Práctica No. 4

ADMINISTRACIÓN DE MEMORIA

Integrantes:

Gómez Luna Alejandro

Grupo: 2

Profesor:

M. I. Edgar Tista García

Semestre:

2021-2

Fecha:

18/julio/2021

Objetivo

- Implementar los conceptos del manejo de memoria en los sistemas operativos aplicando dichos conceptos en la simulación de creación y ejecución de procesos.

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.

Tomando como base el simulador realizado con anterioridad para la práctica número 2 de procesos, se realizaron las siguientes modificaciones y nuevas implementaciones para las mismas tres clases, además de añadir una nueva clase:

- Main: Sin modificaciones.
- Proceso: Se añadieron los siguientes atributos:
  - nPaginas: Es un atributo de tipo entero, el cual almacena la cantidad de páginas en las que está dividido el proceso, recordando que el tamaño de página es de 16 localidades.
  - cargadoCompleto: Es un atributo de tipo boolean, el cual sirve como bandera para conocer si el proceso actual ha cargado en su totalidad todas sus páginas en memoria.
  - paginasCargadas: Es un atributo de tipo entero, el cual sirve para conocer la cantidad total de páginas que el proceso actual tiene cargadas en memoria. Está en estrecha relación con el atributo cargadoCompleto, ya que, si está almacenado totalmente en memoria, todas sus páginas estarán cargadas. En caso contrario, se irá realizando el conteo de las páginas que se carguen en memoria.
  - tablaPaginas: Es un atributo que es un arreglo bidimensional de enteros, en el cual se tiene que la primera columna almacena el número de página y la segunda columna almacena el marco correspondiente a dicha página. En caso de no estar cargada la página, se almacena un -1. Como se revisó en teoría, la tabla de páginas permite establecer la relación entre cada página y su correspondiente marco, con la finalidad de que, en un S.O real, se puedan encontrar las instrucciones correspondientes de un proceso en memoria física.
  - secuenciaEjecucion: Es un atributo que es una lista ligada de enteros, en la cual se generará una secuencia de ejecución aleatoria para el proceso, en función de su número de páginas e instrucciones. Esta secuencia de ejecución corresponde a lo que se revisó en teoría cuando se habla de que un programa irá ejecutando diversas instrucciones, las cuales se encuentran almacenadas en diversas páginas, por lo que, esta secuencia de ejecución representa a las páginas que se irán necesitando conforme se vaya ejecutando un proceso.

  Solamente el atributo nPaginas es final, ya que el número de páginas del proceso siempre será el mismo, pero las páginas que se carguen en memoria, si está cargado en su totalidad, etc, pueden modificarse a lo largo de la ejecución del programa.

  Asimismo, para los métodos de esta clase se tiene:

  - *Proceso():* Se le añadió la inicialización de los atributos tablaPaginas, nPaginas y secuenciaEjecucion. nPaginas se obtiene dividendo la memoria entre 16, tablaPAginas se inicializa con valores de -1 para sus respectivas páginas y secuenciaEjecucion se genera aleatoriamente.

- *ejecutarProceso():* Se modificó su funcionamiento. Si el proceso no está cargado totalmente en memoria y al ejecutarse no finaliza, entonces se utiliza el método reemplazoPagina(), al cual se le pasará la actual página de la secuencia de ejecución, repitiéndolo 5 veces, pues es el número de instrucciones que se ejecutan.
- *reemplazoPagina():* Este método sirve para reemplazar las páginas de un proceso que no se encuentra almacenado por completo en memoria. Se utilizó el concepto de reemplazo de página visto en teoría, específicamente el algoritmo de reemplazo óptimo, ya que se tiene toda la cadena de referencia de cada proceso, por lo que es posible conocer los siguientes pasos que ejecutará el proceso y así es como se puede aplicar este algoritmo.
  Recibe como parámetro la página actual de la secuencia de ejecución y, para todas las páginas que tienen un marco asignado, busca aquella que se utilice hasta más adelante en la secuencia de ejecución para reemplazarla por la página actual. En caso de que una página no se encuentre en la cadena de ejecución, entonces se dejará de buscar y esta será reemplazada.
- actualizarPaginas(): Este método actualiza la tabla de páginas del proceso actual. Recibe como parámetro un arreglo de enteros, el cual almacena los marcos disponibles. Si el proceso se encuentra cargado totalmente en memoria, actualiza el marco de todas sus páginas. En caso contrario, solo actualiza el marco de las páginas posibles.
- *mostrarProceso():* Se le añadió que también mostrara la cadena de ejecución del proceso actual, así como su tabla de páginas.
- *memoriaOcupada():* Se modificó para que ahora retorne el número de páginas que ocupa el proceso.
- *paginasCargadas():* Este método devuelve el número de páginas cargadas en memoria para el proceso actual.
- *getCargado():* Este método devuelve si el proceso actual se encuentra o no cargado totalmente en memoria.
- *setCargado():* Este método recibe como parámetro un valor boolean, el cual será asignado al atributo cargadoCompleto del proceso actual.
- *resetPaginas():* Este método sirve para que reiniciar los valores de la tabla de páginas del proceso actual, asignando un valor de -1 a cada página, significando que no tienen ningún marco asignado.

- Planificador: Se añadieron y modificaron los siguientes atributos:
  - *tablaMemoria*: Se modificaron sus dimensiones a 64 x 16, pues la memoria se restringió a 1024 localidades, además de que ahora el tamaño mínimo de almacenamiento es igual al tamaño de página, que es de 16 localidades.
  - *hayEspacio:* Es un atributo de tipo boolean, el cual sirve como bandera para determinar si existe espacio suficiente en memoria para almacenar más procesos.
  - *enEspera:* Es un atributo de tipo Proceso, en el cual se almacenará un proceso que no se haya podido cargar completamente en memoria.
  - *espaciosFaltantes:* Es un atributo de tipo entero, el cual está relacionado con el atributo enEspera, pues sirve para contabilizar los espacios faltantes para almacenar en su totalidad al proceso que no se pudo cargar completamente en memoria.

- *listaMemoria:* Es un atributo que es una lista ligada de BloqueMemoria, el cual sirve para representar a la lista ligada de memoria que lleva el registro de los procesos y huecos en un tiempo de la ejecución.

Dentro de los métodos, encontramos:

- *Planificador():* Solo se añadió la inicialización del atributo listaMemoria.
- *crearProceso():* Se modificó para que, en caso de que un proceso no se pueda almacenar totalmente en memoria, se lleve el registro de este a través de los atributos espaciosFaltantes y enEspera, además de llamar a sus respectivos métodos setCargado() y actualizarPaginas(). Asimismo, para buscar un espacio donde poder almacenar a un proceso, se utiliza el atributo listaMemoria, pues es una lista ligada que lleva el registro de los huecos y procesos en memoria, lo que facilita la búsqueda. Esto se observó en teoría cuando se hablaba de el primer, peor y mejor ajuste, en el cual se asignaba un nuevo proceso al primero hueco donde quepa, el hueco con más espacio y el hueco donde se desperdicie menos memoria, respectivamente. Para este caso, se realiza el primer ajuste.
- *actualizarLista():* Este método es el encargado de que, siempre que existe cualquier tipo de cambio en las localidades de memoria, se actualicen los bloques en la lista ligada de memoria.
- *verEstadoActual():* Sirve para realizar la operación “Ver estado de los procesos”. Sin modificaciones.
- *verLista():* Este método sirve para realizar la operación “Ver estado de la memoria”. Se encarga de recorrer a la lista ligada de la memoria, y mostrar en pantalla si cada bloque que la conforma corresponde a un proceso o a un hueco, así como el registro de memoria donde empieza y su longitud.

  Para realizarlo se recorre toda la primera columna de la tabla de memoria, pues es donde se almacena el id de los procesos que ocupa una determinada fila de memoria (16 localidades). Mientras se realiza el recorrido se verifica que el número de cada fila sea igual. En cuanto se produzca un cambio, entonces se tiene un nuevo proceso almacenado o , si el número es cero, un hueco.
- *agregarPaginasFaltantes():* Este método se utiliza para cuando existe un proceso que no se encuentra almacenado totalmente en memoria y se acaba de liberar espacio en memoria debido a que un proceso finalizó su ejecución o fue aniquilado. Recibe como parámetro un entero y un arreglo de enteros, en los cuales se almacena la cantidad de espacios liberados en memoria y los índices de memoria que fueron liberados, respectivamente.

  Si el espacio liberado es mayor al que le faltaba al proceso para almacenarse completamente, entonces el atributo hayEspacio se vuelve verdadero y el atributo espacioFaltantes se vuelve cero nuevamente. En caso contrario, solamente se actualiza el atributo espacioFaltantes.

  Por último se usa el método actualizarPaginas() del proceso que se encontraba/encuentra en espera de almacenarse completamente, pasándole como parámetro el arreglo de índices de memoria desalojados.

- *ejecutarProceso():* Se modificó para que, en caso de que un proceso se encuentre almacenado en el atributo enEspera (lo que significa que aún no se almacena completamente) y otro proceso distinto finalice su ejecución, entonces se puedan asignar las localidades de memoria disponibles al proceso en espera, haciendo uso del método agregarPaginasFaltantes(), así como del método getCargado() de la clase Proceso, ya que si el proceso que finaliza es el que se encontraba en espera, entonces solamente se actualiza el atributo hayEspacio a verdadero y el atributo espaciosFaltantes a 0.
  Por último se utiliza el método actualizarLista() para actualizar la lista ligada de memoria.
- *aniquilarProceso():* Se le realizó la misma modificación que al método ejecutarProceso().
- *aniquilarTodo():* Solamente se modificó que se actualizara el atributo hayEspacio a verdadero y espaciosFaltantes a 0, pues toda la memoria ha quedado liberada.
- *desfragmentar():* Se le realizó la modificación en cuanto a la manera en cómo se realiza la desfragmentación.
  Primero, se obtienen los ids y páginas cargadas que tiene cada proceso que se encuentra en la cola de procesos listos, además de almacenarlos en otra queue, la cual servirá para ir desencolando los procesos y actualizar su tabla de páginas. Posteriormente, se recorrerá la tabla de memoria y se irán almacenando los procesos de manera contigua, almacenando su primer y última localidad de memoria, con la finalidad de generar sus marcos correspondientes y actualizar su tabla de páginas. En caso de que esté un proceso sin almacenarse totalmente en memoria, entonces primero se usa su método resetPaginas() antes de actualizar su tabla de páginas, con la finalidad de evitar discrepancias.
  Si ya no existen más procesos para almacenar en memoria y aún no se ha recorrido toda la tabla de memoria, entonces a todas estas localidades de memoria faltantes se les asigna un cero, que representa memoria libre.
  Por último se utiliza el método actualizarLista() para actualizar la lista ligada de memoria.

- BloqueMemoria: Esta clase sirve para modelar a cada uno de los bloques que compone a la lista ligada donde se almacenan los procesos y huecos en memoria para algún tiempo de ejecución. Cuenta con los siguientes atributos
  - *id:* Este atributo es de tipo char, y tomará los valores de 'H' o 'P', dependiendo de si el actual bloque representa a un hueco o a un proceso, respectivamente.
  - *registroBase:* Es un atributo de tipo entero y sirve para representar la localidad de memoria a partir de la cual empieza el actual bloque.
  - *longitud:* Es un atributo de tipo entero y sirve para almacenar la longitud de memoria que ocupa el actual bloque.

  Para los métodos, se tienen los siguientes:
  - BloqueMemoria(): Es el constructor de la clase.
  - toString(): Es una sobrecarga, y sirve para que, cuando se imprima en pantalla un objeto del tipo BloqueMemoria, se muestre su id, registro base y longitud.

Finalmente, se tiene el video donde se muestra la simulación del planificador de procesos junto con sus modificaciones y nuevas implementaciones:
https://drive.google.com/file/d/1pS6aD95mqufl6IXjveYMWhwFsHY9LeZD/view?usp=sharing

Conclusiones

Para esta práctica se pudo observar la relación e importancia de los conceptos revisados en clase, tanto para procesos como para la gestión de memoria.

Como ya se había revisado con anterioridad para la práctica 2, el planificador de procesos se encarga de gestionar a los diversos procesos que se encuentran en memoria, sin embargo, faltaban revisar conceptos importantes referentes a la parte de memoria, como lo es la paginación, reemplazo de páginas, etc, los cuales son una parte fundamental en cualquier S.O, pues representan la forma en cómo se almacenan a los procesos en memoria.

En principio, la memoria se asigna y libera de manera dinámica a través del tiempo, lo que significa que muchas veces pueden existir huecos entre procesos, los cuales, al no ser contiguos, no permitan almacenar un proceso de manera contigua, provocando la fragmentación externa, la cual se puede solucionar utilizando la paginación y la desfragmentación.

La paginación permite dividir a un proceso en páginas lógicas, las cuales para esta práctica tenían un tamaño de 16 localidades, y a su vez se lleva un registro de estas páginas junto con sus localidades de memoria físicas (marcos), lo que permite que un proceso se pueda almacenar sin que las localidades de memoria sean contiguas. En caso de que el proceso no se almacene totalmente en memoria, entonces se tendrán páginas sin marcos asignados, lo que provocará que cuando se esté ejecutando instrucciones del proceso que se encuentran en páginas no asignadas, exista un fallo de página, el cual se trata utilizando los diversos algoritmos de reemplazo de páginas.

La desfragmentación consiste en recorrer todos los procesos para que se almacenen de manera contigua y todos los huecos disponibles queden de manera continua.

Es importante recalcar que en todo momento se lleva un registro de los procesos y huecos almacenados en memoria, a través de una lista ligada.

A través de los diversos puntos solicitados en la práctica que fueron añadidos a la simulación de un planificador de procesos, se pudo reforzar lo analizado en teoría, además de comprender la utilidad de la paginación, lista ligada de la memoria, entre otros conceptos. Por otra parte, es importante remarcar que también presentan una dificultad para ser implementados, puesto que se necesita llevar un control más estricto de las localidades de memoria, cuando son ocupadas y liberadas, para que no existan conflictos al almacenar procesos de manera errónea.

Con esta práctica se visualizó de mejor manera lo que realizan nuestros S.O a mucho mayor escala para la gestión de procesos, y se comprendieron las dificultades que se presentan al momento de estar realizando dicha gestión. Dichas dificultades pueden ser tratadas en gran medida a través de diversos recursos y métodos, pero que muchas veces no se pueden llegar a solventar en su totalidad.