Trabajo Práctico 2

Introducción a los Sistemas Operativos

1.
    a. Un **Sistema Operativo (SO)** es el software fundamental que actúa como intermediario entre el hardware de una computadora y el usuario o las aplicaciones. Su objetivo principal es gestionar los recursos del sistema (CPU, memoria, dispositivos de E/S) de manera eficiente y proporcionar un entorno en el que los programas se puedan ejecutar.
    b. Para que un sistema operativo pueda cumplir sus objetivos, necesita interactuar con los siguientes componentes de hardware esenciales:
        - **CPU (Unidad Central de Procesamiento):** Ejecuta las instrucciones del SO y de los programas.
        - **Memoria Principal (RAM):** Almacena temporalmente los datos y las instrucciones que la CPU necesita para trabajar.
        - **Dispositivos de Almacenamiento Secundario:** (Discos duros, SSD) Almacenan el SO, los programas y los datos de forma permanente.
        - **Dispositivos de Entrada/Salida (E/S):** Permiten la interacción con el usuario (teclado, mouse) y otros sistemas (tarjetas de red).
        - **Temporizador (Timer):** Permite al SO recuperar el control de la CPU para evitar que un proceso monopolice el sistema.
    c. Los componentes software principales de un sistema operativo incluyen:
        - **Kernel (Núcleo):** Es el corazón del SO, gestionando las tareas más críticas como la planificación de procesos y el acceso al hardware.
            - **Gestión de Procesos:** Se encarga de crear, eliminar, suspender y reanudar procesos, así como de la comunicación y sincronización entre ellos.
            - **Gestión de Memoria:** Asigna y libera espacio en la memoria RAM para los procesos que lo necesiten.
            - **Gestión del Sistema de Archivos:** Organiza los archivos en los dispositivos de almacenamiento y controla su acceso.
            - **Gestión de Entrada/Salida (E/S):** Administra la comunicación con los dispositivos periféricos.
        - **Interfaz de Usuario:** Puede ser una línea de comandos (CLI) o una interfaz gráfica (GUI), permitiendo al usuario interactuar con el SO.
    d. Una **llamada al sistema** es el mecanismo que utiliza un programa para solicitar un servicio al kernel del sistema operativo. Es la interfaz fundamental entre un proceso y el SO.
    Se implementan mediante **interrupciones por software** (traps). Cuando un programa ejecuta una system call, la CPU pasa del modo usuario al **modo kernel**, permitiendo al SO ejecutar la tarea privilegiada solicitada (como leer un archivo o crear un proceso) de forma segura.

e. **Programa:** Es una entidad **pasiva**. Es un conjunto de instrucciones escritas en un lenguaje de programación y almacenadas en un archivo ejecutable en disco. **Proceso:** Es una entidad **activa**. Es una instancia de un programa en ejecución. Un proceso incluye no solo el código del programa, sino también su estado actual, como el valor del contador de programa, los registros de la CPU y los datos en su espacio de memoria.

f. La información mínima que el Kernel debe mantener sobre un proceso se almacena en una estructura de datos llamada **Bloque de Control de Proceso (PCB - Process Control Block)**. Esta incluye:

- **Identificador del Proceso (PID):** Un número único para identificar al proceso.
- **Estado del Proceso:** (ej. listo, en ejecución, esperando).
- **Contador de Programa (PC):** La dirección de la próxima instrucción a ejecutar.
- **Registros de la CPU:** Valores de los registros para restaurar el proceso.
- **Información de planificación:** Prioridad del proceso.
- **Información de gestión de memoria:** Punteros a las tablas de páginas o segmentos.
- **Información de E/S:** Dispositivos asignados al proceso, archivos abiertos.

g. Los algoritmos de planificación buscan optimizar el uso de la CPU y la experiencia del usuario, persiguiendo los siguientes objetivos:

- **Justicia (Fairness):** Asegurar que cada proceso reciba una porción justa de tiempo de CPU.
- **Maximizar la utilización de la CPU:** Mantener la CPU ocupada el mayor tiempo posible.
- **Maximizar el Throughput:** Completar la mayor cantidad de procesos por unidad de tiempo.
- **Minimizar el Tiempo de Retorno (Turnaround Time):** Reducir el tiempo total que tarda un proceso desde que llega hasta que finaliza.
- **Minimizar el Tiempo de Espera (Waiting Time):** Reducir el tiempo que un proceso pasa en la cola de listos.
- **Minimizar el Tiempo de Respuesta (Response Time):** Reducir el tiempo desde que se hace una solicitud hasta que se produce la primera respuesta.

h. **No Apropiativo (Non-Preemptive):** Una vez que la CPU se le asigna a un proceso, este la conserva hasta que la libera voluntariamente, ya sea porque terminó su ejecución o porque está esperando una operación de E/S. **Apropiativo (Preemptive):** El sistema operativo puede "quitarle" la CPU a un proceso en ejecución (por ejemplo, cuando se le acaba su quantum de tiempo) para dársela a otro proceso de mayor prioridad que está listo.

i. **i. Short Term Scheduler (Planificador a Corto Plazo):** También conocido como planificador de CPU. Selecciona cuál de los procesos que están en la cola de

listos (en memoria RAM) pasará a ejecutarse en la CPU. Se ejecuta con mucha frecuencia.

ii. Long Term Scheduler (Planificador a Largo Plazo): También conocido como planificador de trabajos. Decide qué procesos del conjunto de trabajos en disco serán cargados en la memoria para competir por la CPU. Controla el grado de multiprogramación y se ejecuta con poca frecuencia.

iii. Medium Term Scheduler (Planificador a Mediano Plazo): Se encarga del *swapping*. Puede sacar temporalmente un proceso de la memoria principal y llevarlo a disco (swap out) para reducir el grado de multiprogramación y liberarla. Más tarde, puede reintroducirlo (swap in) para que continúe su ejecución.

j. **Dispatcher:** Es el módulo que cede el control de la CPU al proceso seleccionado por el *Short Term Scheduler*. Sus tareas son:

1. Realizar el cambio de contexto (guardar el estado del proceso saliente y cargar el del entrante).
2. Cambiar al modo usuario.
3. Saltar a la ubicación adecuada en el programa del nuevo proceso para reiniciar su ejecución.

Loader: Es un programa del sistema encargado de cargar un programa ejecutable desde el almacenamiento secundario (disco) a la memoria principal (RAM) para que pueda ser ejecutado por la CPU.

k. **Proceso CPU Bound:** Es un proceso que pasa la mayor parte de su tiempo realizando cálculos y usando la CPU. Sus ráfagas de uso de CPU son largas y rara vez realiza operaciones de E/S.

Proceso I/O Bound: Es un proceso que pasa la mayor parte de su tiempo esperando que se completen operaciones de entrada/salida. Sus ráfagas de CPU son cortas, seguidas de largas esperas por E/S.

l. Un proceso atraviesa distintos estados a lo largo de su ciclo de vida:

- **Nuevo (New):** El proceso está siendo creado.
- **Listo (Ready):** El proceso está en memoria principal, esperando que se le asigne la CPU para ejecutarse.
- **En Ejecución (Running):** El proceso tiene el control de la CPU y está ejecutando sus instrucciones.
- **En Espera (Waiting/Blocked):** El proceso está esperando que ocurra un evento, como la finalización de una operación de E/S.
- **Terminado (Terminated):** El proceso ha finalizado su ejecución.

Diagrama de Transiciones:

1. **Nuevo → Listo:** El *Long Term Scheduler* admite un nuevo proceso en el sistema.
2. **Listo → En Ejecución:** El *Short Term Scheduler* selecciona el proceso para que use la CPU.

3. **En Ejecución → Listo:** El proceso es interrumpido (ej. se acaba su quantum de tiempo) por el planificador apropiativo.
4. **En Ejecución → En Espera:** El proceso solicita una operación de E/S y debe esperar a que termine.
5. **En Espera → Listo:** El evento por el que esperaba (ej. fin de E/S) ha ocurrido.
6. **En Ejecución → Terminado:** El proceso completó su tarea.

m. **Long Term Scheduler:** Controla la transición de **Nuevo → Listo**.
Short Term Scheduler: Controla la transición de **Listo → En Ejecución**.
Medium Term Scheduler: Controla las transiciones hacia y desde un estado "Suspendido" (no mostrado en el diagrama simple), sacando procesos de los estados **Listo** o **En Espera** hacia el disco.

n. **Tiempo de Retorno (TR):** Es el tiempo total que un proceso pasa en el sistema, desde que llega hasta que finaliza por completo. Se calcula como:

$$TR = Tiempo\ de\ Finalización - Tiempo\ de\ Llegada$$

Tiempo de Espera (TE): Es la suma de todos los períodos de tiempo que un proceso pasa en la cola de listos, esperando por la CPU. Se puede calcular como:

$$TE = TR - Tiempo\ de\ Ráfaga\ (CPU)$$

o. Para un lote de n procesos, se calculan como el promedio de los tiempos individuales:

- **Tiempo Promedio de Retorno (TPR):**

$$TPR = \frac{1}{n}\sum_{i=1}^{n} TR_i$$

- **Tiempo Promedio de Espera (TPE):**

$$TPE = \frac{1}{n}\sum_{i=1}^{n} TE_i$$

p. El **Tiempo de Respuesta** es el tiempo que transcurre desde que se envía una solicitud hasta que el sistema produce la **primera respuesta**, no hasta que se completa la tarea. Es una métrica crucial en sistemas interactivos.

2.

a.

i. **FCFS**: es también conocida como primero-entra-primero-sale (FIFO) o como un sistema de colas estricto. FCFS funciona mucho mejor para procesos largos que para procesos cortos.
Considérese el siguiente ejemplo, basado en [FINK88]:

Proceso	Tiempo de Llegada	Tiempo de Servicio (T_s)	Tiempo de Comienzo	Tiempo de Finalización	Tiempo de Estancia (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1,99
Media					100	26

El tiempo de estancia normalizado para el proceso Y es excesivamente grande en comparación con los otros procesos: el tiempo total que está en el sistema es 100 veces el tiempo requerido para su proceso. Esto sucederá siempre que llegue un proceso corto a continuación de un proceso largo. Por otra parte, incluso en este ejemplo extremo, los procesos largos no van mal. El proceso Z tiene un tiempo de estancia de casi el doble que Y, pero su tiempo de residencia normalizado está por debajo de 2,0.

ii. Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero (**SJF**, Shortest Job First).
En la figura encontramos cuatro trabajos (A, B, C y D) con tiempos de ejecución de 8, 4, 4 y 4 minutos, respectivamente. Al ejecutarlos en ese orden, el tiempo de respuesta para A es de 8 minutos, para B es de 12 minutos, para C es de 16 minutos y para D es de 20 minutos, para un promedio de 14 minutos:

Figura 2-40. Un ejemplo de planificación tipo el trabajo más corto primero. (a) Ejecución de cuatro trabajos en el orden original. (b) Ejecución de cuatro trabajos en el orden del tipo "el trabajo más corto primero".

iii. **Round-Robin:** a cada proceso se le asigna un intervalo de tiempo, conocido como quántum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del cuanto, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quántum, la conmutación de la CPU se realiza cuando el proceso se bloquea, desde luego.

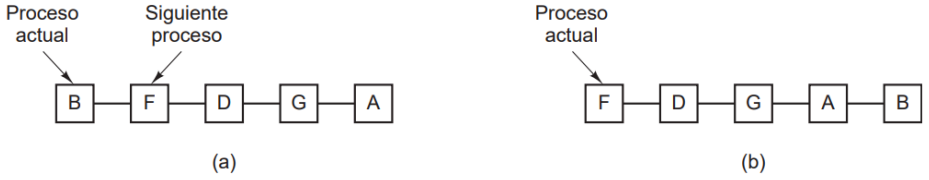


Figura 2-41. Planificación de turno circular. (a) La lista de procesos ejecutables. (b) La lista de procesos ejecutables una vez que *B* utiliza su quántum.

iv. **Prioridades:** asigna la CPU al proceso con mayor prioridad. Los procesos con mayor prioridad se ejecutan antes. Es un algoritmo preemptivo o no preemptivo.

Proceso	Tiempo de llegada	Tiempo de ráfaga	Prioridad
P1	0	6 ms	2
P2	1	8 ms	4
P3	2	7 ms	1
P4	3	3 ms	3

Ejecución:

1. P1 llega en el momento 0 y empieza a ejecutarse. Se ejecuta hasta el final, en el tiempo 6.
2. En el tiempo 6, los procesos P2, P3 y P4 están en la cola. El algoritmo selecciona el de mayor prioridad (número más bajo). P3 (prioridad 1) es el elegido.
3. P3 se ejecuta de 6 a 13 (6+7).
4. Luego, el algoritmo selecciona entre los restantes (P2 y P4). P4 (prioridad 3) tiene mayor prioridad que P2 (prioridad 4).
5. P4 se ejecuta de 13 a 16 (13+3).
6. Finalmente, se ejecuta P2, de 16 a 24 (16+8).

b.

i. **FCFS** no tiene parámetros.

ii. **SJF**: su único parámetro es el tiempo de ráfaga del CPU. Este es la estimación de cuánto tiempo de CPU necesitará el proceso para completar su ejecución.

iii. **Round-robin**: su parámetro es el Cuanto de tiempo (o time quantum). Este parámetro determina la cantidad de tiempo que cada proceso puede ejecutar en la CPU antes de ser interrumpido y enviado al final de la cola de listos.

iv. **Prioridades**: Nivel de prioridad. Es un valor (generalmente un número entero) que se asigna a cada proceso. El planificador de CPU siempre ejecutará el proceso que tenga la prioridad más alta. La forma en que se asignan y gestionan estas prioridades puede variar:

- Prioridad estática: Se asigna un valor fijo que no cambia durante la vida útil del proceso.
- Prioridad dinámica: El valor puede cambiar con el tiempo en función de diferentes factores, como el tiempo de espera o el uso de la CPU.

c.

i. **FCFS**: El más simple. Ejecuta los procesos en el orden en que llegan. Adecuado para **procesos por lotes** donde el orden es importante, pero malo para **sistemas interactivos** por el "efecto convoy".

ii. **SJF**: Prioriza los procesos con el tiempo de ejecución más corto. Óptimo para minimizar el tiempo de espera, ideal para **procesos por lotes**, pero puede causar inanición a los procesos largos.

iii. **Round Robin**: Asigna un pequeño "cuanto de tiempo" a cada proceso de forma cíclica. Es el mejor para **sistemas interactivos y de tiempo compartido**, ya que proporciona un tiempo de respuesta rápido y equitativo.

iv. **Prioridades**: Ejecuta el proceso con la prioridad más alta. Bueno para **sistemas de tiempo real y por lotes** donde la importancia de la tarea es clave. Puede llevar a la inanición de procesos de baja prioridad si no se gestiona bien.

d.

i. **FCFS**:

• Ventajas (Implícitas en su comportamiento):

◦ Se podría inferir que los procesos CPU Bound terminan al comenzar su primer ráfaga.

• Desventajas (Implícitas en su comportamiento):

◦ Los procesos I/O Bound requieren múltiples ráfagas.

ii. **SJF**:

• Ventajas:

◦ Favorece la ejecución de procesos cortos (al colocarlos delante de procesos largos).

• Desventajas:

◦ Los procesos largos pueden sufrir inanición (starvation).

iii. **Round Robin**:

• Desventajas (Relacionadas con el Quantum):

◦ Si el Quantum es pequeño, provoca un overhead de context switch.

iv. **Prioridades**:

• Desventajas:

◦ Los procesos de baja prioridad pueden sufrir inanición (starvation).

3. **a y b.** Diagramas de Gantt hechos
   c. El algoritmo con menor TPR y TPE terminó siendo el SJF (21,4 y 12) ya que encola los procesos por tiempo de ejecución. Le sigue FCFS (27,6 y 18,2) con su simpleza por encolar los procesos por orden de llegada. Y por último, Round-Robin con timer variable (32,6 y 23,2) acabó siendo el más duradero porque encola a los procesos de manera equitativa según el Quantum ingresado (overhead de context switch = coste de detener un proceso y comenzar la ejecución de otro).
4. **a y b.** Diagramas de Gantt hechos
   d. Al aplicar Round-Robin con Q=1 se pudo apreciar que los procesos, a medida que avanzaba su ráfaga de CPU, se iban encolando inmediatamente, haciendo todo el proceso de manera equitativa pero llegando a un TPR de 11,8 y TRE de 7,6. En cambio, el algoritmo con Q=6 fue lo suficientemente grande para que todos los procesos

terminaran sus ráfagas en un solo turno con un TPR de 9,8 y TRE de 5,6, haciendo que el resultado final del diagrama sea idéntico a como si se hubiera usado el algoritmo FCFS.

e. Creo que lo ideal sería elegir un quantum que sea ligeramente mayor que la duración de la ráfaga de CPU promedio en el sistema. De esta manera, la mayoría de los procesos terminarán en un solo turno (minimizando el overhead), pero el quantum seguirá siendo lo suficientemente corto como para interrumpir a los procesos muy largos y no perjudicar la interactividad.

5. **a.** Diagrama de Gantt hecho

   b. Tiene TPR de 7,8 y TPE de 3,6, siendo hasta ahora el algoritmo de planificación más rápido.
6. **a y b.** Diagrama de Gantt hecho
7.
   - **a.** La inanición es una condición o situación en la cual un proceso, aunque se encuentra listo para avanzar, es soslayado o pospuesto indefinidamente por el planificador.

     Se refiere a una condición en la cual un proceso se encuentra listo para ejecutar pero se le deniega el acceso al procesador de forma continuada en deferencia a otros procesos.
   - **b.**
     i. Algoritmo Primero el Proceso más Corto (**SJF** - Shortest Job First):
        - Esta política non preemptive (no expulsiva) selecciona el proceso con la ráfaga de CPU más corta.
        - Con SJF, los procesos largos pueden sufrir inanición (starvation). Si continuamente llegan trabajos o procesos cortos, los trabajos largos nunca podrían llegar a ejecutarse.

     ii. Algoritmo Menor Tiempo Restante (**SRTF** - Shortest Remaining Time First):
        - SRTF es la versión preemptive (apropiativa) de SJF.
        - Al igual que con SJF/SPN, existe el riesgo de inanición para los procesos más largos con SRT.

     iii. Algoritmo con Uso de **Prioridades**:
        - En este esquema, se selecciona el proceso de mayor prioridad de los que se encuentran en la Ready Queue.
        - Los procesos de baja prioridad pueden sufrir inanición (starvation). Esto sucede si existe continuamente un conjunto de procesos de mayor prioridad listos para ejecutarse y las prioridades no se ajustan.

     iv. Esquemas de **Colas Multinivel**:
        - En los esquemas de colas multinivel, la ready queue se divide en varias colas, cada una con su propio algoritmo de planificación.
        - La inanición puede ocurrir en estos esquemas. Específicamente, en un ejemplo de esquema multinivel (con colas RR y FCFS), puede ocurrir inanición con los procesos ligados a E/S si siempre llegan procesos ligados a CPU.

c.

i. Envejecimiento (Aging) o Penalización (Penalty):
◦ Para los algoritmos basados en prioridades, la solución es permitir a un proceso cambiar su prioridad durante su ciclo de vida.
◦ Esta técnica, conocida como Aging (Envejecimiento) o Penalty (Penalización), promueve la prioridad de los procesos que han estado esperando un servicio por un tiempo considerable. El envejecimiento se basa en el principio de que cualquier proceso dado se convertirá, con el tiempo, en el más antiguo y obtendrá el recurso que necesita.

ii. Uso de Algoritmos FCFS/FIFO:
◦ La inanición se puede evitar mediante el uso de una política de asignación de recursos del tipo “primero en llegar, primero en ser atendido” (FCFS).
◦ Dado que este método atiende primero al proceso que espere más tiempo, garantiza que, a su debido tiempo, cualquier proceso obtendrá el recurso necesario.
◦ Los semáforos fuertes (que encolan los procesos en orden FIFO) garantizan estar libres de inanición, a diferencia de los semáforos débiles.

iii. Promoción en Esquemas Multinivel:
◦ En esquemas de planificación retroalimentada (que son susceptibles a la inanición para procesos largos), una solución es promover a los procesos a colas de mayor prioridad después de que pasen un determinado tiempo esperando servicio en su cola actual.

8.

a. Diagrama de Gantt hecho
b. Diagrama de Gantt hecho

9.

a. Round Robin es un algoritmo apropiativo basado en tiempo (*time-sharing*), que otorga a cada proceso un pequeño intervalo de tiempo llamado **quantum (Q)**. Su principal desventaja al manejar procesos mixtos radica en la gestión ineficiente del Q, penalizando a los procesos **ligados a I/O**.
- **Desventajas de RR con Procesos Mixtos:**
1. **Penalización a Procesos I/O-Bound (Falta de Favoritismo):**
   - Los procesos I/O-bound tienen ráfagas de CPU muy cortas (menores al Q) porque rápidamente necesitan realizar una operación de E/S.
   - Bajo RR, cada vez que un proceso I/O-bound usa el CPU, pasa por el proceso de **cambio de contexto** (salvar y cargar el estado del proceso). Dado que sus ráfagas son cortas, la proporción de tiempo dedicada al cambio de contexto con respecto al tiempo útil de CPU es **alta**.

- **Resultado:** Se **desperdicia tiempo** del CPU en sobrecarga administrativa (*overhead*) en lugar de trabajo productivo, lo que ralentiza el progreso de todos los procesos.

2. **Ineficiencia del Quantum Grande:**
   - Si se elige un Q grande para reducir el *overhead* (beneficiando a los procesos CPU-bound), los procesos I/O-bound terminan esperando más tiempo del necesario. Si un proceso I/O-bound vuelve de E/S y tiene una ráfaga muy corta, debe esperar su turno completo en la cola, a menudo detrás de procesos CPU-bound que consumen el Q completo.
   - **Resultado:** Aumenta el **tiempo de respuesta** y el **tiempo de espera** para los procesos I/O-bound.
3. **Ineficiencia del Quantum Pequeño:**
   - Si se elige un Q muy pequeño (beneficiando el tiempo de respuesta), el *overhead* por cambio de contexto se vuelve excesivo para los procesos CPU-bound.
   - **Resultado:** Reduce el **rendimiento** (*throughput*) general del sistema.

b. SRTF es un algoritmo apropiativo basado en la duración de la ráfaga de CPU. Prioriza el proceso con el tiempo de CPU restante más corto. Su principal desventaja es la **inanición** potencial de los procesos CPU-bound y la dificultad de la estimación.

Desventajas de SRTF con Procesos Mixtos:

1. **Inanición (Starvation) de Procesos CPU-Bound:**
   - Los procesos I/O-bound, por su naturaleza, tienen ráfagas de CPU muy cortas (solo lo suficiente para preparar la siguiente E/S).
   - Si un sistema tiene un flujo constante de procesos I/O-bound (o si estos regresan frecuentemente de E/S), siempre tendrán el tiempo restante más corto.
   - **Resultado:** Los procesos CPU-bound, que tienen ráfagas largas, pueden ser **apropiados repetidamente** o **nunca llegar a ejecutarse** si hay un flujo continuo de procesos I/O-bound más cortos. Esto se conoce como **inanición**.
2. **Dificultad de Predicción:**
   - SRTF requiere conocer o **predecir** con precisión la duración de la próxima ráfaga de CPU.
   - La precisión de esta predicción afecta directamente la eficiencia del algoritmo. Las ráfagas de los procesos I/O-bound (que son el factor determinante en SRTF) son a menudo impredecibles.
   - **Resultado:** Si la predicción es incorrecta, el algoritmo puede tomar decisiones subóptimas, causando más apropiaciones innecesarias o una mala planificación.
3. **Mayor Overhead por Apropiación:**

- Como algoritmo apropiativo, SRTF es muy sensible a la llegada de cualquier proceso nuevo (incluidos los I/O-bound que vuelven de E/S) que tenga un tiempo restante más corto.
- Cada apropiación conlleva un cambio de contexto. Si los procesos I/O-bound regresan constantemente, el sistema incurrirá en una **alta sobrecarga de cambio de contexto**, similar al problema de un Q muy pequeño en RR, aunque por una razón diferente (prioridad por duración vs. fin del tiempo).
- **Resultado:** El tiempo útil de CPU se reduce debido a la sobrecarga administrativa.

10. Diagrama de Gantt hecho

11. Sí, es posible que el quantum de un proceso nunca llegue a 0 (cero) bajo este esquema de planificación con el algoritmo VRR (Virtual Round Robin).
La justificación se basa en las condiciones por las cuales un proceso que está siendo ejecutado es retirado de la CPU, las cuales no se limitan únicamente a la expiración de su quantum.
El algoritmo Round Robin (RR), del cual VRR es una variante, funciona asignando a cada proceso un intervalo de tiempo fijo, conocido como quantum o rodaja de tiempo (Q). En el esquema descrito, este quantum se representa mediante un contador que se decrementa en 1 con cada interrupción de reloj.
El contador de un proceso no llegará a cero si el proceso abandona la CPU por cualquiera de las siguientes razones antes de que se agote su tiempo asignado:

1. Finalización Voluntaria: El proceso termina su ráfaga de CPU (tiempo de servicio esperado).
   - Si el tiempo de ejecución requerido por el proceso es menor que el Q, el proceso finaliza y es retirado de la CPU antes de que el contador llegue a cero, y no se vuelve a encolar.
2. Bloqueo por Espera: El proceso se bloquea debido a que necesita esperar una operación de E/S, un semáforo o algún otro evento.
   - Cuando un proceso realiza una llamada al sistema que lo obliga a esperar, pasa al estado "bloqueado" (o "esperando") y es retirado de la CPU, independientemente del valor restante en su contador.

En estos casos, el proceso cede el control del procesador de forma voluntaria antes de que el contador se agote, lo que significa que el valor del quantum no llega a cero, y por lo tanto, la expulsión no se debe a la expiración de la rodaja de tiempo.
Esta situación es particularmente común en el contexto de VRR, ya que este algoritmo está diseñado para favorecer a los procesos limitados por E/S (I/O-bound). Los procesos limitados por E/S se caracterizan por tener ráfagas de CPU más cortas. Por lo tanto, es probable que estos procesos utilicen el procesador por un período breve y luego se bloqueen para esperar E/S, regresando a la cola antes de consumir su quantum completo.
Si el contador estuviera asociado a una variante de Round Robin de Timer Fijo, el contador se inicializaría en Q solo si su valor anterior era cero. Sin embargo, si se utiliza la variante más común, el Timer Variable (la más utilizada), el contador se

inicializa al valor de Q cada vez que el proceso es asignado a la CPU. En ambos casos, si el proceso se bloquea o termina antes, el contador no habrá alcanzado cero. Si se utiliza el Timer Variable, incluso si el proceso no agota su quantum, se reinicializará al valor Q en la próxima asignación, asegurando que el proceso nunca necesite alcanzar 0 si finaliza o se bloquea prematuramente.

12. .

13.

El algoritmo de Colas Multinivel (Multilevel Queue) surge de la combinación de algoritmos de planificación para aumentar la eficiencia. En este esquema, la cola de procesos listos (ready queue) se divide en varias colas, lo que se asemeja a la planificación con prioridades. Cada cola tiene su propio algoritmo de planificación, conocido como planificador horizontal, y existe un algoritmo adicional para planificar entre las colas, denominado planificador vertical.
A continuación, se proponen los algoritmos más adecuados para planificar los procesos interactivos y los procesos por lotes (Batch), basándose en sus características operativas:

a. Algoritmo de planificación para cada cola (Planificador Horizontal)
La elección del algoritmo de planificación para cada cola debe basarse en la naturaleza y los objetivos de los procesos que contienen:
1. Cola de Procesos Interactivos
Los procesos interactivos son aquellos que requieren una respuesta rápida y generalmente son limitados por E/S (I/O-bound), ya que pasan gran parte de su tiempo esperando la entrada/salida del usuario.
Algoritmo sugerido: Round Robin (RR) o Turno Rotatorio.
• Justificación: RR es uno de los algoritmos más antiguos, simples y equitativos, muy utilizado en sistemas interactivos.
• A cada proceso se le asigna un intervalo de tiempo fijo (quantum) para ejecutarse.
• Si el proceso utiliza todo su quantum, es expulsado y colocado al final de la cola. Sin embargo, los procesos limitados por E/S rara vez consumen su quantum completo y se bloquean rápidamente, lo que permite que la CPU sea asignada rápidamente a otro proceso, mejorando el tiempo de respuesta percibido.
• Si el quantum es apropiadamente seleccionado (entre 20 y 50 ms es a menudo razonable), se logra un buen equilibrio entre la sobrecarga del cambio de contexto y la respuesta a peticiones cortas.
(Nota: Alternativamente, se podría usar una planificación basada en prioridades dinámicas que favorezca a los procesos I/O-bound, como establecer la prioridad en función inversa a la fracción del quantum utilizado, lo que permite que los procesos con ráfagas cortas tengan mayor prioridad).
2. Cola de Procesos Batch (Lotes)
Los procesos Batch son típicamente limitados por la CPU (CPU-bound), realizan trabajos de rutina sin interacción urgente del usuario, y el objetivo principal de su planificación suele ser maximizar el rendimiento (throughput) y la utilización de la CPU.
Algoritmos sugeridos: First-Come, First-Served (FCFS/FIFO) o Shortest Job First (SJF).
• FCFS (Primero en llegar, primero en ser atendido):

◦ Justificación: FCFS es el algoritmo más simple y es aceptable en sistemas de procesamiento por lotes, especialmente si se utiliza una planificación no apropiativa o con periodos largos, ya que minimiza la sobrecarga por conmutación de procesos, mejorando el rendimiento. Los procesos se asignan en el orden en que lo solicitan y se ejecutan hasta que se bloquean o terminan.

• SJF (El trabajo más corto primero):

◦ Justificación: Si se conocen los tiempos de ejecución de antemano (lo cual es un supuesto común en muchos entornos Batch), SJF produce el tiempo de respuesta promedio mínimo. Esta es una política no apropiativa que selecciona el proceso con la ráfaga más corta.

b. Algoritmo para administrar las colas (Planificador Vertical)

El planificador vertical debe determinar qué cola (Interactiva o Batch) se selecciona para elegir el siguiente proceso.

Algoritmo sugerido: Planificación por Prioridad (Preemptiva)

• Estrategia: Se debe asignar una prioridad mayor a la cola de Procesos Interactivos que a la cola de Procesos Batch.

• Prioridad: Cola Interactiva > Cola Batch.

• Justificación:

◦ En sistemas de tiempo compartido, la meta primordial es la puntualidad y la respuesta rápida para los usuarios interactivos. Por lo tanto, se debe dar preferencia a los procesos que están esperando la interacción.

◦ La planificación por prioridad funciona seleccionando el proceso ejecutable con la prioridad más alta. Si la cola interactiva tiene procesos listos, se ejecutan antes que cualquier proceso de la cola Batch.

◦ Dado que los procesos interactivos utilizan RR (o un algoritmo similar) para sus ráfagas cortas, la planificación de colas puede ser preemptiva (expulsiva). Si un proceso Batch está ejecutando y un proceso Interactivo llega al estado Listo, el proceso Batch debe ser interrumpido (expulsado) inmediatamente para dar paso al proceso Interactivo.

• Riesgo de Inanición: Si solo se utiliza una prioridad estricta, los procesos de baja prioridad (Cola Batch) podrían sufrir inanición si hay un flujo constante de procesos de alta prioridad (Interactivos).

• Mitigación: Para evitar la inanición, el algoritmo de planificación entre colas podría permitir que un proceso cambie su prioridad durante su ciclo de vida (Aging o Penalty). Otra solución es utilizar un enfoque de "prioridad con clases" donde la cola de mayor prioridad se ejecuta por Turno Rotatorio, y solo si está vacía, se pasa a la siguiente cola, y así sucesivamente. Esto asegura que los procesos Batch reciban tiempo de CPU eventualmente, pero solo cuando la actividad interactiva se reduce.

14.

- **a.** Diagrama de Gantt hecho
- **b.** Diagrama de Gantt hecho

15. .

16. .

17.

a. Prioridad determinada estáticamente con el método del más corto primero (SJF)

El algoritmo Shortest Job First (SJF), o Primero el Proceso Más Corto (SPN, Shortest Process Next) en su versión no expulsiva, selecciona para su ejecución el proceso con el tiempo de procesamiento más corto esperado.
Esta estrategia beneficia principalmente a los trabajos cortos en general, ya que los procesos cortos se colocan delante de los procesos largos.

1. Cortos acotados por CPU y Cortos acotados por E/S:
    - Ambos tipos de trabajos se benefician. La principal característica que favorece SJF es la corta duración de la ráfaga de CPU.
    - El objetivo de SJF es minimizar el tiempo de respuesta promedio (tiempo de estancia).
    - Dado que los procesos limitados por E/S (I/O-bound) se caracterizan por tener ráfagas de CPU cortas y esperas frecuentes de E/S, la heurística del "más corto primero" favorece intrínsecamente a los procesos I/O-bound.
    - La versión expulsiva de SJF, conocida como SRTF (Shortest Remaining Time First), selecciona el proceso al cual le resta menos tiempo de ejecución en su siguiente ráfaga, y favorece explícitamente a los procesos I/O Bound.
2. Largos acotados por CPU y Largos acotados por E/S:
    - No se benefician. De hecho, los procesos largos (independientemente de si están limitados por CPU o E/S, aunque un proceso largo I/O-bound es menos común ya que I/O-bound implica ráfagas cortas) pueden sufrir inanición (starvation) si hay una llegada constante de procesos más cortos.

Conclusión para (a): SJF beneficia a los trabajos Cortos acotados por CPU y Cortos acotados por E/S, ya que prioriza las ráfagas de procesamiento más cortas para minimizar el tiempo de respuesta promedio.

b. Prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S.

Esta política de planificación dinámica está diseñada para recompensar a los procesos que se bloquean por E/S frecuentemente, lo cual es la definición de los procesos limitados por E/S (I/O-bound).

1. Cortos acotados por E/S:
    - Son los principales beneficiarios. Los procesos limitados a E/S tienen ráfagas de CPU cortas y, por ende, realizan esperas frecuentes por E/S.
    - Si la prioridad es inversamente proporcional al tiempo transcurrido desde la última E/S, un proceso que se desbloquea después de una E/S tendrá un tiempo transcurrido casi nulo y, por lo tanto, una prioridad muy alta.
    - El objetivo es que este proceso obtenga la CPU inmediatamente para emitir su siguiente petición de disco, permitiendo que las operaciones de E/S procedan en paralelo con otros cálculos.
    - Linux, por ejemplo, implementa heurísticas de interactividad para recompensar a los hilos interactivos (típicamente I/O-bound) y castigar a los hilos que acaparan la CPU. Windows también impulsa la prioridad de un hilo cuando se completa una E/S para que pueda volver a ejecutarse con rapidez y pueda iniciar más operaciones de E/S.

2. Largos acotados por CPU:
    - Son penalizados. Los procesos limitados a cálculos (CPU-bound) tienen ráfagas largas de CPU. Si un proceso CPU-bound está ejecutándose, el "tiempo transcurrido desde la última operación de E/S" aumenta constantemente, lo que hace que su prioridad disminuya dinámicamente. Este proceso está siendo "castigado" por acaparar la CPU.

Conclusión para (b): Esta estrategia beneficia a los trabajos Cortos acotados por E/S (I/O-bound), ya que se les da preferencia inmediatamente después de una E/S para optimizar la utilización de los dispositivos de entrada/salida y mejorar el tiempo de respuesta para tareas interactivas.

18.

La razón fundamental es que si el quantum (Q) en Round-Robin (RR) se incrementa sin límite (o hasta ser mayor que la ráfaga de CPU más larga), el algoritmo de planificación pierde su característica principal de apropiación basada en el tiempo y, por lo tanto, se comporta como el algoritmo Primero en Llegar, Primero en Ser Atendido (FIFO), que es un algoritmo no apropiativo.

A continuación, se explica brevemente la dinámica:

1. Definición de Round-Robin (RR)

El algoritmo de planificación por Turno Rotatorio (Round-Robin) es una política basada en un reloj, en la cual a cada proceso se le asigna un intervalo de tiempo fijo conocido como quantum (Q) para ejecutarse. Es una política de planificación apropiativa (o expulsiva).

Si un proceso sigue ejecutándose al final de su quantum, la CPU es apropiada (expulsada) para dársela a otro proceso. Cuando un proceso es expulsado de la CPU, es colocado al final de la Cola de Listos (Ready Queue), que opera como una cola FIFO circular.

2. Definición de Primero en Entrar, Primero en Ser Atendido (FIFO/FCFS)

El algoritmo Primero en Entrar, Primero en Ser Atendido (FCFS o FIFO) es la política de planificación más sencilla y es no apropiativa (nonpreemptive).

Con FCFS, la CPU se asigna a los procesos en el orden en el que la solicitan. Una vez que un proceso comienza a ejecutarse, se le permite seguir ejecutándose hasta que se bloquea (ya sea en espera de una operación de E/S) o libera la CPU de forma voluntaria (al terminar).

3. La Aproximación por un Quantum Ilimitado

La única diferencia funcional entre RR y FCFS es la regla de desalojo: RR puede desalojar por tiempo (apropiación), mientras que FCFS solo desalojar por finalización o bloqueo (no apropiación).

Si el valor del quantum (Q) se establece demasiado largo, o incluso mayor que el tiempo de ejecución de la ráfaga de CPU más larga del sistema:

• Se anula la preeminencia por tiempo: La apropiación (expulsión) forzada debida a la expiración del quantum no ocurrirá con frecuencia, o nunca si el Q es mayor que la ráfaga máxima.

• Comportamiento FCFS: El proceso se ejecutará hasta que su ráfaga de CPU termine o se bloquee por una operación de E/S (es decir, abandona la CPU voluntariamente). Este

es precisamente el comportamiento que define a un algoritmo no apropiativo como FCFS.

• Degeneración: En el caso extremo de un quantum que es mayor que el proceso más largo en ejecución, la planificación RR degenera en FCFS.

En resumen, cuando Q tiende a infinito, se elimina la restricción de tiempo que distingue a RR, y los procesos se ejecutan según el orden en que entraron a la cola de listos (FIFO circular) sin ser interrumpidos por el planificador, lo que equivale a la lógica de FCFS.

19.

a. Hola
[lista los archivos del directorio actual] (impreso por el hijo)

b. hola
[muestra los procesos corriendo actualmente] (impreso por el hijo)
[lista los archivos del directorio actual] (impreso por el padre)

c. Anda a rendir el Primer Parcial de Promo!
Estoy comenzando el Examen
[muestra los procesos corriendo actualmente] (impreso por el hijo)
¿Como te fue?

20.

a. Aparece 8 veces.

i. P1 (padre) -> fork(): *se crea 1 proceso más*
ii. P1 (padre) y P2 (hijo de P1) -> fork(): *se crean 2 procesos más*
iii. P1 (padre), P2 (hijo de P1), P3 (hijo de P1) y P4 (hijo de P2) -> fork(): *se crean 4 procesos más*

Entonces P1 (padre) + 7 procesos nuevos = 8 procesos que van a ejecutar `printf("Proceso \n");`

b. En este programa sí, es el mismo. 2^n = cant. líneas; siendo n = la cantidad de pasos del for().

21.

a. Se imprimirá como máximo:

```
Comienzo
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
```

b. `Comienzo` y las demás lo mismo.

c. .

d. .

22. La MMU es la parte de la CPU encargada de la traducción de direcciones. Aunque actualmente suele estar integrada en el chip de la CPU, lógicamente podría ser un chip separado, como lo era hace años. **Funciones principales:**

1. Traducción Dinámica de Direcciones: La función principal de la MMU es convertir instantáneamente la dirección virtual (o lógica) generada por un programa en la dirección física real en la Memoria de Acceso Aleatorio (RAM) donde reside el dato.
2. Manejo de la Paginación: Opera dividiendo la dirección virtual entrante en un número de página y un desplazamiento (offset) para luego utilizar el número de página como índice en la tabla de páginas, extrayendo el correspondiente número de marco de página físico.
3. Gestión del TLB: La MMU suele contener un dispositivo de hardware llamado Búfer de Traducción Adelantada (TLB). Cuando se presenta una dirección virtual, la MMU comprueba si el número de página virtual está presente en el TLB; si lo está, el marco de página se toma directamente del TLB, acelerando la traducción sin tener que pasar por la tabla de páginas en memoria.
4. Soporte de Protección: La MMU comprueba los bits de protección (permisos de lectura/escritura/ejecución) asociados a la página. Si el acceso viola estos bits, se genera un fallo de protección.
5. Administración del Sistema Operativo: El sistema operativo interactúa directamente con la MMU para su administración. Por ejemplo:

- La MMU debe restablecerse cuando se planifica un nuevo proceso para su ejecución.
- Las rutinas del kernel pueden requerir la administración de la MMU, incluyendo el acceso al estado de hardware privilegiado como los registros de control de la MMU.
- Ciertos controladores de Acceso Directo a Memoria (DMA) pueden utilizar la MMU para traducir direcciones virtuales a físicas durante las transferencias de datos.
- La funcionalidad ampliada de la MMU mejora el rendimiento del sistema operativo, como en el caso de Symbian OS.

23. Es el conjunto de todas las direcciones de memoria lógicas que un proceso puede referenciar. Incluye secciones para el código (texto), datos, la pila (stack) y la heap.

24.

a. **Dirección Lógica o Virtual**: es una direccion generada por la CPU desde la perspectiva de un proceso. Cada proceso tiene su propio espacio de direcciones lógicas, que es un conjunto contiguo de direcciones (por ej., de 0 a 2^32-1).
b. **Dirección física:** es la dirección real en la memoria principal.

25.

a.

Método	**Funcionamiento**	**Ventajas**	**Desventajas**

Particiones Fijas	La memoria principal se divide en **regiones con límites estáticos y fijos** determinados en el momento de la generación del sistema. El proceso se carga en una partición de tamaño **igual o superior**. Las particiones pueden ser del mismo o de distinto tamaño.	**Sencilla de implementar** y requiere **poca sobrecarga** para el sistema operativo.	**Fragmentación interna**: Se desperdicia espacio si el proceso es menor que la partición asignada. El número de particiones **limita el número máximo de procesos** activos.
Particiones Dinámicas	Las particiones se crean de **forma dinámica** y son de **longitud y número variable**. El proceso recibe **exactamente tanta memoria como requiere** y no más.	**No existe fragmentación interna**. Hay un **uso más eficiente de la memoria principal** ya que se ajusta al tamaño del proceso.	**Fragmentación externa**: La memoria se fragmenta en **muchos huecos pequeños** externos a las particiones. Requiere **compactación** (lo que consume tiempo de procesador) para evitar la fragmentación externa. Es **más complejo de mantener**.

b. El Kernel, o administrador de memoria, necesita mantener estructuras de control para gestionar las particiones. En particular, necesita:

i. Registro de Uso de Memoria: El administrador debe llevar el registro de cuáles partes de la memoria están en uso (asignadas a procesos) y cuáles están libres. Esto incluye mantener un registro de las reservas de memoria principal por parte de los procesos.

ii. Información de Localización: Necesita conocer la posición en memoria principal de cada proceso.

iii. Gestión de Huecos (Particiones Dinámicas): Debe mantener una lista de los procesos y huecos (bloques de memoria libre) ordenada por dirección, y saber cuánta memoria debe asignar.

iv. Registros de Control (Para la reubicación y protección): Para la traducción de direcciones y la protección de memoria, el Kernel

controla los registros base y límite del hardware. El Kernel debe asegurarse de que cada imagen de proceso esté aislada de las demás mediante estos registros.

- La información debe incluir los atributos de protección que restringen el uso de la memoria principal y virtual.

c.

i. **De lógicas a físicas:**

Componente	**Fórmula**	**Función y Justificación (Según las fuentes)**
Página (Virtual)	página = (dirLógica / tamañoPágina)	La dirección virtual se divide en un **número de página virtual** (bits de orden superior) y un desplazamiento (bits de orden inferior). Si el tamaño de página es una potencia de 2, la división entera aísla el número de página.
Desplazamiento	desplazamiento = (dirLógica MOD tamañoPágina)	El desplazamiento (*offset*) especifica la ubicación del *byte* dentro de la página y se obtiene del resto de la división. Este valor **se mantiene sin cambios** durante la traducción.
Marco (Físico)	marco = marcoAsignadoAPágina	El número de página virtual se utiliza como **índice en la tabla de páginas** para encontrar el número de marco de página físico asociado (si la página está en memoria).
Dirección Física	dirFísica = (marco * tamañoPágina) + desplazamiento	La **Dirección Física** se construye al combinar el número de marco con el desplazamiento. Esto es matemáticamente equivalente a concatenar el número de marco y el desplazamiento.

ii. **De físicas a lógicas:**

Componente	**Fórmula**	**Función y Justificación (Relación inversa)**
Marco y Desplazamiento	marco = (dirFísica / tamañoPágina) y desplazamiento = (dirFísica MOD tamañoPágina)	Descomponen la dirección física en el número de marco y el desplazamiento dentro de ese marco, lo cual es la composición de la dirección física.
Página (Virtual)	página = páginaAsignadaAMarco	Esta es la operación inversa central. El SO debe determinar a qué página virtual y a qué proceso pertenece un marco físico dado. Esta necesidad es la razón de ser de estructuras de datos como la Tabla de Páginas Invertida (Inverted Page Table), que se indexa por el número de marco de página real y contiene la información del número de página virtual y el identificador del proceso que lo posee.
Dirección Lógica	dirLógica = (página * tamañoPágina) + desplazamiento	Reconstruye la dirección virtual del proceso al concatenar el número de página virtual recuperado y el desplazamiento.

26.

Alternativa	**Ventajas**	**Desventajas**

Particiones de Igual Tamaño	La **ubicación de los procesos es trivial**. Un proceso se carga en cualquier partición disponible, ya que no importa qué partición se utiliza. Es un esquema **sencillo de implementar**.	**Fragmentación interna** significativa: se desperdicia espacio si el proceso es más pequeño que el tamaño de la partición. Si un programa es **demasiado grande** para caber en una partición, el programador debe usar la técnica de *overlays* (superposiciones).
Particiones de Diferente Tamaño	Permite acomodar programas de **mayor tamaño** sin la necesidad de *overlays*. Puede minimizar el desperdicio de memoria (o lograr **menor fragmentación interna**) si los programas pequeños se asignan a las particiones más pequeñas que los puedan albergar. Ofrece un grado de **flexibilidad** frente a las particiones fijas.	La **fragmentación interna** sigue siendo un problema, aunque mitigado. Si se utilizan colas separadas para cada partición, el sistema podría ser **subóptimo** (una partición grande podría quedar sin usar aunque haya procesos más pequeños esperando en otras colas).

27.

a.

i. La fragmentación **interna** ocurre cuando la memoria se divide en particiones o bloques de tamaño fijo.
Referencia del problema: Este problema surge cuando se asigna un bloque de datos (o un proceso) a una partición que es mayor que el tamaño real del proceso.
- El espacio desperdiciado se encuentra dentro de la partición asignada.
- La memoria malgastada se limita generalmente a la fracción de la última partición que queda sin utilizar.
- En el esquema de Particiones Fijas, esto sucede si el proceso que se carga es más pequeño que el tamaño de la partición asignada.

La fragmentación **externa** ocurre cuando la memoria se divide en particiones de tamaño variable, como es el caso de las Particiones Dinámicas.
Referencia del problema:
- A medida que los procesos se cargan, se ejecutan y se eliminan o se intercambian a disco, el espacio libre de memoria principal

(denominado huecos) se fragmenta en muchos trozos pequeños que son externos a las particiones ocupadas.
• Este fenómeno se conoce también como el "efecto de tablero de ajedrez".
• Aunque el espacio total de los huecos podría ser suficiente para acomodar un nuevo proceso, estos huecos están dispersos, haciendo que la utilización de la memoria se reduzca.

b. Técnica: **Compactación**

La compactación es la técnica utilizada por el Kernel para eliminar la fragmentación externa.

Mecanismo de Mitigación:

1. Desplazamiento de Procesos: De forma periódica, el sistema operativo desplaza los procesos que están en la memoria principal.
2. Contigüidad: El objetivo de este desplazamiento es asegurar que los procesos queden contiguos en la memoria.
3. Unificación de Memoria Libre: Al mover los procesos para que queden juntos, toda la memoria libre (los huecos fragmentados) se une en un único bloque. Esto permite que se puedan cargar procesos adicionales que requieran un gran bloque contiguo.

Requisito esencial:

La compactación requiere la capacidad de reubicación dinámica (dynamic relocation). Esto significa que debe ser posible mover un programa desde una región de memoria principal a otra sin invalidar las referencias de memoria del programa.

28.

Técnica	**Mecanismo de Funcionamiento**	**Información que debe mantener el Kernel**
First Fit (Primer Ajuste)	El Kernel **escanea la lista de segmentos** (procesos y huecos) desde el principio de la memoria y selecciona el **primer hueco** que sea lo suficientemente grande para acomodar el proceso. El hueco se divide en dos partes: una para el proceso y otra para el resto sin utilizar (a menos que el ajuste sea exacto).	El Kernel debe mantener una **lista de huecos** (bloques libres de memoria) y procesos, generalmente **ordenada por dirección**.

Next Fit (Siguiente Ajuste)	Funciona de manera similar a First Fit, pero el Kernel mantiene un **registro de dónde se detuvo la última vez** y **comienza la búsqueda** del siguiente hueco adecuado desde ese punto, en lugar de empezar siempre desde el principio.	El Kernel debe mantener una **lista de huecos** y un **puntero o registro** de la ubicación de la última asignación exitosa.
Best Fit (Mejor Ajuste)	El Kernel **busca en toda la lista** de huecos, de principio a fin, y selecciona el **hueco más pequeño** que sea lo suficientemente grande para la solicitud. Este método busca minimizar el desperdicio de espacio (*fragmentación interna*) dentro del hueco asignado.	El Kernel debe mantener una **lista de huecos**. Esta lista a menudo se mantiene **ordenada por tamaño** para acelerar la búsqueda, lo que hace que el algoritmo sea más rápido en la asignación al encontrar inmediatamente el mejor ajuste sin buscar toda la lista.
Worst Fit (Peor Ajuste)	El Kernel **busca en toda la lista** y selecciona el **hueco más grande** disponible. El objetivo es que, al dividir el hueco más grande, el fragmento sobrante sea lo suficientemente grande como para ser útil para futuras asignaciones.	El Kernel debe mantener una **lista de huecos**.

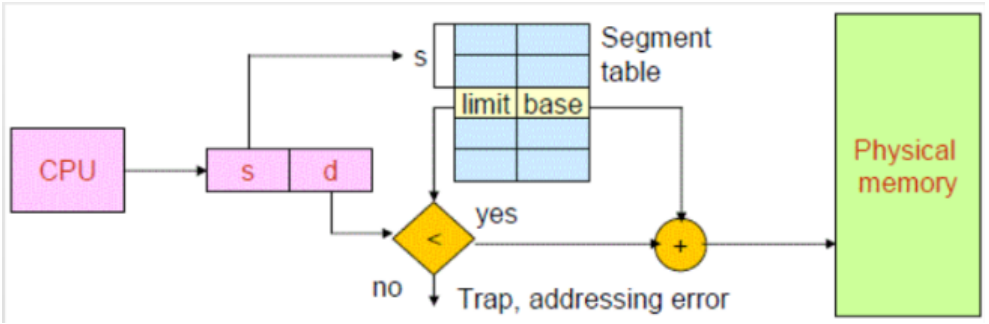
29.

a. La segmentación trabaja basándose en los siguientes principios:
• **Estructura Lógica:** La memoria es vista como una colección de **espacios de direcciones completamente independientes**, llamados **segmentos**. Estos segmentos son entidades lógicas de las que el programador está consciente.
• **Longitud Variable:** Los segmentos tienen **longitudes variables** (que pueden cambiar durante la ejecución). Un segmento puede contener un procedimiento, un *array*, una pila o una colección de variables, pero generalmente no una mezcla de distintos tipos.

• **Direccionamiento Bidimensional:** Para especificar una dirección, el programa debe proporcionar una dirección en **dos partes**: **un número de segmento** y una **dirección dentro del segmento** (desplazamiento).
• **Asignación al Programa:** En la segmentación simple, todos los segmentos de un proceso deben cargarse en la memoria principal para que el programa se ejecute. Los segmentos **no tienen que estar localizados de forma contigua** en la memoria principal.
• **Independencia:** Debido a que cada segmento es un espacio de direcciones separado, los segmentos pueden **crecer o reducirse de manera independiente** sin afectarse unos a otros.

b. El **Kernel** (Administrador de Memoria) necesita estructuras de datos para gestionar la asignación, traducción y protección de los segmentos:
1. Tabla de Segmentos: Por cada proceso, el Kernel mantiene una tabla de segmentos. Esta tabla es utilizada por el hardware de gestión de memoria para la traducción. Cada entrada de la tabla debe contener:
    ◦ La dirección inicial o base del segmento en la memoria principal.
    ◦ La longitud del segmento, utilizada para comprobar la validez de las direcciones (protección).
    ◦ Bits de control o protección (por ejemplo, permisos de lectura/escritura).
    ◦ Un bit de presente/ausente (en el caso de segmentación con memoria virtual) para indicar si el segmento está en la memoria principal.
2. Lista de Memoria Libre (Huecos): El Kernel debe mantener una lista ligada de los segmentos de memoria asignados y libres (huecos). Dado que la segmentación utiliza particiones de tamaño variable, esta lista ayuda a ubicar un hueco adecuado para los segmentos que llegan.

c. En la segmentación, la traducción dinámica de una Dirección Lógica (número de segmento + desplazamiento) a una Dirección Física se realiza sumando la dirección lógica al Registro Base del segmento, siempre y cuando el desplazamiento esté dentro del límite de longitud del segmento.



1. El procesador extrae el Número de Segmento (S) de la dirección lógica y lo utiliza para indexar la Tabla de Segmentos.
2. La Tabla de Segmentos devuelve la Dirección Base (B) y la Longitud (L) del segmento.

3. El hardware compara el Desplazamiento (D) con la Longitud (L). Si D >= L, se produce un fallo de segmento (violación de protección).
4. Si D < L, se calcula la Dirección Física sumando la Dirección Base y el Desplazamiento: Dirección Física = B + D.

d. En el esquema de segmentación pura:
• Fragmentación Interna: No se produce fragmentación interna. Esto se debe a que cada segmento recibe exactamente la cantidad de memoria que requiere.
• Fragmentación Externa: Sí se produce fragmentación externa. Al igual que en el particionamiento dinámico, la memoria se divide en trozos que contienen segmentos y huecos dispersos. Con el tiempo, esto genera muchos huecos pequeños fuera de las particiones ocupadas (el efecto de tablero de ajedrez).

e. Dado que la segmentación utiliza particiones de tamaño variable (similar al Particionamiento Dinámico) y depende de encontrar huecos libres para cargar los segmentos, las técnicas de asignación son cruciales para mitigar la fragmentación externa.
De las técnicas de asignación (Best Fit, Worst Fit, First Fit y Next Fit) vistas previamente, el algoritmo que generalmente se ajusta mejor y produce mejores resultados en la práctica es **First Fit** (Primer Ajuste):
• **Velocidad**: First Fit es la técnica más rápida ya que escanea la lista de huecos de memoria desde el principio y se detiene tan pronto como encuentra un hueco lo suficientemente grande para el segmento. Busca lo menos posible.
• **Rendimiento en Fragmentación**: Aunque intuitivamente Best Fit podría parecer mejor (al elegir el hueco más pequeño que ajusta), Best Fit tiende a dejar muchos huecos residuales pequeños e inútiles, lo que deteriora rápidamente la memoria. First Fit, al ser menos selectivo, a menudo deja fragmentos más grandes y utilizables en el resto de la memoria, lo que es beneficioso para futuras solicitudes de segmentos.

30.

a. **Dir. lógica 0000:9001**
Dir. base: 102 ; Tamaño: 12500.
Como el desplazamiento 9001 < al tamaño 12500...
Entonces se traduce a la dir. física = dir. base + desplazamiento = 102 + 9001 = **9103**.

b. **Dir. lógica 0001:24301**
Dir. base: 28699 ; Tamaño: 24300.
Como el desplazamiento 24301 > al tamaño 24300...
No se puede traducir.

c. **Dir. lógica 0002:5678**
Dir. base: 68010 ; Tamaño: 15855.
Como el desplazamiento 5678 < al tamaño 15855...
Entonces se traduce a la dir. física = 68010 + 5678 = **73688**.

d. **Dir. lógica 0001:18976**
Dir. base: 28699 ; Tamaño: 24300.
Como el desplazamiento 18976 < al tamaño 24300...

Entonces se traduce a la dir. física = 28699 + 18976 = **47675**.

e. Dir. lógica 0003:0

Dir. base: 80001 ; Tamaño: 400.

Como el desplazamiento 0 < al tamaño 400...

Entonces se traduce a la dir. física = 80001 + 0 = **80001.**

31.

a. La paginación es una solución elegante y potente al problema de la fragmentación externa. Su concepto central consiste en romper la restricción de la contigüidad, permitiendo que el espacio de direcciones de un proceso se disperse por la memoria física. Para lograrlo, tanto el espacio de direcciones lógicas como la memoria física se dividen en bloques de tamaño fijo: el espacio lógico se divide en páginas y la memoria física en marcos de página.

b. Para implementar la paginación, la MMU utiliza una estructura de datos llamada tabla de páginas. Cada proceso tiene su propia tabla de páginas, que mapea sus páginas virtuales a los marcos de página físicos.

c.

i. La CPU genera una dirección lógica, que la MMU divide en dos componentes:
   - Número de Página (p): Los bits de orden superior de la dirección.
   - Desplazamiento (d): Los bits de orden inferior de la dirección.

ii. El número de página (p) se utiliza como índice para acceder a la tabla de páginas del proceso.

iii. La entrada correspondiente en la tabla de páginas contiene el número de marco (f) donde reside esa página en la memoria física.

iv. La dirección física final se construye combinando el número de marco (f) con el desplazamiento (d).

d. La fragmentación interna es inherente al diseño de la paginación:

i. Causa: La paginación divide tanto el espacio de direcciones lógicas como la memoria física en bloques de tamaño fijo (páginas y marcos). Es muy raro que el tamaño de un proceso sea un múltiplo exacto del tamaño de la página.

ii. Mecanismo: La última página de un proceso, en general, no se llena completamente con código o datos. El espacio restante en esa última página, aunque asignado al proceso, permanece sin usar.

32. Tamaño de página: 512 bytes;
C/ dir = 1 byte;
RAM: 10240 bytes (10 KiB);
P1: 2000 bytes.

Página	Marco
0	3
1	5
2	2
3	6

a.

Programa
Página 0
Página 1
Página 2
Página 3

Tabla de páginas	
Página	**Marco**
0	3
1	5
2	2
3	6

Memoria	
Marco	**Página**
0	-
1	-
2	2
3	0
4	-
5	1
6	3
..	-
19	-

- **Número de páginas para P1:** 2000 / 512 = 3.9 (4 páginas).
- **Número de marcos:** 10240 / 512 = 20 (0 - 19).
- **Rango de dir. lógicas:**
  - **Página 0:** 0 - 511
  - **Página 1:** 512 - 1023
  - **Página 2:** 1024 - 1535
  - **Página 3:** 1536 - 1999 (porque P1 ocupa 2000 bytes)

b. - **Desplazamiento**: dirLógica MOD tamañoPágina.
- **Dir. física**: (marco * tamañoPágina) + desplazamiento.

i. Corresponde a la página 0.
- Desplazamiento: 35 MOD 512 = 35.
– Marco: 3
- Dir. física: (3 * 512) + 35 = **1571**

ii. Corresponde a la página 1.
- Desplazamiento: 512 MOD 512 = 0
- Marco: 5
- Dir. física: (5 * 512) + 0 = **2560**

iii. No corresponde.

iv. Corresponde a la página 0.
- Desplazamiento: 0 MOD 512 = 0
- Marco: 3
- Dir. física: (3 * 512) + 0 = **1536**

v. Corresponde a la página 2.
- Desplazamiento: 1325 MOD 512 = 301
- Marco: 2
- Dir. física: (2 * 512) + 301 = **1325**

vi. Corresponde a la página 1.
- Desplazamiento: 602 MOD 512 = 90
- Marco: 5
- Dir. física: (5 * 512) + 90 = **2650**

c. **Marco**: (dirFísica / tamañoPágina)
Desplazamiento: (dirFísica MOD tamañoPágina)
Dir. lógica: (página * tamañoPágina) + desplazamiento

i. Marco: 509 / 512 = 0 -> **el marco 0 no está asociado a ninguna página de P1.**

ii. Marco: 1500 / 512 = 2 (página 2)
Desplazamiento: 1500 MOD 512 = 476
Dir. lógica: (2 * 512) + 476 = **1500**

iii. Marco: 0 / 512 = 0 -> **el marco 0 no está asociado a ninguna página de P1.**

iv. Marco: 3215 / 512 = 6 (página 3)
Desplazamiento: 3215 MOD 512 = 143
Dir. lógica: (3 * 512) + 143 = **1679**

v. Marco: 1024 / 512 = 2 (página 2)
Desplazamiento: 1024 MOD 512 = 0
Dir. lógica: (2 * 512) + 0 = **1024**

vi. Marco: 2000 / 512 = 3 (página 0)
Desplazamiento: 2000 MOD 512 = 464
Dir. lógica: (0 * 512) + 464 = **464**

Nota: *aunque el marco corresponda a una página, también se debería chequear que la dir. lógica obtenida sea menor al tamaño del proceso (<2000).*

33. Tamaño de página: 2048 bytes;
C/ dir = 1 byte;
Rango de dir. lógicas:

a. **Página 0:** 0 - 2047
b. **Página 1:** 2048 - 4095
c. **Página 2:** 4096 - 6143
d. **Página 3:** 6144 - 8191
e. **Página 4:** 8192 - 10239

Página	Marco
0	16
1	13
2	9
3	2
4	0

i. Página: 5120 / 2048 = 2
Desplazamiento: 5120 MOD 2048 = 1024
Dir. física: (9 * 2048) + 1024 = 19456

ii. Página: 3242 / 2048 = 1
Desplazamiento: 3242 MOD 2048 = 1194
Dir. física: (13 * 2048) + 1194 = 27818

iii. Página: 1578 / 2048 = 0
Desplazamiento: 1578 MOD 2048 = 1578
Dir. física: (16 * 2048) + 1578 = 34346

iv. Página: 1
Desplazamiento: 0
Dir. física: (13 * 2048) + 0 = 26624.

v. Página: 8191 / 2048 = 3
Desplazamiento: 8191 MOD 2048 = 2047
Dir. física: (2 * 2048) + 2047 = 6143

34. .

35.

a. En el esquema de paginación **simple**, cada proceso tiene su propia tabla de páginas. La dirección virtual (o lógica) se divide en dos partes: un número de página virtual (bits de orden superior) y un desplazamiento (bits de orden inferior).
El número de página virtual se utiliza como índice en la tabla de páginas para buscar la entrada correspondiente. Esta entrada proporciona el número de marco de página físico asociado (si la página está presente en memoria). Finalmente, el número de marco se adjunta al extremo de orden superior del desplazamiento para formar la dirección física que se envía a la memoria.

b. En un esquema de **dos niveles**, la dirección virtual se particiona, por ejemplo, en un campo TP1 (Tabla de Páginas de Nivel Superior), un campo TP2 (Tabla de Páginas de Segundo Nivel) y el Desplazamiento.

1. El campo TP1 se utiliza como índice en la tabla de páginas de nivel superior (o directorio de páginas).
2. La entrada resultante proporciona la dirección (o número de marco de página) de una tabla de páginas de segundo nivel.
3. El campo TP2 se utiliza como índice en la tabla de páginas de segundo nivel seleccionada para obtener el número de marco de página físico.
4. Este número de marco se combina con el Desplazamiento para obtener la dirección física.

Linux, por ejemplo, utiliza un esquema de cuatro niveles para la paginación para mejorar la eficiencia en arquitecturas de 32 y 64 bits

c. A diferencia de los enfoques tradicionales, la tabla de páginas **invertida** (IPT) tiene una entrada por cada marco de página en la memoria real, en lugar de una entrada por cada página del espacio de direcciones virtuales. Cada entrada lleva un registro de qué proceso y qué página virtual se encuentra actualmente en ese marco físico.
Para traducir la dirección virtual (proceso n, página virtual p):

1. El hardware no puede indexar la tabla directamente.
2. Se debe buscar en toda la tabla de páginas invertida la entrada que contenga el par (proceso n, página virtual p).
3. Si la entrada se encuentra, la posición de esa entrada en la tabla (su índice) es el número de marco de página.

36. **C/ dir.** = 1 byte
    Tamaño de dir.: 32 bits
    Tamaño de pág.: 2048 bytes
    Proceso P1: 51358 bytes de código y 68131 bytes de datos.
    a. Tamaño de página: 2048 bytes;
       Bits para desplazamiento: 2^d = 2048 -> d = **11**
       Bits para identificar página: 32 - d = 32 - 11 = **21**
    b. 2^32 bytes = 4GiB
    c. 2^21 páginas (2097152)
    d. 2^32 / 2048 = 2097152 marcos.
    e. 51358 / 2048 = 26 páginas
    f. 68131 / 2048 = 34 páginas
    g. Total: 3391 bytes de fragmentación interna.
        i. Para código: 26 * 2048 = 53248 -> 53248 - 51358 = 1890 bytes de fragmentación interna.
        ii. Para datos: 34 * 2048 = 69632 -> 69632 - 68131 = 1501 bytes de fragmentación interna.
    h. 2^20 * 1024 = 2^30 = 1GiB. La cantidad de PTE es igual a 2^(cantidad de bits dedicados a identificar páginas).