Administración de Memoria

↳ Sistema operativo → gestiona la memoria de trabajo, en donde la memoria principal es RAM

MONOPROGRAMACIÓN

↳ El SOST ejecuta de principio a fin, un solo proceso a la vez,
↳ toda la memoria se le asigna al proceso y otra porción de mem al usuario

PERO → MULTIPROGRAMACIÓN con particiones fijas (mejor)

↳ permite hacer + uso de la CPU ya que los procesos están la mayor parte haciendo E/S
↳ dividimos la memoria y vamos encajando los procesos en el lugar + chico que entren
↳ x) celdas muy subutilizadas y celdas vacías → no aprovechamos al max la memoria disponible,
↳ x) desperdiciamos una parte grande en algo chico
↳ x) no podemos tener procesos que sean + grandes que la partición
↳ x) son poco flexibles
↳ ✓) son fáciles de implementar → ya no se usan en la práctica

Reubicación vs Protección → como garantizar que un proceso no lea y se sobreescriba en el lugar del otro

↳ como definimos las direcciones de memoria? → no podemos, ej el programa no conoce donde se va a cargar al momento de ser enlazado.

Solución: Registro base y un registro límite + direcciones relativas, cuando se quiere acceder a una dirección se le suma el registro base y se verifica que este dentro del límite
↳ no se usa → muchos ciclos de CPU

Problema de falta de memoria

→ quiero abrir mi 76ava pestaña, pero no tengo espacio para alojar un nuevo thread

↳ solución: usar la MEMORIA RAM + un poco de DISCO

↳ estrategias:

* **SWAPPING** → cuando se quiere ejecutar un proceso:

✓) si ya estaba cargado, no hago nada
✓) ya guardado en disco y hay mem disponible → lo cargo en memoria
✓) si estaba guardado en disco y no entra en la memoria disponible, copio uno o varios procesos a disco para liberar memoria y lo cargo.

✓) aceptamos procesos de los que no entran en memoria → no tiene particiones fijas (+ flexible)
x) fragmentación externa → muchos huecitos → compactar memoria (muy costoso)

* **MeM Dinamica** → asignan memoria de más para que los procesos puedan crecer en ejecución
↳ ej: malloc

NOTA

Scheduling

↳ Proceso del sistema operativo
↳ encargado de cambiar el estado de un proceso a en "ejecución" → hace uso del PROCESADOR
↳ su funcionamiento, depende de un tipo preventivo
↳ algunos sistemas brindan ⊖ de los implementación
↳ Rol clave en la organización de un sistema.

↳ no existe un scheduler ideal "todo terreno", depende fuertemente del contexto de uso de la computadora.

↳ Distintos usos/objetivos a maximizar

↳ Tiempo de ejecución: tiempo que tarda un proceso en completar su ejecución
↳ Utilización de la CPU: porcentaje de tiempo en que la CPU está en uso.
↳ Tiempo de respuesta: el tiempo de reacción percibido por un usuario en procesos interactivos
↳ Rendimiento (throughput): la cantidad de procesos que se terminan por unidad de tiempo
↳ Equanimidad (fairness): todos los procesos tienen una cuota "equitativa" de tiempo de CPU. No necesariamente tiene que ser igual para todos los procesos.
↳ Liberación de recursos: priorizar la ejecución de procesos que más recursos consumen (para que los liberen cuanto antes)

ALGORITMOS de scheduling

↳ depende del conjunto de criterios que queremos maximizar
↳ en ~~el~~ situaciones donde se ~~debe~~ activa el cambio de proceso en ejecución

① El proceso en ejecución pasa al estado "en espera" producto de x ejemplo solicitud de E/S
②③ " " " " " " " "preparado" producto de una interrupción
uno de los procesos pasa de "en espera" a "preparado" cuando finaliza E/S
④ El proceso en ejecución termina

1 y 4 → Planificador sin desalojo o cooperativo, una vez que un proceso empieza a ejecutar se mantiene en el procesador el tiempo que necesite

Si no:
Si el planificador puede desalojar un proceso en cualquier momento, se dice que es con desalojo o apropiativo

RAFAGAS de CPU y E/S → estructura de un programa → secuencia intercalada de instrucc

↳ más que usan la CPU y escrituras o lecturas en dispositivos de E/S

↳ las rafagas de CPU tardan ⊖ que E/S, los procesos están la mayor parte del tiempo esperando que finalice una operación de E/S

```
0 load VAR1
1 SHL VAR2      } CPU = 3 segs
2 lectura
3 espera de es
...
10 espera de es } E/S = 7 segs
```

Dispatcher

↳ Tarde lo mínimo Posible → tiempo "muerto" que los algoritmos tienen que tener en cuenta → *Llamarlo muy seguido: saca de P, mismo y procesos al espera.*
↳ Tiempo donde el procesador NO HACE NADA

[ALGORITMOS DE PLANIFICACIÓN]

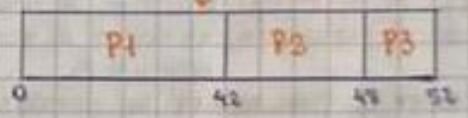
FCFS : First come First served

↳ Se ejecutan los procesos en orden de llegada → algoritmo cooperativo
↳ Se implementa utilizando la estructura de datos cola.

Ejemplo Procesos en estado Preparado

PROCESO	TIEMPO DE RAFAGA
P1	42
P2	6
P3	4

Diagrama de Gantt



Tiempo de espera Promedio = (P1: 0 + P2: 42 + P3: 48) / 3 = 30 milisegundos

✓ muy sencillo de implementar
✓ no tiene overhead
✓ no requiere de metadata
✓ Aprovecha la CPU, cuando comienza a ejecutarse el proceso → ✓ no tiene starvation

✗ no es predecible el tiempo de espera
✗ Efecto convoy: un proceso lento puede retrasar
✗ no es recomendable si interactivos

SJF : Shortest Job First

↳ Algoritmo con prioridad → Elige siempre el proceso con la siguiente ráfaga de CPU @ corta
Si hay empate FCFS → COOPERATIVO y APROPIATIVO

PROCESO	TIEMPO DE RAFAGA
P1	8
P2	24
P3	15
P4	3

Diagrama de Gantt

✓ Minimiza el tiempo de espera Promedio
✗ No puedo la CPU estimar los tiempos de ráfagas
✗ Starvation

Tiempo de ejecución = Suma de T. ráfaga

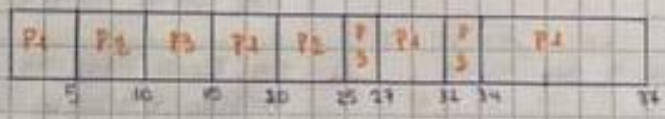
HOJA Nº

FECHA

ROUND ROBIN

↳ Similar al FCFS Pero con desalojo
↳ A cada Proceso se le asigna un quantum o cuanto de tiempo
↳ Cuando un proceso consume su quantum, se hace un context switch y se pasa al siguiente Proceso
↳ Se puede implementar mediante una cola circular
↳ Cuando un proceso es desalojado se lo envía al final de la cola
↳ quantum → corto: dedicado a planificar y hacer también de corrector
↳ Largo: genera FCFS y se vuelve inadecuado para Procesos interactivos
↳ Práctica: entre 10 y 100 ms mismos que el cambio de contexto

Proceso	Tiempo de Ráfaga
P1	12
P2	12
P3	7



Quantum: 5

ALGORITMO DE Planificación
con Prioridad

↳ Se define un esquema de prioridad entre Procesos y se ejecuta en orden de mayor a menor
↳ la prioridad se puede definir → INTERNAMENTE → SJF
↳ EXTERNA → usuario determina Prioridad
↳ Pueden ser cooperativos como apropiativos
↳ Puede producir starvation → un Proceso de baja Prioridad se queda esperando la cpu indefinidamente
↳ envejecimiento → aumentar la Prioridad del proceso con el paso del tiempo.

COLAS MULTINIVEL

↳ se dividen grupos de procesos, según sus características
↳ cada grupo tiene asociado una cola con su propio algoritmo de planificación
↳ Se deben planificar las colas y los procesos dentro de cada cola
↳ la planificación entre las colas suele ser apropiativa y con prioridades fijas
↳ esquema poco flexible con riesgo de inanición.

COLAS M. realimentadas

↳ los Procesos se pueden mover entre las distintas colas
↳ soluciona el problema de inanición de las colas multinivel → si un proceso espera mucho se lo pasa a una cola de mayor prioridad.
↳ las colas se dan cuenta del tiempo de ~~este~~ RAFAGA de cpu
↳ c. mayor Prioridad: P RAFAGAS CORTAS
↳ c. menor prioridad: P RAFAGAS LARGAS
↳ Procesos irregulares se moverán entre las colas.
↳ el quantum de tiempo asignado a cada cola inversamente proporcional a su Prioridad.
↳ si un proceso en la cola N consumió todo su quantum y no terminó, se lo pasa a otra de menor Prioridad.

¿Para cuando elegir cada algoritmo? → no existe uno ideal

Sistema con mayoritariamente uso intensivo de los dispositivos → RTA: Round Robin ct quantum corto ya que manda a los procesos a esperar rápidamente, ya que usan intensivamente E/S.

Sistema con mayoritariamente uso intensivo de CPU → RTA: el algoritmo FCFS porque cuando un proceso se empieza a ejecutar aprovecha la CPU al máximo todo el tiempo que necesite y no se desaloja hasta que termine.

↳ RTA 2: el algoritmo con Round Robin luego así puede hacer un buen uso de la CPU sin ser interrumpido rápidamente pero que de igual forma los vaya mandando a la cola para que no ocurra el efecto convoy

Distribución desconocida → RTA: Round Robin con quantum large para que los procesos se vayan a cabo, por ejemplo ejecución.

Procesos de corta duración

↳ FCFS: favorece a los procesos de corta duración ya que ya ejecutando uno tras el otro y al ser todos de corta duración no se produce convoy

↳ SJF: favorece a los procesos de corta duración ya que ejecuta uno tras otro rápidamente.

↳ SRT: no favorece ya que pierde mucho tiempo haciendo los cambios de contexto que ocupando CPU. (no se aprovecha el tiempo)

↳ Round Robin: no favorece cuando el quantum es corto ya que estaría constantemente cambiando de contexto y procesos de ejecución. Si el quantum es largo, se evita el cambio de contexto y se ejecutan los procesos en orden de llegada.

↳ Colas multinivel: favorece para no es conveniente porque tarda + tiempo Planificando ejecutándolo que ejecutándolo.

Procesos con mucha E/S

↳ FCFS: no es conveniente porque estos se quedarían esperando una E/S mientras los con la CPU, cuando podrían ser desalojados a la cola de espera para que la CPU corra otro proceso.

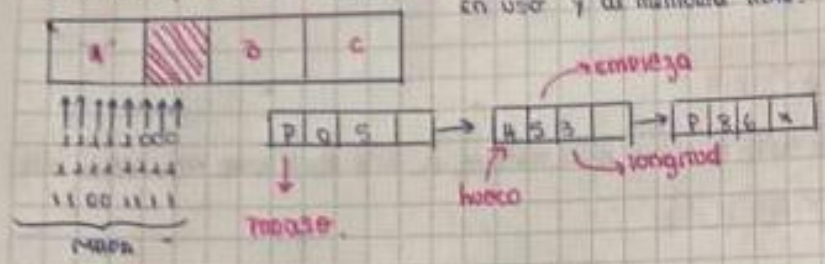
↳ SJF: no favorece a los procesos que hacen mucha entrada y salida ya que si un proceso esta esperando una E/S, no se para a ejecutar el siguiente proceso por lo que desperdicia tiempo de CPU.

↳ SRT: no favorece ya que al ejecutar los procesos de corta duración, si hacen mucha E/S van a tardar mucho en ser ejecutados

↳ Round Robin: favorece ya que como cada proceso se le asigna un tiempo de CPU, no se espera indef

↳ Colas: favorece, al disponer de quantum el algoritmo de espera

BITMAPS y LISTAS ENLAZADAS → registro de cuales son las porciones de memoria en uso y de memoria libre.



* **MEMORIA VIRTUAL** → Podría suceder que todos los procesos juntos no estén en memoria pero que no todos estén usando toda su memoria en un instante dado. ⇒ extender el espacio de direcciones virtualizadolas.

ESPACIO D. VIRTUALES = MEMORIA RAM + DISCO (Programas solo acceden a dirc virtuales)
↳ MMU traduce virtuales a físicas → con la TABLA DE PÁGINAS

PAGINACIÓN → son potencias de 2 para divide en offset e índice

↳ El espacio de direcciones virtuales se divide en unidades llamadas Páginas.
↳ El espacio de direcciones físicas se divide en unidades llamadas marcos → (igual tamaño Páginas)
↳ Reduce el tamaño de la tabla.

INDICE / CANT DE PÁGINAS = memoria virtual / tamaño de página
OFFSET = [illegible] T página (n.v-1)
T. TABLA = cant. de entrada * bit de marco

INDICE	OFFSET

↳ Si la página que buscamos no está cargada en memoria (su bit de presente/ausente está en 0) se produce Fallo de Páginas o Page fault → Existen algoritmos para elegir qué pág bajar a disco → FIFO → NRU → LRU → clock → ...

Pasos a realizar en Page Fault

① comprobamos una tabla interna (que usualmente se mantiene con el bloque de control del proceso) correspondiente a ese proceso, para determinar si la referencia era un acceso de memoria válido e inválido.

② Si la referencia era inválida, terminamos el proceso. Si era válida pero esa página todavía no ha sido cargada, la cargamos en memoria.

③ Buscamos un marco libre

④ ordenamos una operación de disco para leer la página deseada en el marco recien asignado

⑤ Una vez completada la lectura de disco, modificamos la tabla interna que se mantiene con los datos del proceso y la tabla de páginas para indicar que dicha página se encuentra ahora en memoria.

⑥ Reiniciamos la instrucción que fue interrumpida. El proceso podrá ahora acceder a la página como si siempre hubiera estado en memoria.

NOTA

HOJA Nº

FECHA

Tablas De Páginas

Multinivel

↳ si tenemos mucha memoria virtual la tabla de páginas puede llegar a ocupar mucha memoria RAM

↳ con la tabla Multinivel → solo tenemos en memorias las partes que estan en uso.

→ la paginación nos baja el rendimiento del sistema (RAM lenta respecto CPU)
Solución: metemos una caché.

↳ TLB (asociativa) → la MMU primero se fija en la TLB y si no esta ahí, se fija en la tabla de páginas. (actualiza TLB)

RESUMIENDO:

El manejo de memoria es un sistema multiprogramado → (desafíos)
Para aumentar el rendimiento del sistema, el grado de multiprogramación se aumenta
Pudiendo no poder entrar todos los procesos en memoria.
memoria virtual → paginación
Paginación eficiente → MMU, tabla multinivel, TLB, algo. reemplazo, etc.

PAGINACIÓN → TRANSPARENTE al programador

MEMORIA FÍSICA vs VIRTUAL → TÉCNICA de administración de memoria, que permite a los usuarios ejecutar programas más grandes que la memoria física real. La parte de la memoria virtual que no esta direccionada a la RAM, esta direccionada al disco.

se refiere a la memoria RAM real del sistema conectada a la placa base

NOTA