory_manager] Process_manager with id 3 has been created.
[OK][process_manager] Process_manager with id 5 waiting to
produce 2 elements.
[OK][process_manager] Process_manager with id 1 waiting to
produce 3 elements.
[OK][process_manager] Process_manager with id 3 waiting to
produce 2 elements.
[OK][process_manager] Belt with id 5 has been created with a
maximum of 5 elements.
[OK][queue] Introduced element with id 0 in belt 5.
[OK][queue] Introduced element with id 1 in belt 5.
[OK][queue] Obtained element with id 0 in belt 5.
[OK][queue] Obtained element with id 1 in belt
5[OK][process_manager] Process_manager with id 5 has produced 2
elements.
[OK][factory_manager] Process_manager with id 5 has finished.
[OK][process_manager] Belt with id 1 has been created with a
maximum of 2 elements.
[OK][queue] Introduced element with id 0 in belt 1.
[OK][queue] Introduced element with id 1 in belt 1.
[OK][queue] Obtained element with id 0 in belt 1.
[OK][queue] Introduced element with id 2 in belt 1.
[OK][queue] Obtained element with id 1 in belt 1.
[OK][queue] Obtained element with id 2 in belt 1.
[OK][process_manager] Process_manager with id 1 has produced 3
elements.
[OK][factory_manager] Process_manager with id 1 has finished.
[OK][process_manager] Belt with id 3 has been created with a
maximum of 5 elements.
[OK][queue] Introduced element with id 0 in belt 3.
[OK][queue] Introduced element with id 1 in belt 3.
[OK][queue] Obtained element with id 0 in belt 3.
[OK][queue] Obtained element with id 1 in belt 3.
```

```
[OK][process_manager] Process_manager with id 3 has produced 2
elements.
[OK][factory_manager] Process_manager with id 3 has finished.

[OK][factory_manager] Finishing.
```

EXCEPCIÓN: Si ocurre un error en algún *process_manager*, el *factory_manager* debe detectarlo, pero no finalizar la ejecución. Debería saltar al siguiente *process_manager* y permitir su ejecución.

NOTA: ESTOS MENSAJES SON LOS ÚNICOS ACEPTADOS. NO SE DEBEN INCLUIR MENSAJES DIFERENTES EN LA ENTREGA FINAL (se pueden agregar para depurar, pero esos mensajes deben ser eliminados para la entrega).

3. Entrega

3.1. Plazo de entrega

La fecha límite de entrega de la práctica en AULA GLOBAL será el **5 de Mayo de 2025 (hasta las 23:55h).**

3.2. Procedimiento de entrega de las prácticas

La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas. En concreto, **se habilitará un entregador para el código de la práctica y otro de tipo TURNITIN para la memoria de la práctica**.

2.3 Documentación a Entregar

En el entregador para el código se debe entregar un archivo comprimido en formato zip con el nombre **ssoo_p3_AAAAAAAAA_BBBBBBBBB.zip** donde A…A y B…B son los NIAs de los integrantes del grupo. El archivo debe contener:

- factory_manager.c
- process_manager.c
- queue.c
- queue.h.
- Makefile
- autores.txt - Fichero de texto en formato csv con un autor por línea. El formato es: NIA, Apellidos, Nombre

En el entregador TURNITIN deberá entregarse el fichero con el nombre:

ssoo_p3_NIA1_NIA2_NIA3.pdf

La memoria tendrá que contener al menos los siguientes apartados:

- **Portada:** con los nombres completos de los autores, NIAs, grupo al que pertenecen y direcciones de correo electrónico.
- **Índice:** con opciones de navegación hasta los apartados indicados por los títulos.

- **Descripción del código** detallando las principales funciones implementadas. NO incluir el código fuente de la práctica en este apartado. Cualquier código será automáticamente ignorado.
- **Batería de pruebas** utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:
    - Que un programa compile correctamente y sin advertencias (*warnings*) no es garantía de que funcione correctamente.
    - Evite pruebas duplicadas que evalúan los mismos flujos del programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.
    - Evitar el uso de capturas de pantalla de terminales y código.
- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuarán también los siguientes aspectos relativos a la **presentación** de la práctica:

- Debe contener portada, con los autores de la práctica y sus NIAs.
- Debe contener un índice de contenidos navegable.
- La memoria debe tener números de página en todas las páginas (menos en la portada).
- El texto de la memoria debe estar justificado.

La longitud de la memoria no deberá superar las 15 páginas (portada e índice incluidos). Es imprescindible aprobar la memoria para aprobar la práctica, por lo que no debe descuidar la calidad de la misma.

NOTA: El entregador de la memoria permite una **única entrega**. La valoración de esta entrega es la única válida y definitiva.

4. Normas

1) **Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.**
2) **Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, los alumnos involucrados (copiados y copiadores) perderán las calificaciones obtenidas por evaluación continua.**
3) **Los programas deben compilar sin warnings.**
4) **Los programas deberán funcionar bajo un sistema Linux, no se permite la realización de la práctica para sistemas Windows. Además, para asegurarse del correcto funcionamiento de la práctica, deberá chequearse su compilación y ejecución en los laboratorios de informática de la universidad y en el servidor guernika.lab.inf.uc3m.es. Si el código presentado no compila o no funciona sobre estas plataformas la implementación no se considerará correcta.**
5) **La entrega de la práctica se realizará a través de aula global, tal y como se detalla en el apartado Entrega de este documento. No se permite la entrega a través de correo electrónico sin autorización previa.**
6) **Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.**
7) **Se debe realizar un control de errores en cada uno de los programas.**

5. Anexos

5.1 man function

man es el paginador del manual del sistema, es decir permite buscar información sobre un programa, una utilidad o una función. Véase el siguiente ejemplo:

man 2 fork

Las páginas usadas como argumentos al ejecutar *man* suelen ser normalmente nombres de programas, utilidades o funciones. Normalmente, la búsqueda se lleva a cabo en todas las secciones de manual disponibles según un orden predeterminado, y sólo se presenta la primera página encontrada, incluso si esa página se encuentra en varias secciones.

Para salir de la página mostrada, basta con pulsar la tecla 'q'.

Una página de manual tiene varias partes. Éstas están etiquetadas como NOMBRE, SINOPSIS, DESCRIPCIÓN, OPCIONES, FICHEROS, VÉASE TAMBIÉN, BUGS, y AUTOR. En la etiqueta de SINOPSIS se recogen las librerías (identificadas por la directiva *#include*) que se deben incluir en el programa en C del usuario para poder hacer uso de las funciones correspondientes.

Las formas más comunes de usar *man* son las siguientes:

- **man sección elemento:** Presenta la página de elemento disponible en la sección del manual.
- **man -a elemento:** Presenta, secuencialmente, todas las páginas de elemento disponibles en el manual. Entre página y página se puede decidir saltar a la siguiente o salir del paginador completamente.
- **man -k palabra-clave:** Busca la palabra-clave entre las descripciones breves y las páginas de manual y presenta todas las que casen.

5.2 Semaphores

Los semáforos de POSIX ayudan a los procesos a sincronizar sus acciones. Un semáforo es un entero cuyo valor nunca va a ser menor que 0. Sobre este valor, se permiten dos acciones: incrementar el semáforo (*sem_post*), y decrementar el semáforo (*sem_wait*). Si el valor del semáforo es 0, la acción *sem_wait* se bloqueará hasta que el semáforo vuelva a ser superior a 0. Cuando varios procesos quieren acceder a un recurso controlado por un semáforo, éstos reducen en 1 el contador, y en el momento en que llega a 0 ningún proceso podrá acceder al recurso. Una vez finalizada la tarea, un proceso debe restablecer su contador añadiendo 1 al semáforo para que otros puedan acceder.

Para poder utilizar estos elementos se debe realizar una inclusión de *semaphore.h* en el fichero que requerirá de los semáforos así como ejecutar el linkado con la librería pthread.

6. Bibliografía

- El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- The UNIX System S.R. Bourne Addison-Wesley, 1983.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Programming Utilities and Libraries SUN Microsystems, 1990.
- Unix man pages (man function)