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

Pablo Escrivá Gallardo 100348802
Cristian Gómez López 100349207

Índice

Factory Manager

Parser

La base de nuestro intérprete de ficheros será le lectura del fichero leyendo solo los números enteros. Diferenciaremos dos etapas de lectura:

- número máximo de procesos a generar
- agrupación de las triadas: esta etapa se aprovecha también para almacenar los id´s de cada triada.

Estructuras de control

Para hacer una correcta inicialización de los *Process_manager*, primero se crean (con valor 0) y nombran los semáforos, guardando también sus id´s para despúes realizar las llamadas *sem_post* y *sem_wait* necesarias.

A continuación, se crearán los procesos hijo encargados de ejecutar cada Process_manager. Cada *Process_manager* creado recibe el nombre del semáforo y hace un wait por él. De esta forma quedarán todos los *Process_manager* preparados para ser activados desde *Factory_manager* mediante la llamada a *sem_post*.

La llamada de cada *Process_manager* mediante un *sempost* finalizará con una llamada *exit*, la cual recogerá el *Factory_manager* mediante una llamada *waitpid* (del pid del proceso que se ha ejecutado) y así de forma sucesiva hasta que cada *Process_manager* se haya ejecutado acorde al orden de los id´s especificado en el fichero.

Por último, se cerrarán y se hará *unlink* de cada semáforo creado.

Control de errores

Se lanzará mensaje de error referente al archivo en el *Factory_manager* si:

- El número de argumentos de *Factory_manager* es >2 ó <2.
- No se pudo abrir el fichero argumento.
- El número máximo de elementos a generar es 0 o negativo.
- Algún elemento de las triadas no es un número entero mayor que 0.
- Alguna triada está incompleta.
- Alguna triada tiene números negativos ó 0´s.
- Hay id´s de cinta repetidos

Se lanzará mensaje de error referente a la creación de *Process_manager* en el Factory_manager si:

- No se pudo hacer *fork()*
- No se pudo ejecutar correctamente el programa *Process_manager*
- No se pudo leer alguno de los semaforos
- No se pudo hacer señal *post* sobre alguno de los semáforos
- No se pudo cerrar algún semáforo
- No se pudo hacer *unlink* de algún semáforo

Process Manager

Ejecución

Lo primero que hará un *Process_manager* al ejecutarse es comprobar si el número de argumentos pasados es el correcto, y lo más importante se bloquerá haciendo *sem_wait* del semáforo que le corresponda.

Fin de la Espera

La espera acaba una vez el *Factory_manager* activa el semáforo correspondiente a cada *Process_manager*. En este momento estará todo listo para empezar el problema productor-consumidor.

Inicio problema Productor-Consumidor

Dos threads actúan como Productor y como Consumidor, los cuales comparten una misma cola circular.
La lógica que hemos usado para solucionar el problema de acceso a la cola al mismo tiempo por parte de consumidor y productor se ha basado en bucles infinitos con condiciones internas para realizar determinadas acciones:

- Productor: sólo podrá instertar elementos cuando la cola esté vacía. Esto se traduce en una llamada a *queue_empty()*. Además para cada una de las inserciones se añadirá al elemento insertado los atributos de la estructura elem(*num_edition, id_belt, last*). Los dos primeros atributos se usarán para hacer una correcta impresión por pantalla, mientras que last será utilizado como condición de parada del elemento insertado, ya que cuando el productor compruebe que ha insertado al último elemento, acabará el thread productor.

- Consumidor: sólo podrá extraer elementos de la cola cuando haya algo que extraer. Esto se traduce en comprobar que la cola está vacía o no. El thread consumidor tambíen usará el atributo *last* del elemento que recogerá de la cola, ya que cuando se cumpla que, en efecto el elemento recogido es el último, lo recogerá y acabará.

Habiendo terminado ambos productor y consumidor solo quedará devolver el control al *Factory_manager* mediante una salida *exit(0)*;

Control de errores

Se lanzará mensaje de error referente a la creación de *Process_manager* en el *Process_manager* si:

- El número recibido de argumentos es >5 ó <5.
- No se pudo hacer lectura del semáforo.
- No se pudo realizar la llamada *sem_wait* sobre el nombre del semáforo recibido.
- No se pudo realizar un join con éxito de alguno de los *threads*.

Cola

Hemos implementado una cola doblemente enlazada en la cual se sigue la lógica FIFO. Para ello nos hemos valido de una estructura nodo (con su *next* y *prev*) en la que hemos incluido a su vez la estructura correspondiente para los atributos de cada elemento. La implementación de las funciones que hacen posible el uso de la cola son las siguientes:

Funciones

- **int queue_init (int size):** Para inicializar la cola hemos inicializado dos punteros a nodos:

- head: para saber siempre dónde comienza la cola.
- current: para llevar un seguimiento de "el nodo actual" en las posteriores iteraciones de los demás métodos

Además, inicializamos nuestra variable de tamaño de cola a 0, para tener acceso al tamaño actual siempre que se desee.

- **int queue_put (struct element* elem)**:

Aquí la lógica es sencilla, recibimos un elemento, hacemos que ese elemento sea parte de un nuevo nodo y por último enlazamos ese nuevo nodo a la cola(conectando por supuesto su next con head). El último paso es aumentar el tamaño actual de la cola.

- **struct element * queue_get(void)**:

En este caso lo que hacemos es sacar un elemento de la cola,
para lo cual usamos un nodo auxiliar en el que se guardará lo que saquemos de la cola. Por último, reenlazamos los nodos que han quedado tras la sustracción en la cola para que la lógica total no se vea afectada y disminuimos el tamaño de la cola.

- **int queue_empty (void)**:

Se comprueba que head sea NULL, si lo es estaremos ante una cola vacía.

- **int queue_full(void)**:

Se comprueba que la cola haya superado o no el tamaño máximo permitido. para ello se llama a una función auxiliar length la cual recorrerá la la cola entera almacenando en un contador el número de nodos encontrados y si ese número coincide con el número máximo recibido al principio, estará llena.

- **int queue_destroy (void)**:

Se liberarán todos los nodos empezando por el "*head—>next*" (para no perder su referencia) y finalmente se libera el head.

Batería de pruebas

Funcionalidad probada	Prueba	Descripción	Objetivo	Resultado	
				Esperado	Obtenido
Fichero vacío	Pasar un fichero vacío como argumento	Comprobar el funcionamiento de fscanf con ficheros vacíos	Comprobar que el parser notifica de error en ficheros vacíos	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Fichero no existente	Pasar un fichero no existente como argumento	Comprobar el funcionamiento de fscanf con ficheros no existentes	Comprobar que el parser notifica de error en ficheros no existentes	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Fichero con número máximo de procesos = 0	Pasar un fichero que comience con un 0 (ej. 0 2 3 1 4 3)	Comprobar el funcionamiento del parser para num máx de procesos = 0	Evitar que se de un caso imposible	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Triadas incompletas	Pasar un fichero que contenga triadas incompletas (ej. 4 2 3 1 4 3)	Comprobar el funcionamiento del parser con triadas incompletas	Evitar que se de un caso de falta de información	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Id´s de cinta repetidos	Pasar un fichero que contenga id´s de cinta repetidos (ej. 4 2 3 1 2 3 7)	Comprobar el funcionamiento del parser con id´s de cinta repetidos	Evitar que se de un caso de de repetición en la identificación de los procesos	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Más números de los necesarios en el fichero de entrada	Pasar un fichero que contenga más triadas que las especificadas en el primer elemento (ej. 1 2 3 1 3 5 2)	Comprobar el funcionamiento del parser con más triadas de las requeridas por el primer número.	Dejar que el programa siga su curso con las triadas indicadas por el primer número ignorando los números no necesarios	Se ignoran las triadas que no tengan sentido para el máximo número de procesos permitidos.	Se ignoran las triadas que no tengan sentido para el máximo número de procesos permitidos.
Fichero que contenga caracteres que no sean números	Pasar un fichero que contenga caracteres que no sean números (ej. 4 2 3 1 A 4 3)	Comprobar el funcionamiento del parser con caracteres que no sean números.	Evitar que el programa continúe su ejecución con elementos de entrada incorrectos	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución

Funcionalidad probada	Prueba	Descripción	Objetivo	Resultado Esperado	 Obtenido
Llamada al factory manager sin los argumentos necesarios	Ejecutar factory manager con argumentos != 2 (ej. ./factory)	Ejecutar al factory solo a partir de su nombre.	Comprobar el factory no se ejecuta si no le llega al menos un argumento	Se lanza un mensaje de error parando toda la ejecución	Se lanza un mensaje de error parando toda la ejecución
Fichero que indique la creación de 1.000.000 de procesos	Pasar un fichero que en alguna de sus triadas indique producir y consumir un millón de elementos	Pasar un fichero que en alguna de sus triadas indique producir y consumir un millón de elementos	Comprobar que no la cola funciona para números extremadamente altos.	Imprime el resultado deseado	Imprime el resultado deseado

Valoración personal

Pablo Escrivá: En mi opinión esta ha sido la práctica más útil debido a que se basa en un ejemplo de un problema real, lo que nos ha permitido hacernos una idea de qué tipo de problemas nos vamos a poder encontrar en nuestra vida profesional.
En cuanto a la dificultad del ejercicio, creo que la segunda práctica fue mucho más compleja y eso se reflejó en el tiempo que le dedicamos, que ha sido bastante menos para ésta última práctica.
Por último, añadir que la parte de la estructura de datos que se nos pide en ésta práctica no ha sido explicada en clase y hemos dedicado demasiado tiempo en aprender a implementarla.

Cristian Gómez: Tal vez esta última práctica ha sido, o al menos la he notado algo más sencilla que la anterior, no por su funcionalidad en si, sino porque ya no era necesario pararse demasiado tiempo a controlar la creación de procesos hijo, llamadas a wait, waitpid…
Algo importante a comentar respecto a nuestras primeras versiones, fue tal vez que intentamos implementar la llamada a cada Process_manager con únicamente un semáforo. También me gustaría hacer hincapie en el esfuerzo que hemos tenido que hacer para implementar la cola, ya que en clase aún no habíamos visto nada realacionado. Finalmente me gustaría también agradecer la libertad de creación de mensajes de error personalizados, ya que de esa forma he sentido que se ha conseguido una control de errores mucho más sólido.