SOAD-Laboratorio 3

Para este laboratorio se utilizará de nuevo el código de la librería libfiber. Continuad con el código de la librería con la que habéis trabajado en el Lab-2.

Queremos añadir a la librería una planificación de threads basada en prioridades fijas y cesión voluntaria del procesador. Los algoritmos propuestos tendrán una gestión muy sencilla, basada en colas ordenadas.

Las llamadas a librería relacionadas serán:

```
int    sched_nice(int pri, int th_id);
int    sched_yield(void);
int    sched_handoff(int th_id);
```

El funcionamiento pedido es el siguiente:

- El thread master (creador del resto de threads) puede modificar su prioridad o la de cualquier otro thread de la aplicación con la llamada sched_nice. Cada vez que se modifica la prioridad de un thread, se reordena la cola, de modo que el thread más prioritario (valor númerico más bajo, es decir, 0 es la prioridad mayor) sea el primero y el menos prioritario el último.
- Cualquier thread puede realizar la llamada a la función sched_yield para ceder el procesador al thread más prioritario (si es él mismo, no se hará cambio de contexto).
- Cualquier thread puede pasar directamente el procesador a un thread determinado, aunque sea menos prioritario, a partir de la llamada sched_handoff.

Implementación de la llamada sched_nice

El prototipo de la llamada es:

```
int sched_nice(int pri, int th_id);
```

`pri:` nuevo valor de prioridad para el thread indicado; si el valor es cero, devuelve la prioridad actual y no hace nada.

`th_id:` identificador de thread al que se le va a modificar la prioridad; si el valor es cero, el thread es el mismo.

retorna: la prioridad anterior si ha ido bien y -1 si ha habido error (determinad qué tipos de error pueden darse).

Implementación de la llamada sched_yield

La cabecera de la llamada es:

```
int sched_yield(void);
```

retorna: el identificador del thread al que le ha pasado el control, cero en caso de que no haya hecho nada (sigue la política de prioridades, por tanto, si el primer thread de la cola tiene una prioridad menor, no le pasará el control), -1 si ha habido error (determinad qué tipos de error pueden darse).

Nota: comprobad si la librería tiene una función con este nombre o con esta funcionalidad (o parecido): comprobad si simplemente hay que hacer un wrapper (es decir, las dos funciones hacen lo mismo, con nombre distinto: entonces una llama directamente a la otra), o si hay que hacer además alguna adaptación, o si tienen dos funcionalidades distintas. Decidid cómo mantenéis la compatibilidad con código que ya la utilice, en caso de que sea necesario mantener dos versiones de funcionamiento distinto, si se da el caso.

Implementación de la llamada sched_handoff

La cabecera de la llamada es:

```
int sched_handoff(int th_id);
```

retorna: 0 si ha ido bien y -1 si ha habido error (determinad qué tipos de error pueden darse).

Haced varios **juegos de pruebas** que permitan comprobar el buen funcionamiento de las llamadas.

Diseñad un caso de uso en el que sea útil disponer de estas llamadas para mejorar el rendimiento de la aplicación.

Entrega

Con todo lo que trabajéis, haced un documento de 4-6 páginas y enviad en un tar.gz todos los ficheros relacionados: el documento, el código realizado (las funciones implementadas, los juegos de pruebas) y la explicación del caso de uso, con su código y las salidas esperadas.

La entrega ha de ser completa (todos los ficheros relacionados que se comentan en el párrafo anterior) para considerarse aceptada.

Bibliografía y referencias

Gestión de threads en Java:

GC: Thread - java.lang.Thread (.java) - GrepCode Class Source:
http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/lang/Thread.java

Gestión de threads de usuario

Implementing a Thread Library on Linux :

http://www.evanjones.ca/software/threading.html