Informe del trabajo final de sistemas operativos

- Comencé haciendo las clases que representan al programa y a la instrucción.

El programa contiene un nombre y una lista de instrucciones, las cuales pueden ser de IO o de CPU. Para representar esta diferenciación, cree una pequeña jerarquía de clases siendo la superclase una instrucción general que contiene un valor y un método abstracto que indica si es de CPU o no y siendo las subclases la instrucción de IO y la instrucción de CPU respectivamente que implementan el mencionado método.

- Implementé la clase del Kernel, el PCB, la CPU y el manejador de entrada/salida (IO Handler).

El PCB contiene un programa, una ráfaga (representado por la cantidad de instrucciones del programa), un pid (entero) y un estado (un string que simplemente indica en que estado se halla).

El Kernel contiene una cola de PCBs listos para ejecutarse, una CPU y un IO Handler. Además tiene dos métodos básicos, uno que elige el próximo PCB y le asigna la CPU si esta está libre y otro que carga un programa nuevo creando su correspondiente PCB y lo pone en la cola de listos.

La CPU conoce al Kernel y guarda un PCB que empieza en nulo y que eventualmente se lo asignará el Kernel. La CPU es un thread (subclase de Thread) por lo que implementa el método “run” que se encarga de ejecutar las instrucciones de CPU del PCB siempre que tenga uno asignado, si la instrucción llega a ser de IO le envía un signal al Kernel para que se lo asigne al IO Handler.

El IO Handler es otro thread como la CPU que conoce al Kernel y tiene una cola de PCBs listos esperando a que se libere algún dispositivo de I/O. En el método “run” se encarga de ejecutar la siguiente instrucción (ya sabe que es de IO) del próximo PCB en la lista (elije con FCFS). Si esta no era la última instrucción del PCB, le envia un signal al Kernel de interrupcion para que vuelva a poner al PCB en la cola de listos.

Tanto en la CPU como en el IO Handler si la instrucción ejecutada es la última del PCB, le envian un signal al Kernel para que termine el proceso.

El Kernel se encarga de comenzar los threads de CPU e IO Handler en su constructor.

- Agregué la clase IRQ y la clase Scheduler.

El IRQ se encarga ahora de manejar las interrupciones que son enviadas al Kernel por la CPU y el IO Handler. El Kernel contiene un IRQ y le designa toda la responsabilidad de esos metodos. El IRQ cuenta con un semáforo que se encarga de hacer entrar en modo kernel al sistema cuando hay una interrupción evitando asi que sigan corriendo los threads de la CPU y el IO Handler. Las interrupciones que maneja son: cuando entra un proceso nuevo y se carga en la cola de listos, cuando se le va a asignar un proceso al manejador de entrada/salida, cuando se termina un proceso, y cuando se suspende un proceso y vuelve a entrar a la cola de listos.

El Scheduler es el planificador de corto plazo que se encarga de escoger el siguiente PCB a ejecutar. Contiene al algoritmo que se decidirá el PCB y dos métodos, uno para agregar un PCB nuevo a la cola y otro que retornará el PCB elegido para ejecutar. Ambos métodos son encargados al algoritmo.

El algoritmo del scheduler es un strategy pattern que tiene una estructura de datos que representará de ahora en adelante a la cola de PCBs listos que antes estaba en el Kernel y que implementa los métodos de agregado y de selección de PCBs.

Implementé tres de los algoritmos vistos en clase: First Come First Serve, Shortest Job First y Round Robin con prioridad no expropiativo. En el caso de Round Robin con prioridad implementé además el envejecimiento de los procesos representado por un diccionario de las prioridades conteniendo a su vez como valor una lista de diccionarios donde la clave es el envejecimiento y el valor es la lista de procesos con dicho envejecimiento y prioridad. El próximo a ejecutar es aquel con menor prioridad y mayor envejecimiento.

Agrego al PCB dos variables de instancia, el quantum y la prioridad. Ambos empiezan en nulo y los setea el Scheduler al momento de ser asignados a la CPU en caso de ser Round Robin con prioridad.

El Kernel conoce al planificador de corto plazo.

- Implemento la clase MMU (memoria) y LongTermScheduler (planificador de largo plazo).

El planificador de largo plazo (LTS) recibe procesos nuevos y se encarga de decidir cuales se pueden cargar en memoria y cuales deben esperar a que haya espacio. Conoce a la memoria, y tiene una lista de procesos entrantes que esperan memoria y la id actual que debe asignar al siguiente PCB. Tiene tres métodos principales: uno para crear un PCB a partir de un programa dado al que luego chequea si se puede cargar en memoria, si no puede lo agrega a la cola de procesos entrantes, el segundo para decirle a la memoria que cargue un proceso y al planificador de corto plazo que lo agregue a la cola de listos y el tercero para terminar un proceso diciendole a la memoria que lo descargue, al hacer esto revisa si puede cargar a memoria uno de los procesos en la cola de entrantes, elige al primero que pueda cargarse y por orden de llegada.

El Kernel conoce al planificador de largo plazo.

La clase MMU representa a la memoria, tiene dos variables: la memoria física (un diccionario de dirección → valor) y la memoria lógica (es distinto dependiendo del algoritmo que se use). Implementa cuatro metodos: uno que chequea si un PCB puede cargarse en memoria fisica, el segundo carga un PCB en la memoria fisica y la memoria lógica, el tercero descarga un PCB de la memoria fisica y de la memoria lógica, y el último busca un PCB en la memoria física. Todos estos metodos son delegados al algoritmo de la memoria lógica.

- Implemento el algoritmo de asignación continua (MVT).

La clase MVT recibe un entero que representa el tamaño de la memoria y una estrategia de selección de bloques (primer ajuste, mejor ajuste o peor ajuste) en el constructor y guarda la estrategia elegida además de contener bloques llenos (diccionario de PCB → bloque) y bloques vacios (en lista) que representan el estado de la memoria física.

Un bloque es una clase que tiene una direccion de inicio y una direccion de fin, el anterior y el próximo bloque dentro de la memoria, un booleano indicando si está vacío y un desplazamiento del PCB asignado a el en caso de estar lleno. Al ser subclase del algoritmo de memoria debe implementar los cuatro metodos mencionados arriba de MMU.

1- El método que verifica que haya al menos un bloque vacío que pueda contener al PCB a cargar o bien que haya espacio suficiente en la suma de espacio de todos los bloques vacíos.

2- El método de carga a memoria cumple varias actividades. Primero busca el bloque vacío que almacenará al PCB en base al método de selección que se le haya asignado. Si encuentra dicho bloque vacío lo divide en una parte llena y otra vacía dependiendo de la ráfaga del proceso, actualiza su estado y le asigna ese PCB a la parte llena; luego se carga en la memoria física instrucción por instrucción y le asigna la direccion base al PCB. Si no encuentra bloque vacío que lo contenga es porque hay espacio suficiente para cargar el proceso pero los bloques están separados por lo que hace una compactación, un algoritmo que se encarga de juntar los bloques vacíos de la

memoria lógica hasta que el bloque fusionado sea lo suficientemente grande para contener al proceso. Una vez que termina la compactación carga el PCB en ese bloque fusionado.

3- El método de descarga de memoria elimina el bloque lleno con ese PCB del diccionario de bloques llenos y lo inserta en la lista de bloques vacíos, y dependiendo de las direcciones que abarquen los otros bloques vacios, lo fusionará con esos bloques o lo insertará como un nuevo elemento. Basandose en la direccion base y la rafaga del PCB descarga todas sus instrucciones de la memoria física.

4- El método de busqueda de la siguiente instrucción toma la dirección base del PCB y busca el bloque lleno que empieza en esa dirección, una vez que lo encuentra accede al valor de la direccion base sumado al desplazamiento contenido en el bloque y lo retorna. Incrementa en 1 el desplazamiento.

El Kernel ahora puede recibir una memoria en el constructor también y se lo asigna a la CPU y al planificador de largo plazo.

La CPU ahora va a buscar la siguiente instrucción del PCB en la memoria.

El PCB ahora guarda también la direccion base que se lo asigna la memoria en caso de ser MVT.

- Implemento el algoritmo de paginación.

La clase Pagination contiene el tamaño de las páginas (entero), una lista con los marcos que están vacíos, una lista con los numeros de las paginas disponibles, un diccionario de paginas en uso (en lista) con el PCB como clave y un diccionario que representa la tabla de páginas, es decir, a que marco corresponde cada pagina en uso.

Como es otro algoritmo de memoria también implementa los cuatro métodos de MMU.

1- El método que verifica si un PCB puede cargarse en memoria simplemente se fija si la cantidad de marcos vacíos alcanza para guardar todas las instrucciones del proceso.

2- El método de carga a memoria, por cada marco que va a necesitar, crea una pagina nueva basandose en los numeros de paginas libres y en el tamaño de pagina y lo agrega al diccionario de paginas del PCB a cargar. Luego saca uno de los marcos vacíos de la lista y agrega una nueva entrada a la tabla de paginas entre el numero de pagina nuevo y el marco. Se carga la página con la cantidad de instrucciones restantes del proceso que puedan entrar. Basandose en la dirección inicio del marco y el contenido de la página se cargan las instrucciones en la memoria física. Todo esto se repite por cada marco que se necesite hasta cargar todas las instrucciones.

3- El método de descarga del PCB, obtiene todas las paginas asignadas a ese PCB desde el diccionario de paginas en uso y por cada pagina hace la descarga física por medio del marco que se obtiene a partir de la tabla de páginas, de esta manera ubica en que direcciones fisicas se encuentran las instrucciones. Luego actualiza los numeros de pagina libres.

4- El método de busqueda de la siguiente instrucción se basa en el numero de la página actual sumandole el desplazamiento actual dentro de esa misma pagina, ambos datos contenidos en el PCB. Con el número de página consigue la dirección del marco asociado a la pagina y a eso le suma el desplazamiento para acceder a la siguiente instrucción en la memoria física y lo retorna.

El PCB ahora guarda también por un lado, la página actual donde tendrá que buscar la siguiente instrucción y su desplazamiento dentro de la página y, por otro lado, las paginas restantes asociadas a ese PCB.

- Implemento la clase HDD que representa al disco.

Esta clase al igual que la memoria y el dispatcher dispone de un algoritmo, en este caso se encarga

de manejar el modo en que se guardan los programas en forma lógica dentro del disco físico. Los algoritmos deben implementar los métodos de guardar programa en disco, de mostrar todos los programas guardados y de busqueda de un programa específico a partir de su nombre.

Yo implementé el algoritmo de Inodo, para ello la clase guarda un mapa de todos los nodos; incluyendo los que son indices y los que apuntan a la dirección física en disco, un diccionario de inodos teniendo como clave el nombre del programa y una lista de numeros de nodo libres.

Al momento de guardar un programa en los sectores del disco, primero chequea que hayan suficientes nodos para contener el tamaño del programa. Si puede guardarse entonces primero elije un numero de nodo libre para el nuevo inodo que por el momento es una lista de nodos vacía, luego por cada instrucción del programa elije un nuevo numero de nodo libre y agrega una entrada nueva en el mapa de nodos con esa entrada para luego actualizar el sector del disco correspondiente a esa entrada con la instrucción actual. Cuando termina de guardar todas las instrucciones actualiza la entrada del inodo nuevo con la lista de todos los indices de los nodos que contienen datos.

Cuando hace la busqueda de un programa por medio del nombre, lo reconstruye usando el diccionario de inodos donde tiene todos los indices de nodos con cada instrucción.

El planificador de largo plazo ahora va a buscar el nombre de programa que recibe al disco para luego crear su PCB y eventualmente cargarlo en memoria.

- Implemento una clase para cargar programas en disco (ProgramLoader) y una clase Shell.

El ProgramLoader es una simple clase que crea una X cantidad de programas con Y cantidad instrucciones CPU y Z cantidad de instrucciones IO, que luego va guardando en disco.

El Shell es otro thread y en el método “run” hice dos implementaciones, una para que vaya cargando cada tanto un programa al azar en el Kernel y otra más manual donde el usuario escribe le nombre del programa a cargar.