20 de Noviembre de 2021

San Sebastián, España

UPV - EHU, Ingeniería Informática

Parte 1 y 2
Simulador de un Kernel

Asier Cruz

Jorge Iglesias

Índice

1. Introducción

El proyecto desarrollado consiste en la simulación del funcionamiento del Kernel de un sistema operativo. La implementación del proyecto ha sido realizada en C en el entorno de desarrollo de Visual Studio y utilizando como sistema de resguardo y de versiones GitHub.

Las especificaciones sobre el sistema a desarrollar, son dadas por los docentes y en base a ellas ha sido construido este proyecto. Se pide desarrollar las estructuras del Kernel en las que encontramos CPUs, cores, hilos, PCBs y una cola de procesos. Adicionalmente se han añadido otro tipo de estructuras para el correcto funcionamiento de los distintos componentes. Además se debía implementar ciertos entes que simulen el funcionamiento del sistema como son el reloj(clock), temporizador(timer), generador de procesos (process generator) y planificador (scheduler).

2. Manual de uso

Las configuraciones del sistema desarrollado, se encuentran en el inicio del archivo main.c en el que se pueden modificar tal y como quiera el usuario. En este lugar no se encuentran ni el número ni el número de cores ni de hilos, los cuales se deben introducir como parámetros al programa. Siendo el número de cores el primer argumento y el número de hilos el segundo. Por otro lado cabe destacar que el número de CPUs únicamente puede ser uno. Se presenta un ejemplo de como compilar y ejecutar el proyecto (en Linux):

```
gcc -o main.exe *.c -pthread
./main 2 3
```

Al ejecutarlo únicamente se mostrará el contenido actual del Kernel, el programa solo finalizará de dos maneras, por interrupción del usuario o por que se haya excedido el tiempo máximo. A continuación se muestra la situación de la ejecución en un golpe de reloj.

```
 **** CLOCK DE RELOJ ****
------------------------------------------------------------------
-->CORE 0
---->HILO 00
        |--> PCB4 --> Tiempo de vida = 6, Tiempo restante = 5, Quantum = 22 Prioridad =
0
---->HILO 01
        |--> PCB2 --> Tiempo de vida = 7, Tiempo restante = 2, Quantum = 2 Prioridad = 4
---->HILO 02
        |--> PCB3 --> Tiempo de vida = 6, Tiempo restante = 2, Quantum = 23 Prioridad =
0
-->CORE 1
---->HILO 10
---->HILO 11
---->HILO 12
```

3. Exclusiones

Debido a problemas a la hora de desarrollar el sistema y por falta de entendimiento y comunicación de los integrantes del grupo se ha desestimado la opción de tener varias CPUs dentro de la simulación, dejando esto para próximas entregas.

Lo siguiente no resulta esencialmente una exclusión ya que en los requisitos del sistema no se contempla somos conscientes que cada cola de procesos, la cual creamos como array almacenan un tamaño desmesurado en memoria. Los integrantes del grupo son conscientes de que sería infinitamente más eficiente la creación de una estructura dinámica de colas, reservando así únicamente el espacio a utilizar. Esto ha sido excluido debido a la falta de manejo con C y los problemas aparecidos a la hora de implementarlo.

4. Estructura del proyecto

La estructura del proyecto, cuenta con la máquina que ejecuta los procesos, la CPU, esta a su vez contiene el número de cores, y cada core el número de hilos. Siendo los cores y los hilos dados por el usuario mediantes parámetros de entrada.

La definición de estos entes es la siguiente:

```
typedef struct{
    int id;
    int core_id;
    int estado; //1 ocupado 0 libre
    PCB_T *pcb;
}thread_T;

typedef struct{
    int id;
    int nHilos;
    thread_T *hilos;
}core_T;

typedef struct{
    int id;
    int nCores;
    core_T *cores;
}CPU_T;
```

Cada core e hilo cuenta con su identificador propio. En el caso de la CPU también ocurre esto, lo que al parecer carece de sentido alguno, pero esto se realiza con el objetivo de que en próximas etapas del proyecto se implemente la funcionalidad de poder crear un sistema con varias CPUs.

Los thread_T cuentan a su vez con una variable de estado que indica si este se encuentra ocupado o libre para poder dilucidar si es posible asignarle un PCB_T o no. Cada thread_T cuenta a su vez con un apuntador a un PCB_T el cual es el proceso a ejecutar por cada hilo y posee la siguiente estructura.

```
typedef struct {
    int id;
    int t_vida;
    int t_restante;
    int quantum;
    int state; //0 listo y 1 ocupado
    int prioridad;
} PCB_T;
```

El PCB_T tiene varios atributos, su id (al igual que los cores y los hilos para poder identificarlo unívocamente), el tiempo de vida total, el tiempo de vida restante hasta finalizar (el cual se irá reduciendo con cada ciclo del clock), su quantum, el estado y la prioridad. Estando esta última entre 0 y el valor indicado por el usuario en las opciones de configuración del main.

Sumado a ello para poder gestionar los PCBs de manera global se ha creado una cola (LIFO) donde se encuentran los hilos disponibles en ese instante, asignando y desasignando los PCBs correspondientes.

```
typedef struct{
    thread_T **threads;
    int size;
}colaCPU_T;
```

```
//Crea la cola de hilos libres de la cpu
Cola_CPU_T * crearCPUcola(CPU_T * cpu){...}

//Añadimos un pcb en la cola para ocupar el hilo en esa posición y lo
sacamos de la cola
int addPCB_CPU(CPU_T * cpu, colaCPU_T * cola, PCB_T * pcb){...}

//Quitamos el PCB del hilo, encolamos el hilo libre a la cola de hilos,
y //el pcb a la cola de procesos
int rmPCB_CPU(CPU_T * cpu, Cola_CPU_T * cola, colaPrioridades_T * {...}
```

Aquí es donde se encuentra el momento donde, se actualizan los tiempos de vida de los PCBs en ejecución, cuando se terminan dichos procesos (consultando tiempo de vida) se eliminan y se encola el hilo libre a la cola CPU.

```
int actualizarTiempo(CPU_T * cpu, Cola_CPU_T * cola){...}
```

Por otro lado, ha sido creada una cola de prioridades (colaPrioridades_T) con el objetivo de gestionar los procesos mediante la siguiente estructura:

```
typedef struct{
    int num_procesos;
    int size;
    PCB_T* pcbs;
    int prioridad;
    int pos_actual;
    int ultimo;
} colaProcesos_T;

typedef struct{
    colaProcesos_T * procesos;
    int size;
} colaPrioridades_T;
```

El tipo cola_Prioridades, está conformado por otra estructura colaProcesos_T además de la prioridad que a su vez también es el número de las colas de procesos que se alojan en esta. En colaProcesos_T se encuentran los procesos (cada uno con la prioridad que se le ha sido asignada), el tamaño de la cola, el número de procesos, la posición actual en la misma, y la última posición.

```
colaPrioridades_T * crearEstructuraPrioridades(int num_procesos, int
prioridad){...}
int encolarProceso(colaPrioridades_T * cola , PCB_T * pcb){...}
PCB_T * desencolar proceso(colaPrioridades_T * cola){...}
```

Asimismo, el sistema cuenta con un generador de procesos, un clock un timer y un scheduler como fue requerido para la realización de esta práctica, su estructura es la siguiente.

```
typedef struct{
    int id_gen;
    pthread_t pthid;
} generadorProcesos_T;

typedef struct {
    int id_clock;
    pthread_t pthid;
} clock_T;

typedef struct{
    int id_temp;
    pthread_t pthid;
} temporizador_T;

typedef struct scheduler{
    int id_sch;
    pthread_t pthid;
} scheduler_T;
```

Las últimas estructuras presentadas, hacen referencia a las entidades que se deben crear para el funcionamiento del sistema. Cada uno de ellos se ejecuta en un hilo distinto pero todos a la vez logrando de esta manera la simulación de un Kernel con acciones en tiempo real. Todos ellos son creados y ejecutados desde el método main a modo de funciones vacías.

```
void * generadorProcesos(void *id_gen) {...}
void * procesoPlanificador(void *s) {...}
void * procesoReloj(void *id_clock) {...}
void * procesoTemporizador(void *id_temp) {...}
```

Todos ellos acceden concurrentemente a secciones críticas, que solo pueden ser accedidas una única vez por cada recurso, para la gestión de secciones críticas se han utilizado semáforos. Adicionalmente ha sido implementado el método de petición ordenada con el objetivo de eludir la espera circular, eliminando así una de las condiciones de Coffman y por ende sorteando el problema del interbloqueo.

5. Problemas encontrados

En primera instancia nos dimos de bruces contra el C, debido a la falta de costumbre de trabajar a un nivel inferior del que estamos acostumbrados, esto se debe principalmente a que ambos integrantes del grupo pertenecen a la rama de Software.

Supuso un gran problema el trabajar con los includes, ya que con los lenguajes trabajados anteriormente era algo que funcionaba por defecto. Pero en este caso resulta un ámbito nuevo en el que teníamos que trabajar. La solución a este simplemente se encontró en la práctica y la consulta de bibliografía.

Finalmente justo antes de acabar el proyecto, nos surgieron problemas al reservar espacio en memoria. Teniéndolos uno de ellos varios días en el dique seco, que gracias a la ayuda de uno de los docentes, el cambio de sistema operativo (de Windows a Linux) y sin tener muy claro porque se acabó solucionando.

6. Conclusiones

Principalmente ambos integrantes nos vimos forzados a escoger esta asignatura por temas administrativos, los cuales consideramos que deberían ser distintos y que el reparto de plazas fuera algo más arbitrario. Además de que es una entre tantas de nuestras asignaturas optativas que no contemplamos ya que no se adapta a nuestras competencias ni hacia dónde queremos dirigir nuestra carrera profesional.

Pero por otro lado estamos disfrutando ampliamente cursarla debido al buen ambiente promovido tanto por los docentes como de nuestros compañeros de clase, siendo esta la “hora feliz del día”. Además de la plena predisposición de los docentes a la hora de echar una mano o de volver a explicar algo que no se ha entendido.

En cuanto a la realización del proyecto nos ha permitido recopilar conceptos impartidos en años anteriores sobre C, los cuales creemos que se quedan bastante lejos de nuestras expectativas. Haciendo desde aquí un llamamiento para la reestructuración y una mejor enseñanza de este lenguaje en cursos inferiores.

El desarrollo de este proyecto nos ha resultado realmente interesante (echando en falta algo de visión gráfica) el conocer de primera mano cómo funcionan las tripas de un sistema operativo, que a pesar de ser difícil de comprender de primeras e incluso a veces desesperante, resulta muy gratificante cuando por fin se encuentra la luz y nos enfrenta a un entorno nuevo.

7. Bibliografía

OS - Tema 2-6 Linux Scheduler.pdf

OS - Tema 2-5 Planificación - Políticas.pdf

OS - Tema 2-4 Planificación - Indicadores.pdf

OS - Tema 2-2 Sincronización.pdf

OS - Tema 2-3 Interbloqueo.pdf

PThread Mutex Lock

How to use semaphores in C language

Uso de Colas

Concepto de puntero vacío

OS - Tema 2-7 pthread.pdf (Parte 1)

OS - Tema 2-7 pthread 2.pdf (Parte 2)