Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 3. Programación del funcionamiento de un mercado de valores (Multithread)

Grado de Ingeniería en Informática

Curso 2018/2019

Índice

1. Enunciado de la Práctica 3
1.1. Descripción de la Práctica 3
1.1.1. Código base 4
1.1.1.1. Parser 4
1.1.1.2. Operations_queue 5
1.1.1.3. stock_market_lib 6
1.1.1.4. Concurrency_layer (Trabajo a realizar) 7
1.1.2. Requisitos de concurrencia 10
1.1.3. Detalle de los ficheros de configuración 11
1.1.4. Implementación del programa principal y ejecución 12
1.2. Código Fuente de Apoyo 14
1.3. Formato de salida 15
2. Entrega 16
2.1. Plazo de entrega 16
2.2. Procedimiento de entrega de las prácticas 16
2.3. Documentación a Entregar 16
3. Normas 18
4. Anexo (man function) 19
5. Bibliografía 19

1. Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con los servicios para la gestión de procesos ligeros (hilos o *threads*) que proporciona POSIX. Se pretende que el alumno comprenda la importancia de sincronizar los hilos cuando trabajan de forma concurrente.

Para la gestión de hilos, se utilizarán las llamadas al sistema de POSIX relacionadas, como `pthread_create` y `pthread_join`. Para la sincronización de hilos, se utilizarán *mutex* y *variables condicionales* mediante llamadas como `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_cond_init`, `pthread_cond_wait`, `pthread_cond_signal`, `pthread_cond_broadcast`, `pthread_cond_destroy`.

El alumno debe diseñar y codificar, en lenguaje C y sobre sistema operativo UNIX/Linux, una simulación del funcionamiento de un mercado de valores con distintos roles representados por hilos que deberán trabajar de forma concurrente.

1.1. Descripción de la Práctica

Se pretende codificar una simulación del funcionamiento de un mercado de valores en el que existen tres roles principales. Estos tres roles deberán trabajar de forma concurrente consultando y gestionando las acciones dentro del mercado de valores. Estos tres roles son:

- **Broker:** es el encargado de insertar nuevas operaciones sobre acciones en la cola de operaciones del mercado de valores. Carga las operaciones a partir de un fichero de operaciones en *batch*. Podrá haber *n* hilos concurrentes con rol *broker*, pero sólo se ejecutará uno cada vez para mantener la integridad de los datos.
- **Operation_executer (procesador de operaciones):** es el encargado de procesar una a una las operaciones insertadas en la cola de operaciones por los brokers. Sólo habrá un hilo de procesado de operaciones.
- **Stats_reader (lector de estadísticas):** es el encargado de consultar el estado actual del mercado de valores. Podrá haber n lectores concurrentes en el sistema y podrán ejecutar todos a la vez. Ninguna tarea de modificación del mercado de valores (operation_executer o broker) puede ejecutarse durante el proceso de lectura.

Broker y operation_executer hacen uso de la cola de operaciones, mientras que los lectores accederán directamente al estado del mercado de valores.

La información sobre las acciones disponibles en el mercado de valores se podrá cargar de un fichero. Las operaciones a realizar por un determinado broker también se cargarán de un fichero de operaciones en *batch*, que contendrá una operación por cada línea.

1.1.1. Código base

El código base proporcionado, consta de varias capas que simulan el comportamiento de diferentes componentes del mercado de valores.

<table>
<tr><td rowspan="3">Bibliotecas</td><td>PARSER</td><td>OPERATIONS_QUEUE</td></tr>
<tr><td colspan="2">STOCK_MARKET_LIB</td></tr>
<tr><td colspan="2">CONCURRENCY_LAYER</td></tr>
<tr><td>main program</td><td colspan="2">CONCURRENT_MARKET</td></tr>
</table>

La capa superior, implementada en los ficheros `parser.c/parser.h` y `operations_queue.c/operations_queue.h` respectivamente incluyen las funciones auxiliares para simplificar la codificación de la funcionalidad del mercado de valores.

1.1.1.1. Parser

Incluye funciones auxiliares para recorrer el fichero de operaciones en *batch* línea por línea y para imprimir por pantalla el estado actual de todas las acciones del mercado de valores.

- `iterator * new_iterator(char * file_name)`
  - **Descripción:** genera un nuevo iterador sobre el fichero pasado por parámetro.
  - **Entrada:** Nombre del fichero sobre el que se quiere iterar (un fichero de operaciones en *batch*).
  - **Salida:** puntero al nuevo iterador o NULL en caso de error.

- `void destroy_iterator(iterator * iter)`
  - **Descripción:** destruye el iterador pasado por parámetro.
  - **Entrada:** puntero al iterador a destruir.
  - **Salida:** void.

- `int next_operation(iterator * iter, char * id, int * type, int * num_shares, int * price)`
    - **Descripción:** dado un iterador pasado por parámetro, rellena los `punteros id, type, num_shares y price` con los datos de la siguiente operación del fichero sobre el que está trabajando el iterador.
    - **Entrada:** puntero al iterador y punteros a parámetros a rellenar.
    - **Salida:** rellena los punteros con la información del operación y devuelve: 0 → ok, -1 → error.

1.1.1.2. Operations_queue

Incluye las estructuras de datos y funciones necesarias para trabajar con una cola de operaciones. Una operación contiene:

- `ID` de la empresa/acción sobre la que se quiere ejecutar la operación.
- `type`: tipo de la operación a realizar (compra (BUY) o venta (SELL)).
- `num_shares`: número de acciones a comprar/vender.
- `share_price`: precio a pagar por cada acción que se quiere comprar/vender.

Funciones (descripción de las funciones disponible en `lib/operations_queue.c` y `lib/operations_queue.h`)

```
operations_queue * new_operations_queue(int max_items);

void delete_operations_queue(operations_queue *q);

int operations_queue_empty(operations_queue *q);

int operations_queue_full(operations_queue *q);

int dequeue_operation(operations_queue *q, operation *
op);
```

```
int enqueue_operation(operations_queue *q, operation *
op);


void new_operation(operation *op, char id[ID_LENGTH], int
type, int num_shares, int share_price);
```

1.1.1.3. stock_market_lib

Contiene la funcionalidad *core* del mercado de valores. Un mercado de valores tendrá las siguientes características:

- `stocks[NUM_STOCKS]` array que contiene NUM_STOCKS acciones. Serán las acciones sobre las que se podrá operar en este mercado.
- `total_value`: es el valor total de este mercado (la suma del valor de todas las acciones del mercado).
- `avg_value`: es el valor medio de cada acción disponible en este mercado.
- `num_active_stocks`: cuántas acciones disponibles hay actualmente en el mercado.
- `stock_operations`: es la cola de operaciones pendientes. En ella los hilos *broker* añadirán nuevas operaciones y el hilo *operation_executer* las irá procesando en orden FIFO.

Cada una de las acciones incluida en el array stocks, contendrá la siguiente información acerca del valor:

- `ID`: identificador unívoco (de máximo 10 caracteres) de este valor.
- `name`: nombre de la empresa (máximo 255 caracteres).
- `total_shares`: número de acciones en las que está dividida la cotización de la empresa.
- `total_value`: valor total de la empresa.
- `current_share_value`: el valor de la empresa dividido entre todas las acciones disponibles (`total_value / total_shares`).

Funciones (descripción de las funciones disponible en `lib/stock_market_lib.c` y `lib/stock_market_lib.h`)

```
int init_market(stock_market * market, char * file_name);

void delete_market(stock_market * market);

int new_stock(stock_market * market, char id[ID_LENGTH], char
name[STOCK_NAME_LENGTH], int current_share_value, int
total_shares);

stock * lookup_stock(stock_market * market, char
id[ID_LENGTH]);

void update_market_statistics(stock_market * market);

int process_operation(stock_market * market, operation * op);

void print_market_status(stock_market * market);
```

1.1.1.4. Concurrency_layer (Trabajo a realizar)

El objetivo de la capa de concurrencia es permitir el acceso simultáneo a múltiples *brokers* al servicio de procesado de operaciones del mercado de valores. De este modo, esta capa contendrá los mecanismos de control de concurrencia necesarios para evitar problemas de concurrencia. **EL ALUMNO DEBE IMPLEMENTAR EL FICHERO `lib/concurrency_layer.c` MANTENIENDO `lib/concurrency_layer.h` DEFINICIÓN DE FUNCIONES Y ESTRUCTURAS. EL FICHERO `lib/concurrency_layer.h` NO PODRÁ SER MODIFICADO.**

El fichero `lib/concurrency_layer.c` deberá contener, al menos, la implementación de las cinco funciones definidas en `lib/concurrency_layer.h`. Se podrán añadir tantas funciones auxiliares y variables globales como se consideren necesarias (por ejemplo: mutex, variables de condición, etc).

- `void init_concurrency_mechanisms()`: inicializa los mecanismos de control concurrencia necesarios para permitir el correcto funcionamiento de la aplicación con varios hilos. No recibe parámetros ni tiene valor de retorno. Será llamada siempre en la función `main()` del programa antes de la creación de ningún hilo.

- `void destroy_concurrency_mechanisms()` destruye todos los mecanismos de control de concurrencia utilizados durante la ejecución del programa. Será llamada siempre en la función `main()` del programa después de la finalización de todos los hilos y antes de la finalización del programa.

- `void* broker(void * args)` implementa la funcionalidad de los hilos *broker* (control de concurrencia incluida). Recibe por parámetro una estructura de tipo `broker_info` que contiene:

  - `char batch_file[256]`: nombre del fichero *batch* que contiene las operaciones bursátiles a realizar por el *broker*.

  - `stock_market * market`: puntero al mercado sobre el que *broker* realizará las operaciones bursátiles.

  Pseudocódigo de la función (control de concurrencia a diseñar por el alumno):

```
Extraer la información de los datos recibidos en el puntero void * args

Crear el iterador sobre el fichero batch (new_iterator)

Mientras haya operaciones pendientes en el fichero

        Leer una nueva operación del fichero con el iterador (next_operation)

        Crear una nueva operación con la información devuelta por el fichero (new_operation)

        Encolar la nueva operación en la cola de operaciones (enqueue_operation)

Destruir el iterador (detroy_iterator)
```

- `void* operation_executer(void * args)` implementa la funcionalidad del hilo procesador de operaciones (control de concurrencia incluida). Recibe por parámetro una estructura de tipo `exec_info` que contiene:

- `int *exit:` puntero al flag de terminación de la aplicación. Cuando el *flag* pase a valor `true` el hilo procesará las peticiones pendientes en la cola y terminará su ejecución.

- `stock_market * market`: puntero al mercado sobre el que el procesador de operaciones realizará las operaciones bursátiles.

- `pthread_mutex_t *exit_mutex`: puntero al *mutex* que protege la variable `exit`. Se deberá acceder a la variable `exit` variable siempre con el *mutex* bloqueado.

Pseudocódigo de la función (control de concurrencia a diseñar por el alumno):

```
Extraer la información de los datos recibidos en el puntero void * args

Mientras el flag exit no esté activo

        Desencolar una operación de la cola de operaciones (dequeue_operation)

        Procesar la operación desencolada (process_operation)
```

- `void* stats_reader(void * args)` implementa la funcionalidad de los hilos que consultan el estado del mercado de valores (control de concurrencia incluida). Recibe por parámetro una estructura de tipo `reader_info` que contiene:

  - `int *exit:` puntero al *flag* de terminación de la aplicación. Cuando el *flag* pase a valor `true` el hilo terminará su ejecución.

  - `stock_market * market`: puntero al mercado sobre el que el consultor leerá las estadísticas bursátiles.

  - `pthread_mutex_t *exit_mutex`: puntero al *mutex* que protege la variable `exit`. Se deberá acceder a esta variable siempre con el *mutex* bloqueado.

  - `unsigned int frequency`: tiempo que el hilo debe dormirse después de cada consulta (determina la frecuencia de muestreo del mercado).

Pseudocódigo de la función (control de concurrencia a diseñar por el alumno):

```
Extraer la información de los datos recibidos en el puntero void *
args

Mientras el flag exit no esté activo

        Consultar las estadísticas del mercado de valores
        (print_market_status)

        Dormir hasta la siguiente ronda de información
        (usleep(frequency))
```

1.1.2. Requisitos de concurrencia

Para el correcto funcionamiento del mercado de valores existen ciertas operaciones que no pueden ejecutarse al mismo tiempo por hacer uso de recursos compartidos. Estas operaciones son:

- **Operaciones de escritura:** tanto las operaciones de los *broker* como del procesador de operaciones (*operation_executer*) se considerarán operaciones de escritura sobre el sistema, por lo que sólo uno podrá estar en ejecución en un instante de tiempo
    - Sólo habrá un hilo que ejecute el procesado de operaciones. Cada operación se ejecutará de forma aislada en el sistema (ningún broker, ni lector pueden operar a la vez).
    - Podrá haber múltiples hilos que inserten operaciones en el sistema (*brokers*), pero cada operación se ejecutará de forma aislada en el sistema (ningún otro broker, ni operation_executer, ni lector pueden operar a la vez).
- **Inserción/procesado de operaciones**: los hilos que insertan operaciones en la cola deben pararse cuando esté llena y restaurar su ejecución cuando sea posible (cola no llena). Del mismo modo, el procesador de operaciones debe pararse cuando la cola esté vacía y restaurar su ejecución cuando sea posible (cola no vacía). **No se permite que se muestre ningún mensaje de error durante la ejecución del programa.**
- **Operaciones de lectura:** podrá haber n hilos lectores en ejecución en el sistema en cualquier instante de tiempo y las operaciones de lectura podrán ser concurrentes. Durante el procesado de una operación de lectura, no se permitirá ninguna operación de escritura desde el inicio hasta el final de la lectura, pero sí de otras operaciones de lectura.

NO SE PERMITE LA UTILIZACIÓN DE SEMÁFOROS COMO MECANSIMO DE CONTROL DE CONCURRENCIA.

Las bibliotecas que se proveen incluyen mensajes de error para facilitar la depuración. Un programa correcto no debe mostrar ninguna traza que incluya la palabra **ERROR**, pudiendo ser penalizado en caso de que suceda. El alumno **NO DEBE INCLUIR NINGUNA TRAZA** adicional a las incluidas en el código base.

1.1.3. Detalle de los ficheros de configuración

Para la correcta ejecución de la aplicación, existen dos tipos de ficheros que facilitarán la introducción de información.

Fichero de descripción del mercado de valores. Contiene la información de todas las empresas cotizando en un mercado de valores. Dentro del código base se incluye un fichero de ejemplo llamado `stocks.txt`.

El formato del fichero incluye una empresa y su información bursátil en cada nueva línea del fichero. Cada línea incluye la siguiente información con este formato (los separadores serán siempre espacios y los valores serán siempre números enteros):

<ID> <name> <n°_de_acciones> <precio_por_acción>

- `<ID>`: identificador unívoco de la empresa hasta 10 caracteres.
- `<name>`: nombre completo de la empresa hasta 255 caracteres.
- `<n°_de_acciones>`: cantidad inicial de acciones de la empresa.
- `<precio_por_acción>`: valor de inicio de cada acción de la empresa (el valor total de la empresa será: `<n°_de_acciones>*<precio_por_acción>`).

Fichero de descripción de operaciones en *batch*. Contiene las operaciones a realizar por un *broker*. Dentro del código base se incluye un fichero de ejemplo llamado `batch_operations.txt`.

El formato del fichero incluye la información de cada operación en una nueva línea del fichero. Cada línea incluye la siguiente información con este formato (los separadores serán siempre espacios y los valores serán siempre números enteros):

<ID> <tipo> <n°_de_acciones> <precio_por_acción>

- `<ID>`: identificador unívoco de la empresa sobre la que se quiere aplicar la operación (hasta 10 caracteres). La empresa debe existir en el mercado de valores sobre el que trabaja el *broker*.
- `<tipo>`: tipo de la operación: compra o venta. 0 se corresponde con el valor de la constante BUY del fichero `lib/stock_market_lib.h` (operación de compra de acciones) y 1 se corresponde con la constante SELL (operación de venta de acciones).

- `<n°_de_acciones>`: cantidad de acciones de la empresa que se quieren comprar o vender.
- `<precio_por_acción>`: precio que se va a pagar por cada una de las acciones que se compren/vendan.

1.1.4. Implementación del programa principal y ejecución

Los programas principales se deben implementar en `concurrent_market.c` para que sean compilados por el `Makefile` sin requerir ninguna modificación. Si se quieren tener más programas principales, se deberá modificar el `Makefile` para que sean compilados (sólo para la realización de las pruebas). No obstante, para garantizar una correcta entrega, se deberá adjuntar únicamente el Makefile original.

El programa principal debe incluir al menos:

- Incluir la biblioteca `concurrency_layer` implementada por el alumno
  - Ejemplo:
    - `#include "include/concurrency_layer.h"`
- Inicialización del mercado de valores mediante un fichero de descripción del mercado de valores, que incluya la información relativa a las acciones que forman parte del mercado.
  - Ejemplo:
    - `init_market(&market, "stocks.txt");`
- Inicialización de los mecanismos de concurrencia antes de crear ningún hilo.
  - Ejemplo:
    - `init_concurrency_mechanisms();`
- Creación de los hilos que interactúen con el mercado de valores. Al menos deberá haber un hilo `broker` que introduzca nuevas operaciones en el sistema y un hilo `operation_executer` que las procese.
  - Ejemplo:
    - `pthread_create(&(tid[0]), NULL, &broker, (void*) &info_b1);`
    - `pthread_create(&(tid[1]),NULL, &operation_executer, (void*) &info_ex1);`
    - NOTA: las variables `info_b1` e `info_ex1` son instancias de las estructuras `broker_info` y `exec_info` respectivamente, debidamente inicializadas.

- `join` de los hilos *broker*. Se espera en primer lugar por la finalización de TODOS los hilos *broker*.
    - Ejemplo:
        - `pthread_join(tid[0],&res);`
- Activación del *flag* `exit`.
- Join de TODOS los demás hilos creados (`operation_executer` y `stats_reader`).
- Destrucción de los mecanismos de concurrencia y del mercado.
    - Ejemplo:
        - `destroy_concurrency_mechanisms();`
        - `delete_market(&market_madrid);`

- OPCIONAL: imprimir el estado final del mercado de valores para comprobar que todas las operaciones se han realizado correctamente (antes de la destrucción del mercado).
    - Ejemplo:
        - `print_market_status(&market);`
    - NOTA: un estado final correcto no asegura que la práctica sea 100% correcta, se debe asegurar el correcto funcionamiento de todos los hilos en cada parte de la ejecución. No obstante, puede resultar de utilidad.

Se puede encontrar un ejemplo de programa principal básico en el fichero `concurrent_market.c` que hace uso de los ficheros de configuración `stocks.txt` (descripción del mercado de valores) y `batch_operations.txt` (descripción de operaciones sobre el mercado para un *broker*). Este fichero puede servir como base para implementar otros más avanzados.

Para ejecutar el programa se debe compilar y lanzar:

```
make
./concurrent_market
```

1.2. Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero p3_concurrencia_2016.tar.gz que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

tar zxvf p3_concurrencia_2019.tar.gz

Al extraer su contenido, se crea el directorio ssoo_p3_stock/, donde se debe desarrollar la práctica. Dentro de este directorio se habrán incluido los siguientes ficheros:

Makefile
Fichero fuente para la herramienta make. Se puede modificar para la realización de pruebas, pero debe adjuntarse el original en la entrega. Con él se consigue la re-compilación automática sólo de los ficheros fuente que se modifiquen.

concurrent_market.c

Fichero de ejemplo de programa principal. SE PUEDE MODIFICAR.
stocks.txt
Fichero de descripción de las empresas y acciones de un mercado de valores. SE PUEDE MODIFICAR.
batch_operations.txt
Fichero de ejemplo de operaciones en *batch*. Un hilo de *broker* debe añadirlas al sistema. SE DEBE MODIFICAR.
include/concurrency_layer.h
Fichero de cabeceras con las definiciones de las funciones y estructuras de datos. El alumno debe implementar las funciones descritas en este fichero. **NO SE PUEDE MODIFICAR.**
include/operations_queue.h
Fichero de cabecera que contiene las funciones que se pueden utilizar para interactuar con la cola de operaciones. No se puede modificar.
include/parser.h
Fichero de cabecera que contiene funciones auxiliares para el tratamiento de datos (iterador sobre el fichero de operaciones e impresión de estadísticas). No se puede modificar.
include/stock_market_lib.h
Fichero de cabecera que contiene las funciones que se pueden utilizar para interactuar con el mercado de valores. No se puede modificar.
lib/concurrency_layer.c
Este fichero deberá contener la implementación de las funciones que utilizarán los hilos de la aplicación. Implementa el fichero include/*.h del mismo nombre. En este fichero el alumno debe implementar su práctica. **SE DEBE MODIFICAR.**
lib/operations_queue.c

Fichero de código que contiene la implementación de las funciones descritas en el fichero include/*.h del mismo nombre. No se puede modificar.

lib/parser.c

Fichero de código que contiene la implementación de las funciones descritas en el fichero include/*.h del mismo nombre. No se puede modificar.

lib/stock_market_lib.c

Fichero de código que contiene la implementación de las funciones descritas en el fichero include/*.h del mismo nombre. No se puede modificar.

1.3. Formato de salida

No se permitirá ninguna salida por pantalla adicional a la proporcionada en el código base.

2. Entrega

2.1. Plazo de entrega

La fecha límite de entrega es el domingo 28 de Abril de 2019 a las 23.55.

2.2. 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_AAAA_BBBB_CCCC.zip** donde A…A, B…B y C...C son los NIAs de los integrantes del grupo. El archivo debe contener:

Makefile
concurrent_market.c
stocks.txt
batch_operations.txt
include/concurrency_layer.h
include/operations_queue.h
include/parser.h
include/stock_market_lib.h
lib/concurrency_layer.c
lib/operations_queue.c
lib/parser.c
lib/stock_market_lib.c
authors.txt

authors.txt contiene el nombre de cada alumno perteneciente al grupo y su NIA en dos líneas distintas. El NIA se separará del nombre completo por un tabulador. Ejemplo:

Nombre Apellido1 Apellido2<tab>NIA
Nombre Apellido1 Apellido2<tab>NIA
Nombre Apellido1 Apellido2<tab>NIA

La memoria se entregará en formato PDF en un fichero llamado **ssoo_p3_AAAA_BBBB_CCCC.pdf** que será entregado en el enlace a Turnitin correspondiente. Solo se corregirán y calificarán memorias en formato pdf. Tendrá que contener al menos los siguientes apartados:

- **Descripción del código** detallando las principales funciones implementadas. Es especialmente importante detallar el diseño e implementación de los mecanismos de control de concurrencia. NO incluir 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 de 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.
- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuará 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 índice de contenidos.
- La memoria debe tener números de página en todas las páginas (menos la portada).
- El texto de la memoria debe estar justificado.

El archivo pdf se entregará en el entregador correspondiente la memoria de la práctica (entregador TURNITIN) y tendrá una extensión máxima de 8 páginas.

NOTA: La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

3. Normas

Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.

Los programas deben compilar sin warnings. De lo contrario, se penalizará la nota de la práctica.

Todo código entregado deberá contener comentarios explicativos, en caso contrario, la calificación podrá verse afectada.

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 salvo casus extraordinarias.

El código deberá funcionar en las instalaciones del departamento de informática (laboratorios y/o guernika) en la plataforma de linux. Es responsabilidad del alumno asegurarse de que el código entregado funciona correctamente en dichas instalaciones.

Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.

Se debe realizar un control de errores en cada uno de los programas.

Una práctica que no cumpla con los requisitos de formato recibirá una penalización en la nota.

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.

Los programas entregados que no sigan estas normas no se considerarán aprobados.

4. Anexo (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 pthread_create' o ***'man 3 pthread_create'***

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 del 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. 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`)