eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática
Grado de Ingeniería Informática

Informe Técnico del trabajo práctico

Sistemas Operativos
Gestión de procesos

Informe del simulador de un sistema operativo

Borja Moralejo Tobajas

12 de Noviembre 2021

Tabla de contenidos

Abstracto

El objetivo de este proyecto es aplicar todos los conocimientos aprendidos en clase para la elaboración de un simulador de un sistema operativo. Este simulador debe de ser lo más cercano a la realidad para observar los efectos de diferentes políticas en la máquina y medir los efectos en distintas arquitecturas de máquinas. Para ello se utiliza ANSI C y la biblioteca de pthread, una herramienta que permite la creación de hilos comunicados y facilita unas herramientas para sincronizar estos hilos mediante mutex, semáforos o variables condicionales.

Otro objetivo es aprender, no en demasiada profundidad, el funcionamiento interno del sistema operativo de Linux mediante la mimetización de su implementación y crear estructuras de datos que en otros lenguajes de programación, como Java, ya están implementadas.

Fundamentos teóricos

Para empezar a programar un sistema operativo primero tenemos que saber qué es un sistema operativo.
Un sistema operativo es un conjunto de software que gestiona de forma segura y eficiente los recursos de la máquina, además de crear una abstracción de la máquina en la que está instalada.
Los recursos que debe de gestionar son los procesadores, la memoria, los dispositivos y los ficheros y para cada recurso existen diversas políticas y mecanismos donde la elección de dichas se basa en la relación de rendimiento y eficiencia. Para medir esas cualidades existen los siguientes indicadores: Eficiencia temporal, mide la tasa de uso efectivo del recurso en el tiempo; capacidad del almacenamiento, cantidad de almacenamiento que se puede representar; eficiencia espacial, tasa de aprovechamiento espacial del recurso de almacenamiento.

Procesos y elementos de comunicación y sincronización.

Los procesadores son uno de los recursos que se encarga de gestionar un sistema operativo y son los que computan los procesos. Un proceso, proceso pesado o tarea es una unidad básica de asignación de recursos. Mientras que un proceso ligero, también conocido como thread, o hilo, es la unidad básica de ejecución.

Cada proceso no tiene conocimiento de los demás y no están diseñados para trabajar conjuntamente. Pocas veces las tareas de un sistema son independientes unas de otras y utilizan recursos que comparten. Para lograr un funcionamiento adecuado y eficiente los procesos necesitan comunicarse y sincronizarse utilizando los siguientes elementos: locks, mutex, semáforos o variables condicionales.

A la hora de comunicarse se tiene que tener en cuenta las condiciones de carrera y mantener un acceso exclusivo a las secciones compartidas de memorias. Por lo tanto se debe garantizar el acceso exclusivo pero con esto se crean dos problemas, el interbloqueo y la inanición.

Primera parte del proyecto

El objetivo principal del trabajo es desarrollar un simulador de un sistema operativo. El programa tiene las siguientes fases en esta primera parte:

- Recibir como paŕametros las opciones de configuración.
- Inicializar las estructuras de datos que se vayan a utilizar.
- Lanzar los hilos que implementarán los distintos subsistemas.
- Realizar la comunicación utilizando memoria compartida y los elementos de sincronización vistos en clase.

Descripción de los elementos iniciales del sistema:
Cada thread controlará uno de los siguientes elementos:

- Clock: Simula el reloj de los procesadores, es decir, cada vez que se ejecuta significa un ciclo de ejecución de cada hilo hardware. Por un lado, deberá hacer avanzar toda la estructura de CPUs, cores e hilos hardware y, por otro, deberá señalar al Timer (o timers) para que este funcione.
- Timer: Se encarga de generar una señal cada cierto periodo de tiempo. Recibe los pulsos del Clock y con la frecuencia que se haya decidido produce una interrupción del temporizador, un tick.
- Process Generator: Genera procesos (PCBs) de manera aleatoria con una frecuencia variable.
- Scheduler/Dispatcher: Se encarga de planificar y de realizar los cambios de contextos de los procesos aunque en está primera parte no hará ninguna tarea. Se despertará con cada interrupción del temporizador, es decir, cada tick que produzca el Timer.

La solución desarrollada

Estructura del programa

El programa está estructurado de la siguiente forma, es un breve esquema del método main:

Procesado de parámetros + Control de errores:

En esta parte se procesan los argumentos y se tiene en cuenta lo siguiente:

- Argumentos sin sentido, como n_cpu = 0.
- Argumentos incorrectos, si se introduce texto como argumento donde se espera un número.
- Argumentos con conflictos entre ellos, tratar de utilizar un clock manual y un clock retardado.

Inicialización de estructura de datos:

Se utilizan subrutinas para inicializar la piscina de memoria de los PCBs y la cola de PCBs.

Inicialización de los elementos de sincronización:

Se utilizan las llamadas de biblioteca de pthread,h.

Creación y gestión de hilos:

Se crean los hilos de los diferentes elementos del simulador y se pasan los diferentes elementos de sincronización necesarios. En esta parte también se les pasa algunos argumentos mediante las funciones de pthread_create.
Primero se crea el hilo del clock, luego los de los timers y por último los hilos de los diversos elementos
Al final se incluye un bucle donde se espera la finalización de cada hilo.

Clases y cabeceras
A continuación muestro una tabla con una lista de las clases y cabeceras generadas junto a una breve descripción.

Clases y/o cabeceras	Descripción
clock.c, timer.c, scheduler.c, pgenerator.c	Son las clases de los elementos del simulador, cada una tiene las funciones para llevar a cabo la tarea designada.
defines.h, globals.h	Cabeceras que contienen datos que se van a recurrir por más de una clase.

Clases y/o cabeceras	Descripción
estructuras.c, estructuras.h, queue.c, queue.h	Estas son las clases y cabeceras de las estructuras de datos.
simulador.c	Clase que contiene el método main y las funciones de procesamiento de argumentos.
randomp.c	Clase auxiliar para generar un número entre dos parámetros.

Parametrización de elementos

El simulador permite varios parámetros para poder cambiar valores de los componentes de la máquina o las frecuencias de los diferentes elementos sin tener que recompilar el programa. También está la opción de variar el tamaño de las estructuras de datos y poner el elemento Clock en modo manual para que el usuario pulse una tecla para avanzar un ciclo. Para generar está parametrización se ha utilizado la biblioteca getopt()

La siguiente tabla muestra la parametrización utilizada:.

Argumento	Descripción
-m --clock_manual	Flag de Clock modo manual
-r --clock_retartado	Flag de Clock con periodo configurable en función de freq_clock_ms
- l --freq_clock_ms	Periodo en milisegundos del elemento clock usando flag retardado
-d --freq_disps_sched	Frecuencia de ciclos de reloj para que el timer active el Scheduler/Dispatcher
-g --freq_pgenerator	Frecuencias de ciclos de reloj para que el timer active el Process Generator
-b --ttl_base	Valor mínimo para la generación aleatoria del tiempo de vida de proceso, en ciclos de reloj
-x --ttl_max	Valor máximo para la generación aleatoria del tiempo de vida de proceso, en ciclos de relo
-c --n_cpu	Número de CPUs

Argumento	Descripción
-k --n_core	Número de núcleos por CPU
-t --n_thread	Número de hilos por núcleo
-p --poolsize	Tamaño inicial de memoria de PCBs en elementos
-q --queuesize	Tamaño inicial de cola de PCBs en elementos

Sincronización de procesos

El Clock está sincronizado con los Timers mediante variables condicionales. Con esta sincronización se busca que el clock no avance de ciclo si los timers no han terminado de ejecutar el ciclo.

Para manejar la comunicación entre timers y sus elementos, se ha optado por unas variables condicionales y unos mutex. Las variables condicionales avisan a los elementos que tienen la vía libre para avanzar un ciclo con su ejecución.
Mediante un mutex se limita el acceso a la memoria principal de PCBs y las colas y de esa manera conseguimos que sea atómica para meter y sacar procesos de la cola de procesos. En vez de tener las funciones para bloquear y desbloquear los mutex en los propios elementos, se han metido en las funciones que llaman los elementos y son comunes para todos los que intenten acceder.

Desarrollo de los elementos

Clock: Genera los ciclos que controlan el tiempo en el sistema. A esos ciclos los he llamado pulsos y en cada pulso reduce el ttl de los procesos que estén en cualquiera de los hilos de la máquina.
Timer: Se encarga de generar la señal de interrupción que avisará periódicamente al resto de funciones del sistema. Por cada elemento hay un timer y una frecuencia para la llamada de la función virtual asignada, pero para las siguientes versiones del proyecto estas funciones se verán reemplazadas por una única función generalizada.

Process Generator: prepara los PCBs de los procesos nuevos. Coge un PCB ya generado por la estructura de datos, de la piscina de memoria, y rellena sobreescribe los campos correspondientes al proceso. De momento sólo incluirá el identificador del proceso con un tiempo de vida aleatorio según los argumentos especificados. Al terminar de prepararlo, lo añade a la cola de procesos.

Scheduler/Dispatcher: Atiende a los PCBs que están en la máquina y han terminado(por ahora es su ttl ha llegado a 0 o menor), los saca de la máquina y los devuelve a la piscina de memoria principal. Una vez revisado los hilos en proceso, mira la cola de procesos y va metiendo en los hilos de la máquina los PCBs de la cola hasta que se quede sin elementos en la cola o se llenen todos los hilos de la máquina.

Simulación de proceso en ejecución

El elemento clock manda a la máquina una señal para que vaya reduciendo el contador de tiempo de vida de los PCBs que tenga en los hilos de los procesadores. Después, manda pulsos a los timers. Sí el flag de manual está activado, el usuario deberá pulsar cualquier tecla para hacer un pulso de reloj.

Los timers cada periodo de tiempo que está definido por una frecuencia parametrizada y la frecuencia del clock, dan un tick y activan el elemento que están controlando para volverlos a bloquear y esperar a nuevos pulsos del reloj. Ese elemento que están controlando es el Dispatcher/Scheduler o el ProcessGenerator.

Dispatcher y Scheduler básico: Coloca en los hilos procesos y comprueba si el hilo se ha completado para terminarlo. Ahora mismo está funcionando como el scheduler/dispatcher principal, en las siguientes partes de la práctica este mandará a los schedulers de cada cpu.

ProcessGenerator: Intenta coger un elemento libre de la piscina de PCBs y si encuentra uno, rellena el PID y el TTL según el ttl_base y ttl_max. Una vez preparado, se intenta meter en la cola y si no lo consigue lo devuelve a piscina de PCBs.

Estructura de datos

Colecciones de datos del simulador

-Mempool: Es un manager de memoria dinámica. Almacena estructuras de tipo PCB en un array dinámico. Contiene una Linked List de enteros (lkdList_int_t) para saber qué índices están libres. Tiene un mutex para controlar el acceso atómico a su memoria.

-Linked List de enteros: Es una lista creada con nodos de linked list de enteros. Contiene punteros a la memoria dinámica, número de elementos máximos y número de elementos actuales.

-Nodo de Linked List de enteros: Esta estructura tiene un puntero a un nodo de linked list y el valor del entero del nodo. El elemento n-ésimo conoce al elemento n+1-ésimo en caso de que no sea el último elemento de la lista, en ese caso el siguiente elemento será nulo.

-Process Queue: Estructura que contiene los procesos que están en espera. Es una cola circular de PCBs. Además tiene un mutex para acceder a sus datos.

Estructura de los elementos del simulador

-PCB: Estructura de datos que representa cualquier proceso. Tiene 3 parámetros: un identificador PID, un tiempo de vida del proceso y un puntero al nodo de la lista de la pool de PCBs.

-Machine: CPUs, núcleo e hilos. La CPU tiene uno o más núcleos y un identificador. Cada núcleo tiene uno o más hilos. Los hilos están inicializados en una matriz tridimensional y luego a cada elemento se le asigna su hilo. Esta estructura se ha definido en previsión a las siguientes partes e implementaciones necesarias.

Aspectos que han quedado pendientes

Los objetivos principales de la práctica han sido completados pero hay hueco para la mejora. Hay aspectos que se podrían hacer para añadir una aportación considerable a la aplicación. Una de ellas es la finalización correcta de la aplicación; los hilos hijos del simulador no terminan y la única forma de parar es abortando la aplicación.
En cuanto a las estructuras de datos eliminar de alguna manera el elemento de la cola de la mempool del PCB pero mantener el sistema de "PCB pool" implementada.
Respecto a la sincronización de los elementos, poner la cola como productor consumidor en vez de un mutex reduciría el conflicto entre el/los Dispatchers/Schedulers y el loader que se vaya a meter en las siguientes partes del programa.
Por último, una forma de registrar los datos más cómoda y que no llene la terminal, por ejemplo en un fichero log y que se use el comando watch para ir viendo los cambios o algo por el estilo.

Conclusiones

Objetivos cumplidos
En este proyecto se han aplicado los conocimientos aprendidos en clase. Desarrollar un simulador de un sistema operativo es un proyecto de una magnitud muy grande, demasiada como para ver todo los detalles en una práctica de una asignatura.

Aún así este proyecto ha servido para asentar lo aprendido y aprender cómo funcionan realmente los sistemas operativos a bajo nivel. Me ha ayudado a ver la importancia de los sistemas operativos, Además, no solo deben desarrollarse, deben mantenerse para poder actualizarse o arreglarse mientras se van encontrando problemas o implementando las mejoras.

En cuanto al análisis de las diferentes políticas, es un punto que debo aprender en profundidad pero es una inversión de tiempo muy alta para este proyecto.

Bibliografía

Hay mucha información sobre este campo dado que la tecnología lleva desarrollándose durante años. La mayoría de la información extra que he necesitado la he sacado de los apuntes de la universidad o de las páginas de referencia de las herramientas.

Código fuente de linux	https://github.com/torvalds/linux
Apuntes de SO	Egela
Apuntes de C avanzado	Egela
Manual getopt	https://man7.org/linux/man-pages/man3/getopt.3.html
Referencias pthread	https://hpc-tutorials.llnl.gov/posix/
Referencias Makefile	https://www.gnu.org/software/make/manual/make.html