Práctica No. 2

PROCESOS

Integrantes:

Gómez Luna Alejandro

Grupo: 2

Profesor:

M. I. Edgar Tista García

Semestre:

2021-2

Fecha:

20/junio/2021

Objetivo

- Implementar los conceptos del curso relacionados con procesos realizando una implementación que permita simular la ejecución de procesos en un sistema operativo
- Que el alumno pueda retomar sus habilidades de programación aplicándolo a un concepto fundamental del curso

Desarrollo

Para el desarrollo de esta práctica se utilizó el lenguaje de programación Java, ya que se encuentra en un paradigma orientado a objetos, lo cual facilita bastante la realización de las diversas actividades propuestas, como se detallará a continuación.

Se realizó la creación de tres clases. A continuación, se detallan los atributos y métodos que conforman a dichas clases, así como el objetivo con el que fueron creadas:

- Main: Esta clase sirve solamente para poder realizar una instancia de la clase Planificador y así empezar la ejecución del programa.
- Proceso: Esta clase es la que nos permitirá crear objetos que modelen a un proceso. Esta clase cuenta con los siguientes atributos:
  - *nombre*: Es un atributo de tipo String, el cual almacena el nombre del proceso.
  - *proceso_id:* Es un atributo de tipo entero, el cual almacena el id del proceso, el cual es un identificador único que se asocia con cada proceso.
  - instrucciones: Es un atributo de tipo entero, el cual almacena la cantidad total de instrucciones que tiene cada proceso.
  - *instruccionesRestantes:* Es un atributo de tipo entero, el cual llevará el control de la cantidad restante de instrucciones que faltan por ser ejecutadas por un determinado proceso para que finalice su ejecución. Este atributo tomará valores entre 10 y 30.
  - *memoria:* Es un atributo de tipo entero, el cual almacena la cantidad de memoria que ocupa el proceso. Este atributo tomará los valores puntuales de 32, 64, 128, 256 o 512.

  Todos los atributos, a excepción de instruccionesRestantes, son de tipo final, ya que, una vez creado el proceso, no se pueden modificar los valores de sus atributos. El atributo instruccionesRestantes no es final porque este se irá modificando conforme se utilice el método ejecutarProceso().

  También es importante remarcar que los atributos son privados, por lo tanto, solo se podrán acceder a ellos a través de algunos métodos definidos.

  Asimismo, para los métodos de esta clase se tiene:
  - *Proceso():* Constructor de la clase. Recibe como parámetros el nombre e id con el que se identificará a un proceso. Para la parte de instrucciones y memoria, estas se generan de manera aleatoria, siguiendo el rango establecido.
  - *ejecutarProceso():* Este método es el que permita simular la ejecución de un proceso en CPU. Modifica el valor de instruccionesRestantes, decrementando su valor en 5, pues se supone que se han ejecutado 5 instrucciones del proceso. En caso de que se llegue a un valor menor o igual a cero, entonces se retorna true, lo cual significa que el proceso ya ha finalizado. En caso contrario, retorna false, lo cual significa que el proceso aún tiene instrucciones faltantes.
  - *mostrarProceso():* Este método muestra en pantalla los valores de los atributos de un proceso, como lo son su nombre, ID, instrucciones totales y ejecutadas.

- *memoriaOcupada():* Este método sirve para obtener la memoria que ocupa un proceso. Retorna el valor como un múltiplo de 32, lo cual representa un «bloque de memoria».
- *procesoID():* Este método sirve para obtener el ID asociado a un proceso.
- *nombreProceso():* Este método sirve para obtener el nombre de un proceso.
- *instruccionesPendientes():* Este método sirve para obtener la cantidad de instrucciones faltantes de un proceso.

- Planificador: Esta es la clase primordial, en la cual se realiza el proceso de simulación de un Planificador de Procesos. Cuenta con los siguientes atributos:
  - *colaProcesosListos, colaProcesosFinalizados* y *colaProcesosAniquilados:* Estos tres atributos son estructuras de datos de tipo cola, implementados en Java a través de Queue. Se decidió por implementar colas ya que es lo más apropiado para llevar el control de los procesos, como se detalla a continuación
    - La colaProcesosListos es la cola que servirá para simular a la cola de procesos preparados, es decir, aquí es donde se irán enfilando los nuevos procesos que se crean, esperando su turno a ser atendidos por el CPU.
    - La colaProcesosFinalizados es la cola que servirá para registrar a todos los procesos que hayan finalizado su ejecución.
    - La colaProcesosAniquilados es la cola que servirá para registrar a todos los procesos que hayan sido aniquilados.
  - *tablaMemoria*: Es un arreglo bidimensional de enteros, el cual sirve para simular a la tabla de localidades de memoria en donde se almacenarán los diversos procesos que se creen. Sus dimensiones son de 64 x 32, dando un total de 2048 localidades. Estas dimensiones se escogieron ya que todos los procesos ocupan espacios múltiplos de 32, por lo tanto, para cada renglón (32 localidades de memoria) se almacena solamente el valor del ID del proceso en la primera columna de un renglón, y con eso se sabe que el proceso con dicho ID está ocupando las localidades de memoria de dicho renglón.
  - *idProcesoActual:* Es un número entero secuencial, el cual inicia con un valor de 1 y será el identificador que se asocie con cada proceso creado.

  Como se puede observar, la clase Planificador será la encargada de la creación y gestión de diversas instancias de la clase Proceso, ya que es el planificador el encargado de llevar el control de los diversos procesos, así como de administrar la memoria que ocupan.

  Dentro de los métodos, encontramos:
  - *Planificador():* Constructor de la clase. No recibe parámetros, solamente inicializa las colas y la tabla de memoria. Las colas se inicializan sin elementos y la tabla de memoria se inicializa con valores de cero.
  - *crearProceso():* Es el método encargado de realizar la operación de "Crear Proceso nuevo". Una vez creado el proceso, se busca dónde almacenarlo en la tabla de memoria.

    Para asignarle memoria al proceso, solamente se busca que exista un cero en el primer registro de algún renglón, pues el cero en un renglón significa que ese espacio está libre. Se repite este proceso hasta que se asegure que se pueden asignar todos los bloques de memoria correspondientes al proceso.

En caso de que no haya suficiente espacio, se muestra un mensaje al usuario (Sistema Operativo) y el proceso no se crea. En caso contrario, el proceso se crea y se enfila en la cola de procesos preparados.

- *verEstadoActual():* Es el método encargado de realizar la operación de "Ver Estado Actual del Sistema".
  Utiliza el método size() del atributo colaProcesosListos para saber cuántos procesos están enfilados listos para ejecutarse.
  Los atributos colaProcesosFinalizados y colaProcesosAniquilados se recorren utilizando un for each, con la finalidad de conocer los procesos que ya terminaron su ejecución y que fueron interrumpidos, respectivamente. Se muestra para cada proceso su ID y nombre correspondiente.
  Por último, se muestra la tabla de memoria, solamente mostrando el ID asociada al proceso que está ocupando alguna determinada fila en memoria, o 0, indicando que dicha fila está libre.
- *mostrarProcesosActuales():* Es el método encargado de realizar la operación de "Imprimir cola de procesos".
  Se utiliza el atributo colaProcesosListos para recorrerlo y así conocer el ID junto con el nombre de cada proceso listo para ejecutarse.
  Asimismo, se utiliza el método peek() de colaProcesosListos, el cual permite obtener el primer proceso en la cola, el cual funge en la simulación como el proceso activo, es decir, que se encuentra siendo ejecutado por el CPU.
- *ejecutarProceso():* Es el método encargado de realizar la operación de "Ejecutar proceso actual".
  Se utiliza el método poll() del atributo colaProcesosListos, pues este nos permite remover y obtener el primer proceso que se encuentra en la cola de procesos listos. Una vez que obtenemos el proceso podemos utilizar su método ejecutarProceso() para simular la ejecución de dicho.
  En caso de que el proceso termine su ejecución, se le notifica al usuario (Sistema Operativo) y posteriormente, se buscan la(s) fila(s) en memoria que ocupaba el actual proceso, con la finalidad de liberarlas. Solamente se busca en la primera columna de la tabla de memoria, ya que es la que contiene el ID de cada proceso. Se busca que el ID del proceso sea igual a la de la fila y se cambia su valor a 0, para notificar que esa fila ahora queda libre y podrá ser utilizada para almacenar un nuevo proceso. Por último, este proceso finalizado se enfila en la colaProcesoFinalizados, pues es el que lleva el registro de procesos finalizados.
  En caso de que no termine su ejecución, solamente se manda al final de la cola.
- *procesoActual():* Es el método encargado de realizar la operación de "Ver proceso actual".
  Nuevamente utiliza el método peek() de colaProcesosListos, y, una vez se obtiene el proceso activo, se emplea su método mostrarProceso().
  Finalmente se buscan las filas en memoria que tengan el mismo ID que el proceso activo, para determinar el espacio que ocupa.
- *siguiente():* Es el método encargado de realizar la operación de "Pasar al proceso siguiente".
  Solamente saca al proceso activo de la colaProcesosListos y lo manda al final de la cola.

- *aniquilarProceso():* Es el método encargado de realizar la operación de “Matar proceso actual”.
  Nuevamente se utiliza el método poll() de colaProcesosListos, y, una vez se obtiene el proceso activo, se procede a liberar sus localidades de memoria.
  Finalmente, este proceso finalizado se enfila en la colaProcesoAniquilados, pues es el que lleva el registro de procesos aniquilados y se muestra la cantidad de instrucciones que le faltaban por ejecutar.
- *aniquilarTodo():* Es el método encargado de realizar la operación de “Matar todo y terminar”.
  Se recorre colaProcesosListos y para cada proceso en ella se muestran las localidades de memoria que ocupaban en la tabla de memoria. También se agrega cada proceso a colaProcesoAniquilados.
  Finalmente se limpia la colaProcesosListos mediante su método clear().
- *desfragmentar():* Este método es «extra», y es utilizado para desfragmentar a la tabla de memoria, ya que, conforme se eliminan procesos, estos empiezan a dejar espacios en filas no necesariamente contiguas, por lo que, cuando se crean nuevos procesos, estos tenderán a almacenarse en filas dispersas.
  Atendiendo a la simulación, esto podría provocar que la ejecución del proceso se tarde porque se deben de realizar saltos en la memoria, por lo que, con la finalidad de mejorar el rendimiento, se desfragmenta la tabla de memoria para que los procesos se almacenen en filas contiguas y no se pierda tiempo realizando saltos.
  Para el proceso de desfragmentación primero se obtienen los ID y espacios que ocupa cada proceso que se encuentra en colaProcesosListos. Posteriormente, se ordenan los ID de cada proceso y se irán almacenando nuevamente en memoria, pero de manera contigua.
- *mostrarMenu():* Es el método encargado de mostrar el menú con el cual se pueden realizar las diversas operaciones propuestas.

Finalmente, se tiene el video donde se muestra la simulación del planificador de procesos.
https://drive.google.com/file/d/1c_1jZ_g_yttWn5_L01-lon33UQ1GR-B0/view?usp=sharing

Conclusiones

Para esta práctica se pudo comprender de mejor manera los diversos conceptos teóricos revisados en clase para la parte de procesos.

Como se ha revisado en diversas sesiones, existe el Planificador de Procesos, el cual es uno de los módulos más importantes de todos los Sistemas Operativos, ya que es el encargado de llevar la correcta gestión de los procesos que se encuentran en el sistema.

A través de las diversas actividades de la práctica se pudo comprender de mejor manera cómo se realiza la gestión de procesos en un S.O, ya que, si bien solo es una simulación simple, nos ofrece un mejor entendimiento del planificador de procesos, así como de las dificultades que pueden llegar a presentarse.

Para realizar la simulación se tuvieron que considerar varios aspectos. Entre estos aspectos está la manera en cómo se modelarían los procesos y la forma de irlos almacenando. Esto se pudo solventar gracias a que Java es un lenguaje orientado a objetos, lo que permitió la creación de los objetos Proceso y Planificador, a los cuales se les definieron sus respectivos métodos y atributos para poder llevar a cabo las actividades solicitadas en la práctica. El objeto Proceso contenía la información necesaria para identificar a un proceso, como nombre, identificador, cantidad de instrucciones, etc. El objeto Planificador contenía la cola de procesos listos para ejecutarse, en donde se enfilaban los nuevos procesos creados.

Asimismo, otro aspecto importante a considerar es la asignación de memoria que ocupaba cada proceso, lo cual se pudo solventar con la definición de un arreglo bidimensional, el cual servía para simular la tabla de memoria. En este caso, cada fila de la memoria se conformaba por 32 localidades, y contaba con un total de 64 filas para un total de 2048 localidades de memoria.

Se pudieron realizar las diversas operaciones propuestas, como aniquilar, ejecutar y ver diferentes características de los procesos a lo largo de la simulación.

La práctica nos permite entender de mejor manera la dificultad que es gestionar procesos, porque muchas veces nosotros como usuarios no comprendemos o visualizamos todas las actividades que realiza un S.O para que podamos realizar nuestras tareas en la computadora. Con la realización de las actividades en la práctica podemos comprender de mejor manera que el planificador debe de estar siempre al pendiente de coordinar la ejecución de los procesos, los cuales tienen un tiempo finito de ejecución en el CPU. En nuestras computadoras esto se complica conforme se añaden más factores, como la asignación correcta en memoria, la existencia de diversos procesadores, la coordinación para tener paralelismo o concurrencia entre los diversos hilos que pueden llegar a conformar a los procesos, control de interrupciones, creación de nuevos procesos a partir de un proceso padre, entre muchas otras actividades.