**Valentino Herrera Quintana**

**Documentación: Simulador de un Kernel**

**Introducción**

En este informa se va a detallar el diseño e implementación de un simulador de Kernel de un sistema operativo. El simulador es un programa multihilo implementado en lenguaje C, que utiliza bibliotecas como (**clock.h, timer.h, proc\_generator.h, structures.h y scheduler.h**) implementadas por mí mismo, con el fin de manejar los hilos y la sincronización de forma más dinámica y fácil. Junto a esto también se hizo uso de mutex de condición y sincronización, primordiales para la parada y reanudación de las diversas estructuras implementadas.

Con este documento cubriremos la explicación más relevante de la arquitectura, sus componentes y las políticas de planificación empleados.

**Fase 1**

La primera fase del proyecto se basó en la implementación de la estructura básica del simulador, en la que se inicializaron los componentes fundamentales del mismo:

* Clock: responsable de generar ciclos de reloj avanzando el tiempo del sistema. Con cada ciclo, el clock mueve la máquina simulada que incluye una serie de CPUs, núcleos e hilos hardware.

Con la función *clock\_function()* generamos un ciclo por cada segundo y notificamos al resto de hilos mediante una condición.

* Timer: responsable de generar interrupciones de forma periódica en base a los ciclos del clock generados.

Con la función *timer\_function()* escuchamos los ciclos del reloj y generamos ticks para activar el Scheduler.

* Process Generator: responsable de generar procesos con identificadores únicos, tiempos de vida aleatorios y prioridades asignadas aleatoriamente.

Con la función *proc\_generator\_function()* generamos procesos en cada tick generado del timer con PID únicos, tiempos de vida y prioridades aleatorias.

* Scheduler/Dispatcher: en la primera fase solo es responsable de activarse con las interrupciones generadas por el timer, pero que como veremos en la segunda fase será el principal objetivo para cumplir.

La estructura de datos estará conformada por una serie de estructuras definidas en el **structures.c**:

* ProcessQueue: responsable de guardar la cola de procesos que se van generando por el Process Generator.

Con la función *init\_process\_queue()* inicializamos la cola vacía y *add\_process()* agrega los procesos a la cola, verificando su capacidad.

* Machine: responsable de configurar la maquina con los CPUs, núcleos e hilos pasados por parámetro.

Con la función *init\_machine()* reservamos memoria dinámicamente para esta estructura.

Para la implementación de esta primera fase, tan solo debemos inicializar las estructuras anteriormente definidas, configuraremos los parámetros para el funcionamiento de la máquina y añadiremos mutex/variables de condición para coordinar el acceso a los recursos de forma compartida.

**Fase 2**

Implementaremos de una forma más sofisticada el Scheduler/Dispatcher y realizaremos algunos cambios en las implementaciones del resto de programas con la intención de que funcionen de forma correcta. Las políticas elegidas serán Round Robin y Prioridades, que se ejecutarán ambas en la misma cola de procesos, pero serán tratadas de distinta forma dependiendo de la política elegida a la hora de iniciar la máquina.

* Round-Robin: es un algoritmo de planificación utilizado para distribuir el tiempo de la CPU de manera justa entre todas las tareas pendientes.

Con la función *scheduler\_round\_robin()* seleccionamos aquellos procesos que estén en estado READY para poder ejecutarlos durante un periodo de tiempo específico llamado quantum.

Con la función *dispatch\_process\_rr()* asignamos un hilo para cada proceso que se vaya a ejecutar y actualizamos el estado de dicho proceso, en función del tiempo de restante de vida que le quede.

* Prioridades: es un algoritmo de planificación que determina cual va a ser el orden en el que se van a ejecutar los procesos.

Con la función *scheduler\_priority()* ejecutamos aquellos procesos que se encuentren en estado READY y que además cuenten con la prioridad más alta.

Con la función *dispatch\_process\_priority()* asignamos un hilo para cada proceso hasta que este haya finalizado.

Para la implementación de esta segunda fase, hemos realizado varios cambios significativos en el **scheduler.c** y su respectiva biblioteca, pero también han sido de gran relevancia los cambios en el resto de los programas. Reflejados todos en el **main.c**, uno de los principales cambios que se ha añadido y que no viene reflejado como requisito en la implementación del proyecto ha sido la configuración total del Kernel por parte del usuario.

Se solicita al usuario que indique el número de CPUs, núcleos por CPU e hilos por núcleo para poder formar la estructura de la máquina. Además, deja a elección de la persona que pruebe el sistema la frecuencia con la que se generan los ticks del timer y con ello las llamadas al scheduler/dispatcher así como la frecuencia para la generación de procesos o el límite de la cola de dichos procesos creados.

Por otro lado, al implementarse dos políticas distintas, también se deja a elección de la persona que interactúa con la máquina de elegir, que tipo de planificación desea experimentar.

**Conclusión**

El simulador implementado nos permite explorar conceptos de planificación de procesos, sincronización y manejo de recursos en un entorno multihilo. Además, la flexibilidad para configurar la arquitectura y seleccionar políticas nos proporciona un entorno versátil para estudiar el comportamiento del sistema operativo.