JAVA MEMORY MANAGER: MÉTODOS DE ASIGNACIÓN EN JAVA

Covarrubias Solís Víctor Alfonso

Rodríguez Torres Sebastián

Contenido

[Introducción 3](#_Toc151502428)

[Marco teórico 4](#_Toc151502429)

[1. Gestión de Memoria en Sistemas Operativos: 4](#_Toc151502430)

[2. Fragmentación Interna y Externa: 4](#_Toc151502431)

[3. Métodos Clásicos de Asignación de Memoria: 4](#_Toc151502432)

[4. Evaluación de Métodos: 4](#_Toc151502433)

[5. Desafíos Actuales: 5](#_Toc151502434)

[Implementación de los algoritmos 5](#_Toc151502435)

[Buddy 5](#_Toc151502436)

[Buddy 5](#_Toc151502437)

[Procesobuddy 6](#_Toc151502438)

[Automático 6](#_Toc151502439)

[Manual 9](#_Toc151502440)

[Excel 11](#_Toc151502441)

[Ajustes 11](#_Toc151502442)

[AjusteV 11](#_Toc151502443)

[Bloques 12](#_Toc151502444)

[ExportarExcel 13](#_Toc151502445)

[Proceso 13](#_Toc151502446)

[Resultados 14](#_Toc151502447)

[Conclusiones 17](#_Toc151502448)

[Covarrubias Solis Victor: 17](#_Toc151502449)

[Rodríguez Torres Sebastián: 17](#_Toc151502450)

[Conclusión Conjunta: 17](#_Toc151502451)

[Referencias 18](#_Toc151502452)

[Anexo de Responsabilidad 18](#_Toc151502453)

[1 Menu 1 13](#_Toc151501803)

[2 Menu2 14](#_Toc151501804)

[3 Pantalla Buddy Automatico y Manual 14](#_Toc151501805)

[4 Excel Buddy Automatico y Manual 15](#_Toc151501806)

[5 Primer, Proximo y Mejor Ajuste 15](#_Toc151501807)

[6 Excel de los Ajustes 16](#_Toc151501808)

# Introducción

En este proyecto, se aborda el desafío fundamental de la gestión eficiente de la memoria en un entorno de sistema operativo. La memoria es un recurso crítico en cualquier sistema informático, y su asignación y liberación deben manejarse cuidadosamente para optimizar el rendimiento y minimizar el desperdicio.

El objetivo principal de este proyecto es implementar y comparar cuatro métodos clásicos de asignación de memoria en Java: Primer Ajuste, Próximo Ajuste, Mejor Ajuste y el Sistema Buddy. Estos métodos representan diferentes enfoques para manejar la asignación y liberación de memoria, cada uno con sus propias características y desafíos.

La implementación en Java permitirá una comprensión práctica de cómo estos algoritmos operan en un entorno de programación real. A través de este proyecto, se explorarán los siguientes aspectos:

1. Primer Ajuste (First Fit):

- Se implementará el método de asignación de memoria que asigna el primer bloque disponible que sea lo suficientemente grande para satisfacer la solicitud del proceso.

2. Próximo Ajuste (Next Fit):

- Se desarrollará una implementación que mejora la eficiencia en comparación con el Primer Ajuste al comenzar la búsqueda desde la última ubicación asignada.

3. Mejor Ajuste (Best Fit):

- Se trabajará en un algoritmo que busca el bloque de memoria más pequeño que sea suficiente para satisfacer las necesidades del proceso, minimizando así la fragmentación externa.

4. Sistema Buddy:

- Se explorará la implementación del Sistema Buddy, que divide la memoria en bloques de tamaño potencia de 2 y utiliza un enfoque de asignación recursiva.

Cada uno de estos métodos será evaluado en términos de eficiencia, manejo de la fragmentación interna y externa, así como su rendimiento general en diversas situaciones simuladas. Además, se proporcionará una interfaz de usuario que permitirá interactuar con el sistema y observar cómo evoluciona la asignación de memoria a medida que se ejecutan los procesos.

Este proyecto no solo ofrece una oportunidad para la implementación práctica de algoritmos de gestión de memoria en Java, sino también para la comprensión profunda de los desafíos asociados con la asignación de memoria en entornos informáticos.

# Marco teórico

La gestión de memoria es un componente crítico en la arquitectura de sistemas operativos, ya que influye directamente en el rendimiento y la eficiencia del sistema. En un entorno computacional, la memoria se organiza y asigna para satisfacer las demandas de los procesos en ejecución. En este contexto, la implementación de métodos eficientes de asignación de memoria juega un papel crucial. A continuación, se proporciona un marco teórico que contextualiza la importancia de la gestión de memoria y presenta los métodos específicos abordados en este proyecto:

Marco Teórico Ampliado: Gestión de Memoria y Métodos de Asignación en Sistemas Operativos

## 1. Gestión de Memoria en Sistemas Operativos:

- Segmentación y Paginación: Dos enfoques comunes para organizar la memoria en sistemas operativos son la segmentación y la paginación. La segmentación divide la memoria en segmentos lógicos, mientras que la paginación la divide en páginas fijas.

- Memoria Virtual: La memoria virtual es una técnica que utiliza tanto la memoria RAM como el almacenamiento secundario para crear la ilusión de tener más memoria de la físicamente disponible.

## 2. Fragmentación Interna y Externa:

- Mitigación de Fragmentación: Estrategias como la compactación (reorganización de la memoria para eliminar huecos) y la asignación dinámica pueden ayudar a mitigar la fragmentación, pero introducen complejidad en la gestión de memoria.

## 3. Métodos Clásicos de Asignación de Memoria:

- Primer Ajuste (First Fit): Aunque es simple, puede resultar en una alta fragmentación externa y puede no ser el método más eficiente en términos de tiempo de búsqueda.

- Próximo Ajuste (Next Fit): Mejora la eficiencia en comparación con el Primer Ajuste al recordar la última ubicación asignada, pero aún puede generar fragmentación externa.

- Mejor Ajuste (Best Fit): Minimiza la fragmentación externa asignando el bloque de memoria más pequeño posible, pero puede llevar a una mayor fragmentación interna y requerir más tiempo de búsqueda.

- Sistema Buddy: Utiliza bloques de tamaño potencia de 2 y reduce tanto la fragmentación interna como externa, aunque puede resultar en un mayor desperdicio de memoria debido a la división en bloques de tamaño fijo.

## 4. Evaluación de Métodos:

- Carga del Sistema: La eficiencia de cada método puede depender de la carga del sistema y la dinámica de asignación de memoria en tiempo real.

- Políticas de Liberación: Además de la asignación, las políticas de liberación de memoria también influyen en la eficiencia general del sistema.

- Adaptabilidad: La capacidad de adaptarse a cambios en la demanda de memoria y en el tamaño de los procesos es esencial para un sistema operativo dinámico.

## 5. Desafíos Actuales:

- Seguridad: La seguridad de la memoria, incluida la prevención de desbordamientos de búfer y ataques de inyección de código, es una preocupación creciente en entornos informáticos modernos.

# Implementación de los algoritmos

Dentro del proyecto manejamos dos paquetes para los procesos y dentro de cada paquete su correspondiente clases y ventanas

## Buddy

El paquete buddy es el principal, es desde donde se ejecuta el programa y desde el cual se hace el acceso al sistema buddy y el traspaso al paquete Ajustes.

## Buddy

Dentro del paquete buddy existe una clase buddy la cual seria la principal dek proyecto ya que es la que contiene los menus del programa, para desplazarse por los menus se usa el Kyslistener el cual detecta la entrada por teclado del usuario.

Buddy();

Contructor vacio de buddy solo sirva para que al volver a el menú principal este comienze en el menú principal y no en un submenú.

main();

Función principal del proyecto, aquí se manda a llamar el método sistemabuddy.

sistemabuddy();

En este método se crea la ventana que usaran los menus.

menus();

aquí se añade el panel y los dos botones de los menus y por medio de un joptionpane se le informa al usuario como desplazarse por el menú principal.

keyPressed(KeyEvent event);

En este método se encuentra todo el funcionamiento de los menus. Lo primero que se hace aquí es checar en que menú esta ya que dependiendo del menú se hacen diferentes acciones, por ejemplo, el menú principal utiliza la flecha arriba y abajo para poder moverse y el enter para escoger entre el sistema buddy y el de ajustes.

Si te metes al distema buddy te sale un submenú el cual también usa las flechas para el desplazamiento entre opciones, pero el enter ya no se usa, ahora se usa la tecla space para seleccionar una opción.

Al escoger ajuste te envia directo a el ajuste sin entrar en algún submenú.

actionPerformed(ActionEvent event);

en este método se encuentran los pasos a seguir si se presiona uno de los dos botones, si se presiona el botón inicio te regresa al menu principal, esto sirve mas si te encuentras en el submenú de buddy.

Y el botón salir el cual termina la ejecución del programa.

Al final se encuentra una clase extra la cual se encuentra también en cada una de las otras clases de buddy, esta nos permite sobreescribir el método Paint de javax swing para que se muestre un fondo (imagen), esta se implemento para que no se tenga que esperar el tecleo del usuario para mostrar el fondo.

## Procesobuddy

En esta clase se encuentra lo necesario para la creación de los procesos, en esta parte se encuentran sus atributos los cuales son: tamaño, id, bloq, posbloq y fragmentación este ultimo a diferencia de los otros no es entero, fragmentación es double y guarda el porcentaje de fragmentación.

## Automático

En esta clase se encuentran los métodos necesarios para la simulación de un buddy que no necesita de la intervención del usuario, se usan puras cantidades aleatorias para la cantidad de memoria total y los procesos que la utilizaran.

* Automático y Automático(int i);
  + Constructores de la clase, uno sirve solo para crear el objeto de la clase, mientras que el otro, inicializa el sistema dándole un valor a la memoria total y llenando los arreglos que se usaran a lo largo del programa.
* divisiones(int tam);
  + Sirve para saber cuantas veces una cantidad se puede dividir entre dos.
* llenartamaños();
  + Aquí se llenan los arreglos los cuales son: tamaños(contiene los tamaños posibles que puede tener un bloque), canttamaños(contiene cuantos bloques hay de cada tamaño), ocupados(indica si el bloque de una posición está ocupado), tam(indica el tamaño del bloque de una posición), R,G,B(contienen número aleatorios del 0 al 244 y en conjunto crear colores en formato rgb), x(contiene las coordenadas en x de cada bloque, sirve para saber desde donde dibujarlo).
* buddy();
  + En este método se crea la ventana que se usara durante el programa, aquí mismo se llama al método memoria.
* memoria();
  + En este método se crea un contenedor en el cual se agregan los jpanel y los jtextarea, en los jpanel se muestran los procesos mientras que en los jtextarea se muestran las acciones de estos. Dentro de este proceso se encuentra un timer el cual es lo que permite que el programa se mueva automáticamente, funciona en base a intervalos, cada cierto tiempo manda llamar al método movimiento.
* keyPressed(KeyEvent event);
  + En este método se llevan a cabo los eventos del keylstener, en este caso el único evento que toma en cuenta es la utilización de la tecla esc la cual sirve como una especie de pausa, se se usa sale un joptionpane con 5 opciones: continuar (se descongela el nivel y continua), reiniciar (se usa el método reiniciar y se empieza de nuevo), reporte (se hace uso del método reporte), menú principal (cierra la ventana y se regresa al menú principal), salir (termina el programa por completo).
* movimiento();
  + En esta función se toma una elección con un random el cual saca 0 o 1, si sale 0 insertara un proceso, si sale 1 se borrará uno para insertar se crea un proceso entre el total de la memoria entre 32 hasta el total de la memoria entre 2 para que no sea ni muy poco, ni tanto, estas cantidades se envían a el método ocupar. Y cuando se elimina se manda a llamar el método quitar.
* ocupar(int proceso, Graphics papel);
  + Se crea un objeto de clase procesobuddy al cual se le asigna el tamaño creado en movimiento, una id y se checa si cabe en la memoria, si cabe se checa el tamaño de bloque que debería usar y con ello también se checa su fragmentación, después se checa que bloque del tamaño que necesita usara. Para ello primero usa el método estansubocupdos para saber si el bloque que podría usar tiene subbloques que ya están siendo ocupados, si está libre, se usa la función ocupar para que se ocupe el bloque que está libre y todos los subbloques, se guarda el atributo de posición y se dibuja en la memoria.

Aparte de dibujarse se agrega a dos arraylist: Procesos (guarda los procesos que se encuentran en ese momento en la memoria) y totalProcesos (guarda todos los procesos que pasaron alguna vez por la memoria principal)

Si el proceso no cupo, o no se encontró un bloque libre para el entonces se manda a una cola de espera, esta pude ser de corto, mediano o largo tiempo de espera, los procesos más pequeños (memoriatotal/16 o menor) se van a la cola corta y tendrán mayor prioridad, los medianos(memoria/4 o menor) se van a la cola mediana y por ultimo los grandes se van a la larga.

Estas colas son arraylist de procesos (Esperacorta, Esperalmediana, Esperalarga).

* subocupar(int posbloqocupado, int id, int tama);
  + Este fue uno de los más complicados de realizar ya que se recorre el array de ocupados buscando que posición se debe subocupar aparte de la que ya se ocupó, para esto se hacen varios cálculos los cuales cumplen la siguiente función:

Por ejemplo, si el tamaño de la memoria es 200

Entonces el arreglo tam estaría llenado asi: 200,100,100,50,50,50,50, etc y el de ocupados sirve para saber si están siendo usados.

Asi que, si se ocupa la posición 1, se deben ocupar la 3,4 y la 7,8,9,10 y asi, se puede notar que la diferencia entre 1 y 3 es 2 y la diferencia entre 4 y 7 es 3 y la siguiente seria 5 y luego 9 asi hasta el final del arreglo, pero si ponemos atención la diferencia entre estas diferencias va de 1, 2, 4, y se va duplicando, estos son los factores tomados en cuenta para la subocupación

Entonces comencé con una variable llamada pos que guarda (la posición a ocupar +(la posición a ocupar + 1)) por ejemplo en el ejemplo si la posición es la 1 entonces la variable guarda el número 3, también puse un contador que inicia en 1 y se va duplicando y otro contador que inicia en el 2 y también se va duplicando.

Dentro de un for que va desde 0 hasta las veces que se puede dividir entre 2 el tamaño del proceso, existe otro for el cual va desde 0 hasta el segundo contador, dentro de este for se le asigna a ocupados[pos+j](j es la variable del segundo for) a esta posición del arreglo ocupados se le asigna la id del proceso para que se ocupe, ¿Porque se hace esta suma? La suma se hace para que dentro del for se abarquen cada uno de los subbloques consecuentes por ejemplo pos en este caso vale 3 y el for va del 0 al 2 asi que se repite 2 veces entonces la posición 3 y la 4 se ocupan ya que la siguiente ves que se repite el for j vale 1 y se termina el for.

Dentro del primer for pos cambia ahora vale pos + (posición a ocupar + 1)\*(2\*cont) lo cual si seguimos con el ejemplo daría 7 3+ ((1+1)\*(2\*1))esto se hace para que la siguiente vez que se meta al for ocupados comience desde la posición 7 hasta la 9 y esto se repite hasta que ya no existen subbloques.

* estanocupados(int posicionaocupar, int tam);
  + Se usa el mismo procedimiento que en subocupar, pero esta vez en lugar de asignar la id al bloque que se va a ocupar, se checa si este ya es diferente de 0, si lo es retorna verdadero, si se llega al final y no se vio una psicion diferente de 0 se regresa falso para que se sepa que no hay procesos impidiendo la entrada del nuevo proceso.
* ocupardesdecola(int cola, Procesobuddy obj, Graphics papel);
  + Hace lo mismo que ocupar solo que en lugar de recibir el tamaño del proceso recibe un proceso desde una de las colas de espera.
* quitar(int quitproc, Graphics papel);
* Checa el arraylist que contiene los procesos que se encuentran en ese momento en la memoria principal y busca el que concuerde con la id mandada desde movimiento. Se elimina el proceso del arraylist Procesos y se dibuja para borrarlo de la memoria.

Enseguida de la eliminación se checa si hay procesos en el arraylist Esperacorta y se pasa a ocupardesdecola, si no hay se hace lo mismo, pero con la mediana y si no hay en la mediana se hace el proceso con la larga, si el proceso que se pasa a ocupardesdecola no encontró un bloque adecuado se queda en la cola y se regresa a movimiento para saber que hacer después.

* desocupar(int todo, int pos, Graphics papel);
  + Esta función hace lo contrario a ocupar y subocupar, además de que es mucho mas simple. Recorre el arreglo de ocupados y si una posición de este arreglo tiene la id del proceso a eliminar se le asigna 0.
* totalentre(int tam);
  + Esta función se usa al momento de dibujar en la memoria, checa cuantas veces se tiene que dividir entre dos para llegar a la cantidad dada, esto sirve para que al momento de dibujar el bloque se vea esta información escrita encima de este mismo.
* letra(int id);
  + Este método recibe la id del proceso y la convierte en char para mostrarla como letra, esto lo hace por medio del ascii.
* memocupada();
  + Esta función realiza una sumatoria de el atributo bloq de cada proceso en el arraylist Procesos para saber cuanto espacio se esta ocupando en la memoria.
* reiniciar();
  + Este método se puede usar cuando sea al darle pausa al programa, aquí se eliminan los elementos de los arraylist, se desocupan todo el arreglo de ocupados, se limpian los jtextarea, se reinicia la sumatoria de fragmentación y el contador de ids, luego se dibuja encima de la memoria para eliminar todos los bloques y también se dibuja por encima de las colas.
* reporte();
  + Este método sirve para crear un libro en Excel en el cual en cada hoja se encuentra una tabla con los elementos de cada arraylist. Para hacer esto crea un objeto de la clase Excel y usa el método exportarExxcel.
* dibujarcolas(Graphics papel2);
  + En este método se recorren los arraylist de colas de espera y por medio de fors se van dibujando los procesos que cada una de estas colas contiene.

## Manual

En esta clase se encuentra lo necesario para que el sistema buddy sirva mediante operaciones del mismo usuario. Se usan los mismos métodos que en automático solo que algunos de estos cambian. Además de que se usan mas herramientas como lo son: slider, jtextfield, botones y un combobox.

* Manual();
  + Esta clase solo tiene un constructor el cual esta vacío.
* inicio();
  + Este método hace lo que, hacia el segundo constructor del modo automático, solo que la asignación de memoria en lugar de hacerse aleatoria es escrita por el usuario por medio de un joptionpane.
* buddymanual();
  + Hace lo mismo que el buddy de automático solo que aquí aparte de mandar llamar al método memoria también se llama al método inicio.
* memoria();
  + Es igual al método de automático solo que aquí se tienen que agregar las herramientas extras, primero se agrega otro panel para separar lo anterior y lo nuevo. Se agrega el deslizante el cual como cantidad mínima tiene un 2 y como máxima la cantidad de memoria escogida por el usuario. Al final se agrega lo demás.

Hay varios métodos que son iguales a los que tiene Automático los cuales hacen exactamente lo mismo:

* keyPressed(KeyEvent event);
* divisiones(int tam);
* llenartamaños();
* ocupar(int proceso, Graphics papel);
* subocupar(int posbloqocupado, int id, int tama);
* estansubocupados(int posicionaocupar, int tam);
* ocupardesdecola();
* quitar(int quitproc, Graphics papel);
* desocupar(int todo, int pos, Graphics papel);
* totalentre(int tam);
* reiniciar();
  + Hace lo mismo que en automático solo que también tiene que vaciar el jcombobox.
* letra(int id);
* reporte();
* memocupada();

Se usan los mismos métodos ya que sirven para ambos.

* solicitarmemoria();
* Este es uno de los métodos nuevos, se usa para checar la cantidad que ingresa el usuario para la memoria por medio del joptionpane, este método es llamado desde el método inicio y lo primero que hace es preguntar que cantidad se desea usar entre un rango (10, 2000), el rango fue puesto de esta manera para que no sea ni muy poco ni tanto.

Si el usuario presiona cancelar en el joptionpane se le regresa a el menú principal, si presiona ok pero el valor no esta en el rango, o no es un numero se vuelve a pedir la cantidad.

* rellenarlista(ChangeEvent event);
  + En este método se llena el jcombobox el cual sirve para elegir el proceso a eliminar de la memoria, se va llenando en un for con los elementos del arraylist Procesos.
* stateChanged();
  + Este método sirve para el deslizante ya que cada vez que el deslizante cambia de valor el método detecta ese evento, en este caso es usado para que el textfield cambie su valor y muestre el valor del deslizante.
* actionPerformed(ActionEvent event);
  + Este método detecta cada vez que se presiona un botón, con los if se checa cual botón es, si es el de insertar se toma el valor del deslizante para el tamaño del proceso y se manda a llamar la función ocupar. Si el botón es el de quitar entonces se toma el valor del combobox y se manda a llamar el método quitar.

Se podría decir que este método es el sustituto al método movimiento de automático.

## Excel

Esta clase sirve para crear un reporte de los procesos del sistema buddy a un libreo de Excel.

* exportarExcel(ArrayList <Procesobuddy> totalprocesos, ArrayList <Procesobuddy> Esperacorta, ArrayList <Procesobuddy> Esperamediana, ArrayList <Procesobuddy> Esperalarga);
  + Este método resive todos los arraylist y crea una hoja en Excel para cada uno de estos, las tablas de cada hoja las va llenando con los atributos como columnas y por cada fila los procesos.

## Ajustes

El paquete Ajustes contiene todo lo necesario para mostrar en pantalla lo correspondiente a primer, próximo y mejor ajuste. En este paquete tenemos las siguientes clases y funciones por clase:

### AjusteV

Esta clase contiene el jframe el cual se muestra al usuario, es desde donde se hacen las creaciones de procesos principales.

Este jframe fue realizado con paleta de NetBeans y contiene 3 grupos de progressBar, 4 tablas, 4 botones y 2 textField

De manera global manejamos 4 variables globales, las cuales son los modelos para las tablas que se muestran.

* AjustesV();
  + Esta función es la básica para inicializar el jframe pero se modifico para desde aquí agregar los títulos de las 4 tablas
* Agregar();
  + En esta función agregamos un proceso a la tabla de procesos con los valores correspondientes
* bloquesInicio();
  + Aquí inicializamos las tablas de los bloques para que muestren el tamaño inicial de cada bloque
* modificarBloques();
  + En esta función se modifican los valores de las tablas de bloques para actualizar los valores cada que se ingresan los procesos al bloque.
* borrarTabla();
  + Esta modifica la tabla de procesos para que cuando un proceso se elimina este marque un estado de inactivo
* jButton1ActionPerformed(java.awt.event.AtcionEvent evt);
  + Dentro de esta función se crean los objetos de los procesos al insertarlos en memoria. El tamaño del proceso se ingresa por un text field. Se valida que el valor ingresado sea entero  
    Después valida si es posible ingresar el proceso en algún bloque, en caso de ingresarlo crea 3 procesos, uno para cada tipo de ajustes y después se le ingresan los valores a cada proceso.  
    Y se manda llamar las funciones agregar() y modificarBloques()
* jButton3ActionPerformed(java.awt.event.ActionEvent evt);
  + Para borrar un proceso este verifica si se encuentra activo, si se encuentra entonces manda llamar las funciones Borrar, Borrar2 y Borrar3 dentro de la clase Bloques, este quitara el proceso del bloque  
    El objeto creado no se eliminara, ya que este se usa posteriormente para realizar un reporte, el objeto únicamente es desactivado y quitado del bloque
  + De igual manera mandamos llamar las funciones agregar() y modificarBloques()
* jButton4ActionPerformed(java.awt.event.ActionEvent evt)
  + Esta función realiza lo mismo que la función jButton1ActionPerformed con la ligera diferencia que este no recibe el tamaño de un text field, el tamaño es asignado de manera aleatoria.
* jButton2ActionPerformed, jButton5ActionPerformed, jButton6ActionPerformed, jButton7ActionPerformed
  + Estas cuatro funciones mandan llamar a la clase ExportarExcel, al mandarle la tabla correspondiente esta se exportara a un archivo .xls
* JButton8ActionPerformed
  + Esta función quita la ventana de AjustesV, crea un objeto del tipo Buddy y lo ejecuta, para regresar al menú principal

### Bloques

Esta clase es la cual almacena la información relacionada a cada bloque en el conjunto de memoria.

De manera global se manejan las siguientes variables:

Tamanio1, usado1, libre1, partition1, tamanio2, usado2, libre2, partition2, tamanio3, usado3, libre3, partition3. 12 son arreglos de tamaño 10

Ids, total.

Y 3 ArrayList de procesos para guardar los procesos en cada tipo de ajuste

* BloquesAsignacion();
  + En este inicializamos todas las variables de la clase, los tamaños son asignados de manera aleatoria.
* Búsqueda(int espacio);
  + Esta función busca el bloque en el cual se almacenara un proceso en caso de ser primer ajuste
* Busqueda2(int espacio);
  + Esta función recorre todos los bloques y verifica que al menos exista uno en el cual se pueda agregar el proceso
* BusquedaProximo(int espacio);
  + Esta función busca el bloque en el cual almacenarse en base al ultimo proceso ingresado
* BúsquedaMejor(int espacio);
  + En esta función recorremos dos veces el conjunto de bloques, buscando el bloque en el que exista el menor desperdicio de memoria al ingresar un proceso, después lo volvemos a recorrer pero esta vez se busca directamente el bloque para retornar esa posición
* Borrar(int id);Borrar2(int id);Borrar3(int id);
  + Esta función borra un proceso, recorremos un arraylist y si el id coincide con el parámetro que recibimos entonces realiza las modificaciones pertinentes y ajusta los valores en las progessBar

### ExportarExcel

Esta clase únicamente tiene una función la cual recibe como parámetro un jTable junto con todo su contenido.

En esta función se hace uso de la librería extra aplicada, la cual es poi-3.7. Aquí se crea un objeto del tipo JFileChooses, después se usa el filtro para archivos Excel y se crea la configuración del diálogo de selección de archivos.

Se crea la ruta donde se guardara, se crea el archivo, un libro de trabajo y una hoja de trabajo. Comenzamos escribiendo los encabezados y después con otro for el resto de valores en las filas

### Proceso

Esta clase es a partir de la cual creamos los objetos de cada proceso ingresado, los parámetros que tiene son: tamanio, bloque, id, activo, tipo de ajuste y cola.

Estos valores son asignados en la clase AjustesV desde la paleta.

Las funciones InsertarPri, InsertarPro e InsertarM trabajan de la misma manera. Únicamente cambia las variables sobre las que trabajan.

Lo que hacen es agregar el proceso al arraylist de la clase Bloques correspondiente, asigna el tipo de ajuste y después recibe el tamaño del bloque usado, este valor lo divide entre el tamaño del bloque y multiplica por 100. Esto lo hace para asignarle el valor a la progressBar para mostrarlo en pantalla.

Mediante un switch seleccionamos el bloque que se va a modificar.

# Resultados

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Menu 1

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Menu2

Captura de pantalla de un videojuego

Descripción generada automáticamente

Pantalla Buddy Automatico y Manual

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Excel Buddy Automatico y Manual

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Primer, Proximo y Mejor Ajuste

Tabla

Descripción generada automáticamente

Excel de los Ajustes

# Conclusiones

## Covarrubias Solis Victor:

Durante este curso de sistemas operativos aprendí bastantes cosas de las cuales tenía una ligera idea, pero no sabía realmente como funcionaban como por ejemplo no conocía sobre los modelos de estado de los procesos en este tema vimos cómo era el ciclo de vida de un proceso desde que se creaba hasta que de alguna forma llegaba a ser ejecutado (si es que no se ignoraba y se quedaba en estado de inanición). El curso en su mayoría era teoría lo cual en cierta parte estaba bien, pero también creo que eso dificulta un poco el que se nos quedaran las cosas en la cabeza, por lo que el añadir actividades de vez en cuando creo que era un buen acierto.

Cuando se agregaron las actividades en equipo creo que la clase se volvió más dinámica lo cual también favorecía que las personas que casi no participaran trataran de hacerlo. Aunque la calificación grupal debería ser un poco más individual, me refiero a que solo deberían recibir la calificación ganada por el equipo los que contribuyeron a conseguir esa calificación.

## Rodríguez Torres Sebastián:

Durante el curso logramos entender el funcionamiento interno de un sistema operativo, desde la gestión de ejecución de procesos, hasta la gestión de almacenamiento de información en ficheros.

Esta información nos ah ayudado también a en otras materias que ocupan conocimientos sobre esta gestión de recursos. Principalmente en la materia de PIC 1: Linux Essentials. Por lo que considero que es una materia de suma importancia para el desarrollo de prácticas para otras materias.

## Conclusión Conjunta:

También esta buena la dinámica de exentar si demuestras que te interesa la clase, creo que motiva a poner atención y participar en clase.

En conclusión, el curso está bien, la maestra si tiene los conocimientos y si enseña bien, resuelve dudas y pone actividades las cuales ayudan a repasar lo teórico y de vez en cuando lo práctico. Los temas son prácticos y se relacionan entre sí por lo que si llegas a entender en profundidad como se relacionan entre sí. Está bien que no se le sobrecargue al alumno con tareas como otros cursos.

# Referencias

William Stalings, Sistemas Operativos (5a ed.) México: Prentice-Hall

(MVN Repository, 2010)

# Anexo de Responsabilidad

|  |  |  |  |
| --- | --- | --- | --- |
| Actividad Asignada | Actividad Realizada | Responsable | % de Cumplimiento |
| Creación de pantalla y presentación de primer, próximo y mejor ajuste | Creación de pantalla y presentación de primer, próximo y mejor ajuste | Rodríguez Torres Sebastián | 100% |
| Creación de pantalla en representación del proceso buddy | Creación de pantalla en representación del proceso buddy | Covarrubias Solís Víctor Alfonso | 100% |
| Menu de inicio | Menu de inicio | Covarrubias Solís Víctor Alfonso | 100% |
| Creación de la presentación | Creación de la presentación | Rodríguez Torres Sebastián | 100% |
| Creación de documentación | Creación de documentación | Covarrubias Solís Víctor Alfonso y Rodríguez Torres Sebastián | 100% |