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

Informe del simulador de un sistema operativo

Borja Moralejo Tobajas

23 de Diciembre 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 el conjunto 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, del conjunto 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 al conjunto 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 del conjunto 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 al conjunto de PCBs.

Estructuras 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.

Segunda parte del proyecto

En esta segunda parte se busca crear una planificación para el simulador del kernel de la primera parte. El programa requiere de estos puntos:

- Scheduler.
- Dispatcher.
- Inicializar las estructuras de datos que utilicen los schedulers y el dispatcher.
- Realizar la comunicación utilizando memoria compartida y los elementos de sincronización vistos en clase.

Estos elementos ya existían desde la primera implementación del proyecto. Por lo tanto, no se tiene que empezar desde cero mas expandir las funcionalidades de algunos elementos. Descripción de los elementos sistema extendidos:

- Scheduler: Junto al dispatcher, antes formaban un único elemento. Aunque trabajen conjuntamente, he decidido hacer esta separación debido a la complejidad que se les ha dado a cada apartado.
  El Scheduler está dividido en dos partes. El Scheduler maestro o master y los Schedulers esclavos o slaves.
  El maestro se encarga de repartir las tareas nuevas generadas por PGenerator entre los esclavos además de tener control sobre cuando pueden activarse los esclavos mediante variables condicionales.
  Los esclavos están asignados a un esclavo por núcleo del simulador. Estos son los que junto a su estructuras de datos de colas de prioridades, reparten las tareas entre los hilos que tenga el núcleo asignado.
- Dispatcher: Se encarga de realizar los cambios de contextos de los procesos y sacar según diferentes condiciones los procesos de los hilos. No son elementos independientes, cada Scheduler esclavo tiene un Dispatcher en su interior por decirlo de alguna manera.
- Process Generator: Se debe extender la funcionalidad de generar procesos para poder ser utilizada con el nuevo Scheduler/Dispatcher

La solución desarrollada

Para cumplir con los objetivos se ha desarrollado un sistema donde existe un Scheduler maestro que reparte las tareas con los Schedulers esclavos, esto es, cada núcleo está regido por un scheduler esclavo y estos a su vez por el Scheduler maestro. Se ha optado por una política de arbitraje multinivel con prioridades dinámicas y una degradación paulatina aumentando el quantum. Por lo tanto, la política de activación es la expulsora por tiempo y se ha escogido por utilizar un quantum dependiente del nivel de prioridad.

Estructura modificada

La estructura inicial de la primera parte se mantiene pero cambian los puntos relacionados con la planificación de los procesos.

Simulador, resumen de cambios: En vez de tener implementado un solo scheduler único que se tenga que encargar de repartir las tareas entre todos los hilos, he optado por una implementación con un scheduler maestro y schedulers esclavos, teniendo por core 1 scheduler esclavo.
Para ello el maestro debe de tener una vía de comunicación con los esclavos. Eso se hace mediante la inicialización del pthread mediante el paso de argumentos, de esta forma el scheduler maestro puede repartir las tareas generadas por PGenerator según la política de reparto. Esas políticas pueden ser Round Robin y por afinidad de los procesos.

Scheduler/Dispatcher: Este elemento ya no existe como una sola unidad, está dividido en Dipatcher y Scheduler, y este a su vez en Scheduler maestro y esclavo.

Scheduler Maestro: Dispone de una cola de procesos nuevos con la cual se reparten entre los schedulers esclavos. Existen dos políticas de reparto: round robin, va ciclando entre los schedulers uno a uno para repartir de forma equilibrada y por afinidad, con el atributo del PCB de afinidad se asignan directamente a los schedulers esclavos con la misma ID.

Scheduler Esclavo: En el método de retirar PCB nuevo se introduce en la cola de prioridades expirada para ser introducido por el reschedule cuando le toque. Después se encuentra el dispatcher que se encarga de retirar los elementos de los hilos, se explica con más detalle en la siguiente entrada. En este elemento se encuentra la funcionalidad de reschedule o reorganización de los elementos en las colas de prioridades. En la última parte del simulador hay una simulación de procesos bloqueados. Se va restando el valor de blocked hasta que baja a 0 y se saca de la cola de procesos bloqueados.

Dispatcher: Este elemento se encarga de retirar e introducir procesos en los hilos. Para ello tiene implementada la expulsión por tiempo, utilizando el quantum, por tiempo de vida y por programas auto bloqueados.

Clock: Además de haber introducido unos parámetros más al visualizador, como la afinidad, estado del proceso y prioridad, se ha incluido una condición aleatoria que auto bloquee el proceso que toque por el ciclo.

PGenerator: Se ha mejorado el Pgenerator, Se han añadido la generación de campos de afinidad, prioridad, número de procesos generados por tick del timer.

Cabeceras y clases creadas y/o las más modificadas o importantes.

Clases y/o cabeceras	Descripción
scheduler.c, scheduler.h	Es la clase y cabecera que más se ha modificado. Se han separado las funcionalidades de reparto de tareas, dispatcher y control de prioridades.
dispatcher.c, dispatcher.h	Clase que contiene la funcionalidad del dispatcher, no es un elemento independiente, los schedulers lo utilizan.
simulador.c, clock.c, pgenerator.c	Se han tenido que modificar para poder implementar las nuevas funcionalidades como el autobloqueo y la jerarquía maestro esclavo de los schedulers.
list.c, list.h,	Estas es la clase y cabecera de las listas de PCB
rt_struct.c, rt_struct.h	Clase que contiene la estructura de la cola de prioridades y su bitmap.

Parametrización

En este punto la parametrización que se ha añadido está enfocada en los efectos que tiene en el simulador y en las políticas que tiene implementadas. La estructura utilizada para la parametrización se ha expandido.

Argumento	Descripción
-u --blocked_list_size	Tamaño máximo de la lista de bloqueado.
-s --block_chance	Probabilidad de que se autobloquee un proceso, más alto, más raro de que eso ocurra.
-f --max_blocked_time	Número de ciclos máximos que se quedan bloqueados los procesos.
-j --freq_reschedulo	Frecuencia a la que se recalcula las colas de prioridades, es la frecuencia por tick del scheduler.
-z --quantum_per_prio	Cantidad de quantum por cada nivel de prioridad. Escala linealmente.
-e --max_prio	Nivel de prioridad máximo para la tabla de prioridades.
-w --random_priority	Nivel máximo de prioridad que se puede asignar de forma aleatoria. Desde 0 hasta valor introducido.
-a --random_affinity	Poner afinidad aleatoria o no.
-n --pcb_generated	Número de procesos generados por tick de PGenerator.

Desarrollo de los elementos

A continuación voy a explicar brevemente cómo se han implementado los cambios desarrollados.

Schedulers: Para crear schedulers esclavos se utilizan las llamadas a las funciones como constructores de objetos y de esta forma no tener que tener un vector de variables por cada atributo que se quiera utilizar.

Los hilos scheduler maestro y esclavos se comunican mediante el vector de las estructuras de scheduler_dispatcher_data y variables condicionales. La sincronización se podría implementar de muchas otras formas como productor consumidor pero prefiero que tengan un orden para poder visualizarlo y manejarlo mejor, es decir, que primero se ejecute el scheduler maestro, reparta las tareas si es que tiene que repartir alguna y luego que los schedulers esclavos hagan las tareas que tengan que hacer.

En el dispatcher se comprueban los hilos que han finalizado, se han bloqueado o se les han acabado el quantum y los cambia por otros. Nunca va a dar el caso que se reemplace por sí mismo el hilo por cómo está hecha la cola de prioridades. Utiliza las variables contenidas en la estructura pcb_t.

Estructuras de datos

RT_Struct: Esta estructura es la cola de prioridades de los schedulers esclavos. Cada una de esta estructura tiene un número de colas que se utilizan para almacenar los procesos con el mismo nivel de prioridad. Además, contiene un mapa de bits que se utiliza para localizar las colas con algún elemento dentro. La gestión del mapa de bits es transparente mediante el uso de las funciones diseñadas para rt_struct. Tiene tres funciones, inicializar, añadir pcb_t y sacar pcb_t de un nivel de prioridad.

Lista de PCB: Junto a la estructura de nodos de la lista se compone esta estructura. Es una lista de elementos pcb_t y para lograr eso se utilizan nodos de doble dirección. Se han optado los de doble dirección para crear un código más sencillo de interpretar.

Scheduler_Dispatcher_Data: Esta agregación de estructuras contiene información sobre los schedulers esclavos.

Está formada por los siguientes elementos:

-Id, core_id, cpu_id: Identificadores para localizar sus elementos.
-assigned_core: Puntero al núcleo asignado la scheduler.
-rt y rt_expired: Estructuras de prioridades, Una para las actuales y otra para los expirados que se vuelcan al hacer reschedule.
-blocked_list: Lista de PCBs que se han autobloqueado.
-new_q: Cola de procesos nuevos asignados por scheduler maestros.
-Contadores para el control de reschedule y la prioridad con la que se está trabajando en ese momento.

También es utilizada por el Scheduler Maestro, aunque sea solo sea la cola de nuevos procesos.

PCB: Se le ha añadido unos campos para indicar el estado, afinidad, nivel de prioridad, quantum y tiempo que falta para ser desbloqueado (para la simulación).

Arbol rojo negro: Se ha quedado pendiente. Están los ficheros fuentes pero no están listos para compilación y mucho menos para ejecución.

Tercera parte del proyecto

"I have come to believe that a great teacher is a great artist and that there are as few as there are any other great artists. Teaching might even be the greatest of the arts since the medium is the human mind and spirit."

- *John Steinbeck*

"Most of us end up with no more than five or six people who remember us. Teachers have thousands of people who remember them for the rest of their lives."

- *Andy Rooney*

La tercera y última parte del proyecto se centra en la gestión de memoria. El simulador deberá leer y entender ficheros formateados como programas para el simulador y cargarlos en memoria siguiendo un sistema de paginación de memoria. El procesador simulado ejecutará operaciones y accederá a regiones de memoria con direcciones virtuales, por lo tanto, es necesario una unidad de gestión de memoria, también conocida como MMU (Memory Management Unit).

Antes de entrar en más detalle, una breve aplicación del sistema de paginado. Imagínate que tienes un programa que accede a una dirección de memoria en ejecución, como sabes que esa dirección va a estar disponible para el proceso, o qué ocurre si esa dirección ya está ocupada y hay hueco en la memoria de sobra para cargar el programa en memoria. Aquí es donde entra el sistema de paginado que añade una capa de abstracción al usuario del sistema operativo.

A continuación se listan las funcionalidades que se implementarán en esta expansión del simulador:

- Sistema de paginación:
    - Tablas de traducción.
    - MMU y TLB.
- Cargar programas en memoria:
    - Reservar memoria y generar tablas.
- Simular ejecución de programas:
    - Carga, las fases de operaciones (búsqueda, decodificación, carga, operación y escritura), descarga, bloqueos, fallos de página.

Para ello se desarrollarán los siguientes elementos:

- Memoria física y todo lo relacionado con el sistema de paginación.
- Loader. Se utilizará para cargar los programas en el simulador y en la simulación, dado que se cargan en la propia memoria física de la simulación.
- Extensión a la funcionalidad de la estructura de Machine. El motor de ejecución, es necesario para implementar estas funcionalidades de un procesador segmentado.
- Modificar elementos desarrollados:
    - Generación de PCB. Ahora no se generan aleatoriamente y los programas son realmente programas, es decir, tienen instrucciones y valores determinados que esperan resultados coherentes. Por ejemplo, una suma de dos números enteros.
    - Time to live y sus actividades aleatorias (bloqueos). Ahora existe una razón para que los procesos se terminen, bloqueen o finalicen forzosamente.

Solución desarrollada

Para elaborar la implementación de los elementos se ha seguido un orden concreto por las dependencias de las funcionalidades.

Empezando por el componente que nombra la tercera parte, la memoria (physical), además de la inicialización y gestión de los huecos e implantar la política de huecos, se han ido asentando los prototipos de funciones que deberán de desarrollarse para cumplir con los objetivos.

Después, siguiendo las anteriores orientaciones por parte de los prototipos, se ha implementado el motor de ejecución. Este punto son las funciones que permiten las ejecuciones de los programas simulados con las instrucciones simuladas y todas las estructuras relacionadas con este campo.

Por último, se ha implementado el cargador de los programas en memoria donde el formato de los programas, las instrucciones que contiene y las operaciones que se tienen que implementar han sido proporcionadas por el profesorado y las herramientas están incluidas en el apéndice.

Estructura modificada

En la realización de está parte se han modificado elementos que gestionaban el viejo motor de ejecuciones.

Clock: Las operaciones ahora se simulan y puede que ocurra uno de los siguientes tres casos: terminan, sale page fault por intentar acceder fuera de su rango de memoria o se bloqueen por simulaciones como fallo de cache (el load y store(write back)) y por no tener la página de la instrucción o dato en memoria.

Dispatcher: Se han implementado los cambios de contexto, se guardan los registros (del 0 al 15), el registro de instrucciones y el program counter (pc); este mismo punto se tiene en cuenta al cargar un proceso solo que se cargan en el hilo que se encargan, ya sea de memoria o nuevo. Además, al terminar un programa se descarga de memoria usando el Loader.

Thread (hilo) – Machine: Se ha añadido una MMU, se tiene en cuenta la inicialización de registros, el puntero PTBR y es esta estructura a la que se le hace el cambio de contexto.

PGenerator: En desuso, sustituido por loader.

Timer: Reemplazando timer de PGenerator por Loader.

Cabeceras y módulos creados y/o los más modificados o importantes.

Módulos y/o cabeceras	Descripción
physical.c, physical.h	Nuevo módulo que se encarga de inicializar, crear huecos, repartir huecos, restablecer huecos de la memoria principal física.
machine.c, machine.h	Módulo que contiene la funcionalidad del motor de ejecución. Las funciones Fetch, Decode, Load, Operation y Write se encuentran en estos ficheros.
loader.c, loader.h	Implementa la carga y descarga de programas al simulador y a la simulación. Además, tiene la función que se ejecuta cada tick del timer.
tlb.c, tlb.h	Este es el módulo y cabecera del TLB (Translation Lookaside Buffer). Tiene los métodos para añadir y retirar entradas.
queues.c, queues.h	Las utilidades de las colas se han separado de estructuras.c a este módulo y se han añadido las utilidades para la estructura de huecos de la memoria física.
estructuras.h	Cabecera modificada para añadir las estructuras de status, mmu, mm y atributos de estado al thread.
clock.c	Modificación para poder implementar la operación EXIT y bloqueos “inteligentes”.
simulador.c, timer.c	Pequeños cambios para cambiar el hilo del simulador de PGenerator por el de Loader.

Parametrización

Los 3 nuevos parámetros introducidos son program name, program start y number of programs. Uno de ellos (n_of_programs) reemplaza un parámetro de PGenerator.

Argumento	Descripción
-n --n_of_programs	Número total de programas que se quieren cargar. Si no existe ese número, irá cargando programas hasta encontrarse con un número faltante y fallará en cargarlo, pero no parará la simulación.
-o --name	Patrón que siguen los nombres de los programas. En el caso de “prog000.elf” el patrón es prog.
-v --start_number	Número inicial que toma el programa, en el caso anterior ese número es 0.

Desarrollo de los elementos

Antes de explicar cómo se han desarrollado los elementos independientemente, se tiene que hacer una visión general del sistema de paginado. Este sistema no ha sido reinventado y además de la memoria, se necesita una tabla de paginado de los procesos, un puntero a dicha página en los procesadores y un MMU en ellos. También se tienen que tener en cuenta elementos que utilizan estos elementos pero no son principales, como el cargador y descargador de programas, que debe pedir memoria a “physical” para almacenar la tabla de páginas de los procesos o descargarlo.

Los parámetros del sistema de paginado estaban fijados por lo que el trabajo ha sido mucho más ligero al no tener que crear un sistema genérico. Las características del sistema de paginado es el siguiente:

Páginas de 256B
Palabras de 4B
Direccionamiento físico y virtual 24 bits
Espacio lógico: 24 bits, 2^{24} = 16MB
Tamaño página offset: 8 bits
Número de páginas: 2^{16} = 64K = 65.536
Espacio físico: 24 bits, 16MB

Tamaño: 16 bits y 8 offset

Número de marcos: 16 bits
Número PTE: 2^{16}
Tamaño PTE: 2B

Con esos datos, sabemos el tamaño de la memoria (24 bits de direccionamiento, 2^{24} bytes (unsigned char)) y de los distintos elementos que forman las partes del sistema de paginado, como, por ejemplo, el Translation Lookaside Buffer (TLB), Memory Management Unit (MMU), MM del PCB y las Page Table Entries (PTE). La MMU contiene la TLB y este contiene las PTE. Además, a esas entradas se le han añadido un bit de validez para evitar un problema que se creaba al cambiar de contexto. Pero dejando para después esos pequeños detalles, ahora que se tiene una idea un poco más general de lo que está ocurriendo, se van a explicar los componentes que forman este sistema.

Physical
La memoria física es un elemento que simplemente contiene los procesos en memoria. Sigue un particionado variable de múltiplos de 256 bytes o 64 palabras y la política de Worst First, es decir, el peor hueco, escoge el mayor hueco y esto crea una fragmentación externa en los peores casos. La memoria por sí sola es simple, tiene elementos llamados huecos que se organizan en una lista de huecos para gestionar el espacio disponible.

Lo que se encarga del sistema de paginado no es la memoria, es la gestión de esta memoria y en este caso un MMU en cada procesador se encarga de esto y unas tablas para las traducciones de direcciones virtuales a físicas. Estas tablas son generadas por el Loader y debería de añadirse a la memoria principal pero eso sería crear una llamada malloc en el propio simulador y para aligerar un poco la carga de trabajo, se ha utilizado la llamada del sistema de malloc y se ha obviado. A la hora de descargar el proceso, esta tabla se libera de memoria.

MMU
Cada procesador que tenga un proceso asignado tiene un puntero (PTBR) a la tabla anteriormente mencionada. El PCB contiene datos acerca de las direcciones legales a las que el proceso puede acceder, por lo que si se encuentra con una dirección ilegal, termina el proceso con un page fault. Con los programas generados por Prometeo es imposible que esto ocurra pero una MMU debe encargarse de estas ocurrencias.

TLB
Hay una cosa que falta remarcar, no siempre se utiliza el puntero PTBR para acceder a la tabla. Si cada vez que se quiere acceder a una página debe mirarlo desde la memoria principal y eso quiere decir (o no) traer un bloque a la caché, va a ser una operación poco

eficiente. Es por eso que el TLB existe, una caché para las traducciones de direcciones virtuales a físicas. La política que se ha utilizado es FIFO y tiene 32 entradas.

El Motor de Ejecución simula las ejecuciones de las instrucciones de los procesos. Para ello se han diseñado las funciones con cada una de las fases de ejecución de un procesador segmentado pero haciéndolo todo seguido por problemas que se podrían generar. Estas fases son:

- Búsqueda (fetch): Accede a la dirección de memoria que indica el Program Counter (PC) y obtiene la instrucción y la coloca en el registro de instrucciones (RI).
- Decodificación (decode): Mira que código de instrucción debe de ejecutar utilizando RI.
- Carga de operandos (load): Según el código de instrucción, obtiene unos valores (número de registros o direcciones) del registro de instrucciones.
- Operaciones (actions): Según la operación o código de instrucción que debe de hacer, la efectúa. Load accederá a memoria (traduciendola), Store almacenará el valor del registro en la dirección de memoria (traduciendola), Add sumará los elementos y exit marcará el final de la ejecución del proceso.
- Escritura (write): Aquí es donde se deberían de avisar a los buffers de reservas y más cosas... Simplemente se guardan los datos de las operaciones Load y Add en sus registros destino.

Para efectuar este cambio de simulación, se ha tenido que cambiar el funcionamiento de TTL a que detecte EXIT (o term por PAGE FAULT) y si la traducción de virtual a física no está en la TLB o de forma aleatoria por LD y ST, se bloqueará la ejecución simulando un fallo de caché.

LOADER

El loader se encarga de leer los ficheros .elf, cargarlos en memoria (en la carpeta ./programs), descargarlos de memoria y crear los procesos. A la hora de cargarlos, se tiene en cuenta unos requisitos, espacio en la cola de procesos nuevos del scheduler maestro, que existan pcbs y que exista hueco en la memoria. Como este componente se encarga de crear los procesos, también se encarga de enlazar el proceso en memoria con una tabla de páginas para que se pueda hacer la traducción.

Cuando un proceso finaliza o se descarga por un fallo de página, se genera un fichero .end en el mismo directorio que estos programas con un volcado de la memoria en el estado de finalización para comprobar los resultados.

Estructura de datos

La estructura PCB ha tenido unas modificaciones: Se le han añadido dos estructuras nuevas, Status y MM (y se le ha retirado TTL).
Estructura de status: Contiene la información como PC, RI y los registros para usarlos en el cambio de estado.
Estructura de MM: Esta estructura tiene la dirección virtual de inicio del segmento de código del programa, la dirección virtual de inicio del segmento de datos del programa, la dirección virtual del final del programa y el puntero de la tabla de páginas del proceso.

Modificación machine, thread: Se le ha añadido la estructura de MMU (una colección de entradas de páginas), los registros de program counter, registro de instrucciones y registros generales.

Entrada de página: Tiene el número de página y el número de marco asociado. El número de página es un entero, 4 bytes, pero solo se necesitan 2 bytes para el número de página (aunque se usan 3 por comodidad, y evitar hacer el bitshift constantemente). El bit de mayor peso del atributo pagina indica si la entrada es válida o no.

Nodos de huecos: Estos elementos tienen los atributos de dirección y tamaño del hueco. También tiene un puntero al siguiente hueco.

Cola de huecos: Como la cola anteriormente implementada pero condicionada al tipo de elemento nodo de huecos.

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.

Respecto a la primera y segunda parte:
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.
Se ha intentado implementar una estructura utilizando árboles pero se ha dejado sin implementar y comentado el código por falta de tiempo.
Otra mejora interesante sería unas funciones y datos para capturar los datos y poder analizar el rendimiento de las diferentes políticas y parámetros. Falta meter más políticas en el simulador pero son muchas y prefiero mantenerlas fuera del proyecto para no enrevesarlo mucho más de lo que está.
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.

Respecto a la tercera parte, una interfaz gráfica más user friendly siempre viene bien, pero las dos mejoras que hubieran añadido valor a la simulación hubiera sido implementar una caché con cualquier política para ver la transferencia de bloques y tener un mejor control sobre los bloqueos; y la implementación de malloc en la memoria además de añadir las existentes estructuras a la memoria simulada.

Conclusiones

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 o en un curso tal vez.

Este simulador no llega a ser un simulador útil como tal en el mundo real, es más un prototipo educativo. Podría ser interesante seguir trabajando e investigando en esta área para ver los efectos de las diferentes políticas en las diferentes situaciones que pueden surgir, aunque de nuevo, al ser un campo tan extenso y profundo en cada pequeño detalle (el sistema operativo de linux tiene millones de líneas de código) es prácticamente imposible comprobar todo. Además, a cada problema se le presentan varias soluciones donde cada una tiene sus ventajas y desventajas. Si eso fuera todo, aparte de desarrollarse, deben mantenerse para poder actualizarse o arreglarse mientras se van encontrando problemas o implementando las mejoras y es un gran reto.

Pese a no entrar completamente en el mundo de los sistemas operativos y estar en aguas poco profundas, este proyecto ha servido para asentar lo aprendido y entender un poco más cómo funcionan realmente los sistemas operativos a bajo nivel.

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	github.com/torvalds/linux
Apuntes de SO	Egela
Apuntes de C avanzado	Egela
Manual getopt	man7.org/linux/man-pages/man3/getopt.3.html
Referencias pthread	hpc-tutorials.llnl.gov/posix/
Referencias Makefile	www.gnu.org/software/make/manual/make.html
Proyecto de Github(repositorio privado)	github.com/BorjaMoralejo/SO-Practica-21-22

Apéndices

Parámetros recomendados para prueba

Añadir 10 programas en ./programs (aviso que los programas que tienen el code_start y data_start en la misma página me han dado problemas y no funciona correctamente)
./simulador -m -c 1 -k 4 -t 2 -b 1000 -x 2000 -n 100 -g 4

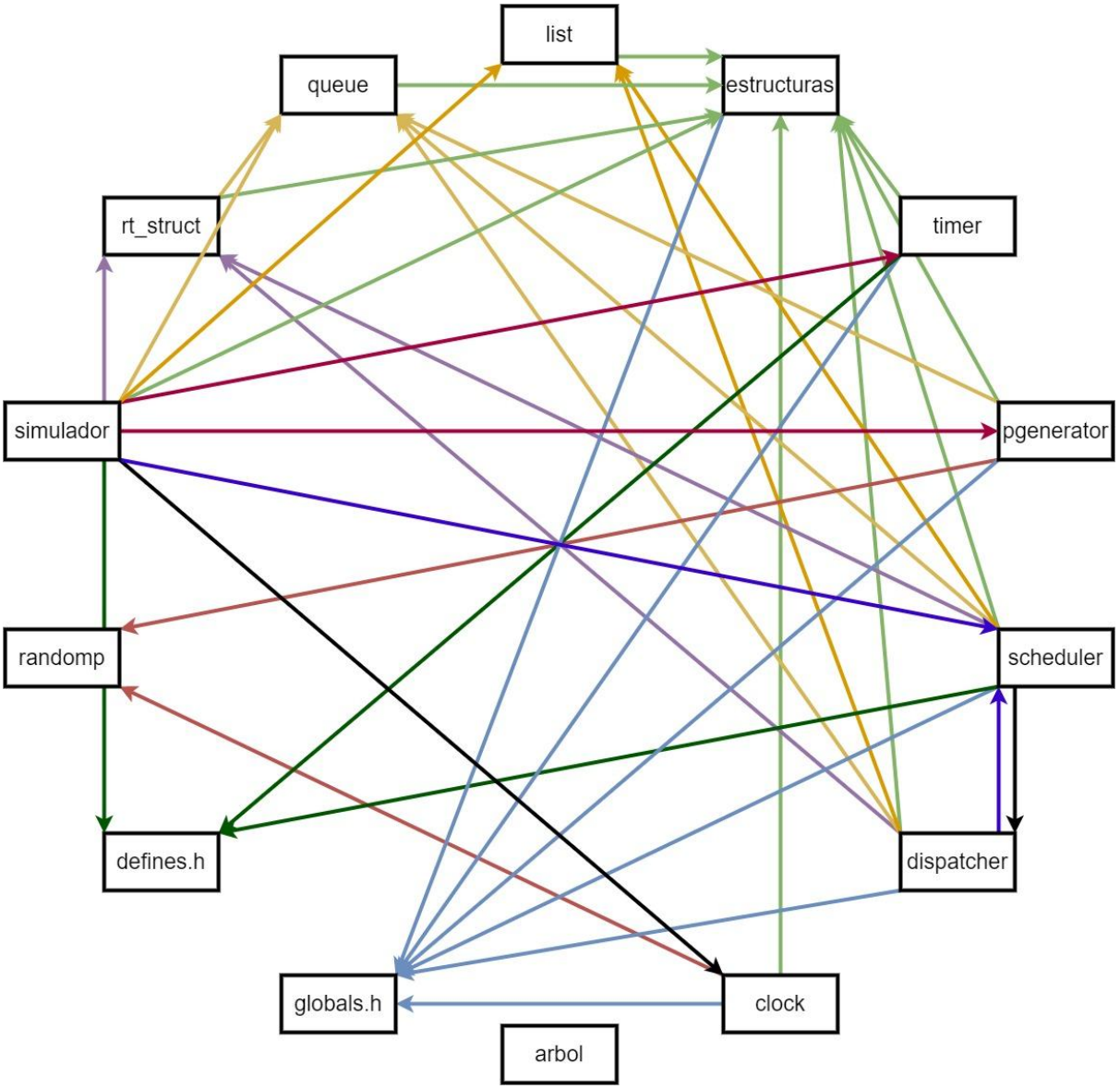
Fichero Makefile

Está diseñado para que solo compile los ficheros necesarios. Si se añaden nuevas fuentes o se crean nuevas dependencias, se deben añadir.
Nota: si se compila con -DDEBUG añadido en CFLAGS se activa el modo debug y muestra trazas de cada operación. Si vas a utilizar esto es recomendable poner el número de hilos, cpus y núcleos a 1 (-k 1 -t 1 -c 1).

```
SRCDIR = src
OBJDIR = obj
INCDIR = include
CFLAGS = -I$(INCDIR) -pthread

objs = $(addprefix $(OBJDIR)/, simulador.o clock.o timer.o scheduler.o estructuras.o randomp.o
queue.o rt_struct.o list.o dispatcher.o physical.o loader.o tlb.o machine.o)

vpath %.h $(INCDIR)
vpath %.o $(OBJDIR)
vpath %.c $(SRCDIR)

#Target principal, monta el simulador
simulador: $(objs)
	gcc -o simulador $(CFLAGS) $(objs)


#Target para cada clase
$(OBJDIR)/clock.o: clock.c globals.h estructuras.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/estructuras.o: estructuras.c globals.h estructuras.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/queue.o: queue.c queue.h estructuras.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/randomp.o: randomp.c
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/scheduler.o: scheduler.c globals.h estructuras.h queue.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/simulador.o: simulador.c defines.h estructuras.h queue.h estructuras.h clock.h
timer.h scheduler.h pgenerator.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/timer.o: timer.c globals.h estructuras.h defines.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/rt_struct.o: rt_struct.c queue.h estructuras.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/list.o: list.c list.h estructuras.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/dispatcher.o: dispatcher.c dispatcher.h estructuras.h rt_struct.h queue.h list.h
scheduler.h globals.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/physical.o: physical.c physical.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/loader.o: loader.c loader.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/tlb.o: tlb.c loader.h
	gcc -c $(CFLAGS) -o $@ $<
$(OBJDIR)/machine.o: machine.c machine.h
	gcc -c $(CFLAGS) -o $@ $<
```

Diagrama de dependencias

Utilizando la herramienta Draw.io.



Herramientas para generar y visualizar programas

Se han utilizado las herramientas para generar programas prometeo3 y para visualizar heracles3. Estos programas han sido facilitados por el profesorado.