



UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

Reporte de Proyecto

“Simulador de Buddy System y Round Robin”



Universidad autónoma
de Aguascalientes.

Centro de Ciencias
Básicas

Ingeniería en Sistemas
Computacionales

Aguascalientes, Ags.

Índice

Introducción	2
Implementación Buddy System	8
Implementación Round Robin	8
Conclusiones	9
Bibliografía	10
Anexo A	11

Introducción

En el contexto actual de la rama informática, la gestión eficiente de los recursos del sistema se ha convertido en un objetivo particular para optimizar el rendimiento de las aplicaciones y garantizar la satisfacción de los usuarios. Entre los aspectos críticos de lo anterior, la gestión de la memoria y la planificación de procesos destacan como áreas esenciales para el funcionamiento coherente de un sistema operativo. En este contexto, el presente proyecto se centra en dos técnicas clave: el sistema Buddy para la colocación de memoria y el algoritmo Round Robin para la planificación de procesos.

La colocación de memoria es un aspecto crítico en la administración de recursos, ya que afecta directamente la velocidad y la eficiencia del acceso a los datos. El sistema Buddy, una técnica de asignación de memoria basada en la subdivisión recursiva y ha demostrado ser una estrategia valiosa para la gestión eficiente de bloques de memoria, minimizando la fragmentación y optimizando el uso del espacio.

El sistema Buddy, también conocido como "Buddy System" o "Buddy Allocator," es una técnica de administración de memoria que aborda eficientemente el problema de la fragmentación interna en sistemas operativos y sistemas de gestión de memoria. Su enfoque se basa en la subdivisión recursiva de bloques de memoria, comenzando con bloques de tamaño potencia de 2 y dividiéndolos en "buddies" (compañeros) iguales cuando es necesario obtener bloques más pequeños.

En un Buddy System, los bloques de memoria disponibles son de tamaño 2^K , para valores de K tal que $L \leq K \leq U$ y donde: 2^L = tamaño de bloque más pequeño assignable, 2^U = tamaño de bloque más grande assignable (generalmente, 2^U es el tamaño de memoria completa disponible para asignación o gestión).

Inicialmente, el espacio entero disponible para la asignación se trata como un solo bloque de tamaño 2^U . Si se hace una solicitud de tamaño s tal que $2^{U-1} < s \leq 2^U$, entonces el bloque entero se asigna. En otro caso, el bloque se divide en dos colegas (buddies) de igual tamaño 2^{U-1} . Si $2^{U-2} < s \leq 2^{U-1}$ entonces la solicitud se asigna a uno de los dos buddies. Si no, uno de los cbuddies se divide por la mitad nuevamente. Este proceso continua hasta que el bloque más pequeño sea mayor o igual que s, generándose y asignándose a la solicitud. En cualquier instante, el Buddy System mantiene una lista de huecos (bloques no asignados) para cada tamaño 2^i . Un hueco puede eliminarse de la lista ($i + 1$) dividiéndolo en dos mitades para crear dos colegas (buddies) de tamaño 2^i en la lista i. Cuando una pareja de buddies de la lista i pasa a estar libre, se elimina de la lista y se unen en un solo bloque de la lista ($i + 1$).

Dada una solicitud para una asignación de tamaño k, tal que $2^{i-1} < k \leq 2^i$, para encontrar un hueco de tamaño 2^i se utiliza el siguiente algoritmo recursivo:

```

Conseguir_hueco(i)
{
    if (i = (U + 1))
        Fallo;
    if (lista I vacía)
    {
        Conseguir_hueco(i + 1);
        Dividir el hueco en colegas (buddies);
        Poner colegas (buddies) en lista i;
    }
    Coger el primer hueco en la lista i;
}

```

La mejor estructura de datos para implementar este esquema de partición es un árbol binario, donde los nodos hoja representan la partición actual de la memoria, si dos buddies están en nodos hoja, entonces al menos uno de ellos está asignado; en otro caso, se liberaría para que quedara sólo el nodo padre de ambos (se unirían en un bloque mayor)

El kernel debe desarrollar un método efectivo y sólido para asignar grupos de marcos de páginas cercanos. El objetivo principal de Buddy System es, por lo tanto, evitar la fragmentación externa. Este es un fenómeno que ocurre cuando grupos de marcos de página contiguos de diferentes tamaños se asignan y liberan con frecuencia; esto podría resultar en una situación en la que varios bloques pequeños de marcos de página libres se dispersan dentro de bloques de marcos de página asignados. Como resultado, es posible que sea imposible asignar un bloque considerable de marcos de páginas contiguos, incluso si hay suficientes páginas disponibles para satisfacer la solicitud.

El kernel opta por la asignación de marcos de página contiguos por varias razones:

1. Necesidad de Contigüidad en Algunos Casos: En situaciones específicas, es necesario que los marcos de página sean contiguos. Por ejemplo, cuando se asigna un buffer para el acceso directo a memoria (DMA), donde la contigüidad es esencial para la transferencia eficiente de datos, como en operaciones de entrada/salida (E/S) que involucran la lectura de varios sectores de disco en una sola operación.
2. Evitar Modificaciones en las Tablas de Páginas del Kernel: Al asignar marcos de página contiguos, no se modifican las tablas de páginas del kernel. Esto evita la necesidad de limpiar el contenido de las Translation Lookaside Buffers (TLBs) de la CPU, lo que mejora la eficiencia del sistema.
3. Acceso Eficiente a Grandes Porciones de Memoria: El uso de páginas de 4 KBytes permite al kernel acceder eficientemente a grandes porciones de memoria física que son contiguas, facilitando operaciones que requieren un acceso rápido a extensiones significativas de memoria.

En cuanto al manejo de la asignación y liberación de marcos de página, el kernel implementa el principio del Buddy System, que organiza los marcos en grupos de tamaños fijos (1, 2, 4, 8, ..., 512 marcos de página). Cuando se solicita la asignación de marcos, se elige un grupo de tamaño inmediatamente superior al requerido y se divide para satisfacer

la solicitud. Al liberar marcos, el kernel intenta fusionarlos con grupos adyacentes para formar grupos más grandes, optimizando así la gestión de la memoria. Este enfoque simplifica la asignación y liberación de marcos de página al permitir solo tamaños predefinidos que coinciden con los tamaños gestionados en las listas del Buddy System.

La asignación de memoria en el sistema Buddy se realiza de manera eficiente, buscando el bloque más pequeño que pueda satisfacer la solicitud y dividiéndolo en bloques más pequeños si es necesario. Este proceso de subdivisión recursiva continúa hasta obtener el tamaño de bloque adecuado. Por otro lado, cuando se libera un bloque de memoria, el sistema Buddy verifica si su "buddy" también está libre, fusionándolos si es el caso. Este proceso de fusión se repite recursivamente hasta que ya no sea posible fusionar. Un ejemplo de esta explicación se puede observar en la Figura1.

Buddy System (Alocación)

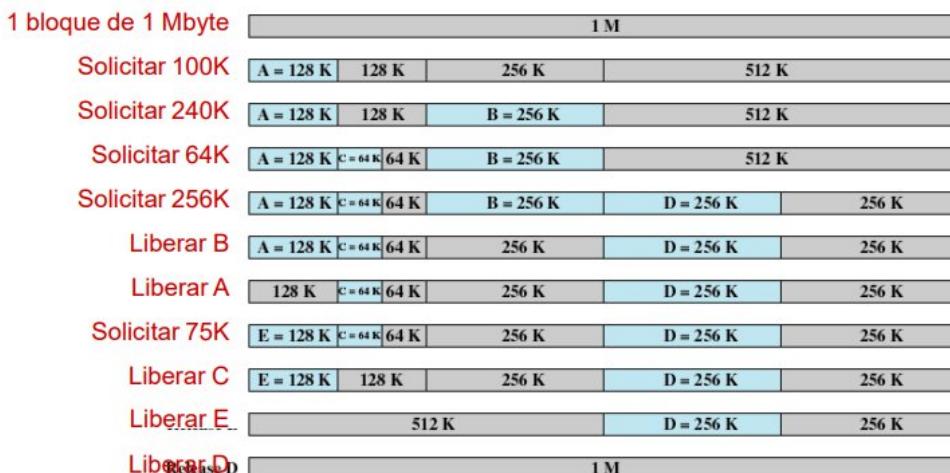


Figura 1. Ejemplificación de la alocación de memoria Buddy

Por otro lado, la planificación de procesos es esencial para garantizar que los múltiples procesos en ejecución compartan los recursos de manera justa y eficiente. El algoritmo Round Robin, un enfoque de planificación de procesos basado en turnos cíclicos, busca lograr una distribución equitativa del tiempo de CPU entre los procesos activos.

El algoritmo Round Robin es una técnica de planificación de procesos utilizada en sistemas operativos para asignar tiempo de CPU a múltiples procesos de manera equitativa. A diferencia de otros algoritmos de planificación, Round Robin se basa en un enfoque de turnos cíclicos, donde cada proceso tiene la oportunidad de ejecutarse durante un intervalo de tiempo predefinido, conocido como "quantum" o "ciclo de reloj."

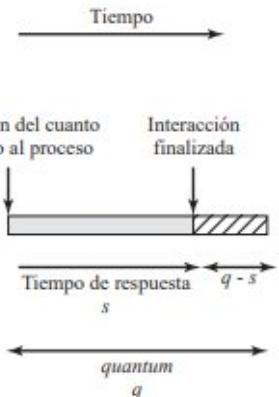
La expulsión basada en el reloj es una forma directa de reducir el castigo que tienen los trabajos cortos con FCFS ó "First-Come-First-Serve" (Primero en llegar, primero en ser

servido). La política de turno rotatorio, también conocida como Round Robin, es la más sencilla. Se produce una interrupción de reloj en intervalos regulares. Según la política FCFS, cuando ocurre una interrupción, el proceso actual en ejecución se sitúa en la cola de listos y se selecciona el siguiente trabajo. Como cada proceso se le da una rodaja de tiempo antes de ser expulsado, esta técnica también se conoce como cortar el tiempo (time slicing).

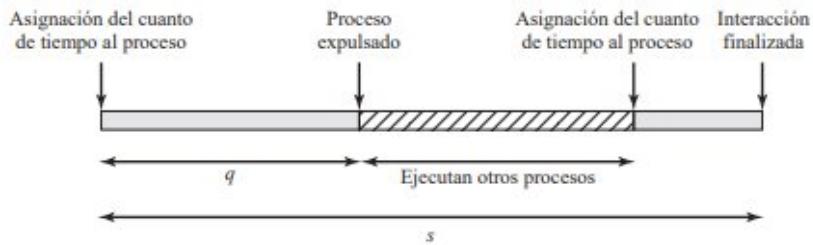
Con la planificación Round Robin , el tema clave de diseño es la longitud del quantum de tiempo, o rodaja, a ser utilizada. Si el quantum es muy pequeño, el proceso se moverá por el sistema relativamente rápido. Por otra parte, existe una sobrecarga de procesamiento debido al manejo de la interrupción de reloj y por las funciones de planificación y activación. De esta forma, se deben evitar los quantums de tiempo muy pequeños. Una buena idea es que el quantum de tiempo debe ser ligeramente mayor que el tiempo requerido para una interacción o una función típica del proceso. Si es menor, muchos más procesos necesitarán, al menos, dos quantums de tiempo. La **Figura 9.6** muestra el efecto que tiene en el tiempo de respuesta. Nótese que en el caso extremo de un quantum de tiempo mayor que el proceso más largo en ejecución, la planificación en turno rotatorio degenera en FCFS.

En sistemas de tiempo compartido de propósito general o de procesamiento transaccional, la planificación Round Robin es particularmente efectiva. Una desventaja de esta planificación es que trata de manera diferente a los procesos limitados por el procesador y la E/S. Los procesos limitados por la E/S generalmente tienen ráfagas de procesador más cortas que los procesos limitados por el procesador.

Si hay una combinación de los dos tipos de procesos, ocurrirá lo siguiente: un proceso limitado por E/S usa el procesador durante un breve período y luego se bloquea; espera a que complete la operación de E/S y luego se une a la cola de listos. Por otro lado, un proceso limitado por el procesador generalmente usa la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de listos. De esta manera, los procesos limitados por el procesador suelen recibir una rodaja de tiempo inequitativa del procesador, lo que resulta en un mal rendimiento de los procesos limitados por el procesador, un uso ineficiente de los recursos del procesador y un aumento en la varianza en el tiempo de respuesta



(a) Cuanto de tiempo mayor que la interacción típica



(b) Cuanto de tiempo menor que la interacción típica

Figura 9.6. Efecto del tamaño del *quantum* de tiempo de expulsión.

En la operación del algoritmo Round Robin, los procesos se colocan en una cola circular, y el proceso en la parte frontal de la cola obtiene la CPU durante un intervalo de tiempo determinado. Después de que se agota este tiempo, el proceso se mueve al final de la cola y el siguiente proceso en la cola obtiene su turno. Este ciclo se repite continuamente, asegurando que todos los procesos tengan la oportunidad de ejecutarse en un orden justo y equitativo.

La principal ventaja del algoritmo Round Robin es su equidad en la distribución del tiempo de CPU entre los procesos. Cada proceso recibe una porción igual de tiempo de ejecución, lo que es especialmente beneficioso en entornos de sistemas multitarea, donde varios procesos compiten por los recursos de la CPU. Esta equidad contribuye a un rendimiento balanceado del sistema y evita que un proceso monopolice la CPU indefinidamente.

Sin embargo, el algoritmo Round Robin también presenta algunos problemas. En particular, puede haber un desperdicio de tiempo de CPU si un proceso no utiliza todo su quantum asignado. Este desperdicio puede afectar la eficiencia general del sistema. Además, en situaciones donde los procesos tienen diferentes requerimientos de tiempo de CPU, el Round Robin puede no ser la opción más eficiente, ya que asigna el mismo quantum a cada proceso independientemente de sus necesidades.

Este proyecto busca explorar a fondo estas dos técnicas, analizando sus principios fundamentales, características, ventajas y cómo se pueden implementar en un sistema como simulador de planificación de procesos. Además, se examinarán comparativas de rendimiento para evaluar estas técnicas en diversas situaciones y entornos de sistemas operativos.

Implementación Round Robin:

Antes de la implementacion de Round Robin se implementó la declaracion y la estructura de los procesos, habiendo para cada proceso, su id, su tamaño y finalmente su quantum de tiempo de ejecucion, luego de declararlos y generarlos aleatoriamente, cada proceso entra a una cola que se declara como vector.

Al ejecutarse el proceso en la posicion inicial de la cola, este se manda al final de la cola y reduce su quantum de tiempo de modo que se le resta el quantum de tiempo del sistema, nuevamente se mete otro proceso, y se ejecuta el siguiente en la posicion inicial de la cola.

Cabe destacar que al inicio se presentaron problemas para presentar Round Robin debido a un malentendido, antes se implementaba de manera que el primero en entrar era el primero que se ejecutaba (FIFO), pero eso no es asi, ya que se ejecuta siempre el proceso que este al inicio de la cola de listos.

Sin embargo al implementar Round Robin podian entrar infinidad de procesos debido a que no habia un limitante, es decir, la memoria, ademas de esto no habia una representacion clara de la memoria y solo se veian los procesos entrando.

Implementación Buddy System:

La implementacion de Buddy System fue dura y ardua, habia una clara representacion de como habia que implementarlo pero se tuvo que dar muchas vueltas con lapiz y papel para encontrar la logica detras de lo que sucede al dividir la memoria a la mitad para encontrar el tamaño adecuado a cada proceso, ademas de liberar la memoria cuando un proceso llega a 0 en su quantum de ejecucion y por ultimo, condensar la memoria para volver al tamaño original si es que los bloques compañeros estan libres.

Una vez hecha la logica a mano, se penso en implementar Buddy System como un ARBOL, ya que al dividirse, se crea un bloque izquierdo, y un bloque derecho, ademas esto facilitaria muchisimo la tarea de condensar la memoria, ya que solamente se comprueba si bloque->izquierda y bloque->derecha estan libres, estando libres simplemente se eliminan y queda el bloque original.

Se presentaron varios problemas tales que:

- Los procesos se alojaban en espacios YA ocupados.
- Los procesos al añadirse en otros espacios, todos eran representados por un solo proceso.
- Los procesos sobrecargaban el limite de la memoria
- Los bloques no se liberaban

Para resolver estos problemas simplemente se resolvio la logica representandola por escrito, y al final luego de volver a programar, el codigo logró servir, muchos de los problemas que habia eran problemas de codigo, es decir, la logica estaba bien, pero la manera en que se programó no lo estaba, por ejemplo; se descubrió que los procesos a los que se referenciaban, no se guardaban dentro de la memoria, por eso mismo se

alojaban en un espacio ya ocupado, ademas de esto, la memoria no se liberaba por que los bloques no se estaban eliminando de una manera correcta.

Otra cosa que ocurría es que los bloques que se dividían no se declaraban como bloqueados, por lo cual, cuando un proceso entraba, se alojaba en un bloque que no existía dentro de la representación, pero esto no se pudo ver de manera clara, ya que al imprimir la memoria, solo se ven los bloques YA DIVIDIDOS, y los bloques que se dividieron, se reemplazan por sus hijos, por eso mismo no se pudo observar bien que es lo que ocurría.

Para implementar la liberación de memoria se hizo el uso de 2 funciones, una función que detecta el bloque por el id de un proceso y limpia el bloque ocupado por ese proceso, y la otra que condensa la memoria, este comienza desde arriba y va hacia abajo, haciendo uso de la recursividad, ya que la función se llama así misma por el bloque->izquierda y bloque->derecha.

Para implementar la división de memoria y el alojamiento de los procesos, se hizo el uso de una función booleana, la cual retorna verdadero si es que el proceso logró alojarse en un hueco, y retorna falso si es que el proceso no pudo entrar.

Las condiciones que tienen que comprobarse para que un proceso entre son:

- Que el hueco tenga igual o mayor tamaño que el tamaño del proceso
- Que el hueco donde se aloje este libre

Y por último la última condición es si el tamaño de proceso es menor o igual al tamaño del hueco entre 2, de ser así, la función de dividir memoria vuelve a dividir para encontrar un tamaño adecuado.

Conclusión General

Este proyecto nos ha demostrado ser una exploración de las técnicas de planificación de procesos y gestión de memoria en sistemas operativos, específicamente mediante la implementación de los algoritmos Round Robin y Buddy System. A través del desarrollo y la integración de estas técnicas, el equipo no solo ha mejorado su conocimiento técnico y habilidades de programación, sino que también ha adquirido una comprensión más profunda de los principios de los sistemas operativos.

La implementación de Round Robin resaltó la importancia de una asignación de tiempo justa y equitativa entre los procesos, mientras que el desafío de implementar el sistema Buddy nos proporcionó una visión clara de la gestión de memoria eficiente y dinámica. Juntos, estos sistemas demostraron cómo la planificación de procesos y la gestión de memoria pueden trabajar en conjunto para mejorar el rendimiento y la eficiencia de un sistema operativo.

Además, se pudo observar la relevancia de las habilidades de trabajo en equipo, comunicación y gestión del tiempo. La utilización de herramientas de control de versiones usando Github y la colaboración efectiva fueron fundamentales para el éxito del proyecto. Al enfrentar desafíos tanto en el código como en la conceptualización, el equipo aprendió a adaptarse y a abordar problemas complejos de manera colectiva.

En resumen, este proyecto no solo ha sido un ejercicio en programación avanzada y teoría de sistemas operativos, sino también en colaboración y solución de problemas en un entorno de equipo. La integración exitosa de Round Robin y Buddy System, junto con una interfaz de usuario amigable y adaptable, refleja nuestro compromiso y la capacidad del equipo para superar desafíos técnicos y trabajar juntos hacia un objetivo común.

Bibliografía

1. Stallings, William, "Sistemas Operativos" (Libro de Texto), Pearson Education
 2. Tanenbaum, Andres, "Sistemas Operativos. Diseño e implementación", Prentice-Hall, ISBN: 9701701658
 3. Deitel, Harvey M. "Introducción a los Sistemas Operativos", Addison-Wesley, ISBN 84-205-3177-
 4. Silberschatz, A., Galvin, P. B., Gagne, G., & Allende, J. S. (2006). Fundamentos de sistemas operativos.
 5. Milenkovic, Milan "Sistemas Operativos (Conceptos y Diseños)", McGraw Hill ISBN 84-481-1871-5
- <http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/SOF.htm>: Sistemas Operativos – Mgter. David Luis la Red Martínez – UNNE – FACENA.
 - <http://members.fortunecity.es/lrmdl/>: Sistemas Operativos – Mgter. David Luis la Red Martínez
 - <http://www.infor.uva.es>: Universidad de Valladolid - Departamento de Informática (España).
 - <http://www.cs.vu.nl/~ast/>: Andrew S. Tanenbaum - Sistemas Operativos Modernos

Anexo A

MÁQUINA VIRTUAL: como ya se cuenta con una máquina virtual para el proyecto, simplemente se tienen que seguir los siguientes pasos ya que el proyecto ya está compilado dentro de la máquina virtual.

1. Importar la máquina virtual descargada desde el link de onedrive



2. Abrir la máquina virtual
3. Abrir la terminal del sistema y ubicarse en el directorio /home/documentos/Proyecto_Sistemas

```
cd Documentos/  
entos  
~/Documentos> cd Proyecto_Sistemas/  
entos/Proyecto_Sistemas
```

4. Escribir ./BR_Linux, pulsar enter y disfrutar del proyecto