Resumen Primer Parcial - Sistemas Operativos

Universidad de Antioquia

Emmanuel Bustamante Valbuena

3 de junio de 2025

Resumen sobre todos los temas que caen en el primer parcial de Sistemas Operativos

1. Virtualización de recursos

1.1. Definición y objetivos

- **Virtualización**: Proceso por el cual el sistema operativo (SO) hace creer a los procesos que disponen de recursos exclusivos, cuando en realidad dichos recursos (CPU, memoria, dispositivos de E/S) son compartidos o particionados.
- **Objetivo principal**: Maximizar la utilización de recursos y ofrecer aislamiento entre procesos.

1.2. Tipos de virtualización

1) **Virtualización de CPU**
   - *Multiprogramación*: Varios procesos comparten la CPU mediante técnicas de planificación (time-sharing).
   - Cada proceso cree que dispone de la CPU de forma continua, aunque se le asignan *cuantos de tiempo* breves.

2) **Virtualización de Memoria**
   - Cada proceso tiene su propio espacio de direcciones.
   - El SO traduce direcciones lógicas (virtuales) a direcciones físicas mediante tablas de páginas o segmentos.
   - Permite que cada proceso piense que dispone de toda la memoria, aunque físicamente esté compartida.

3) **Virtualización de Dispositivos de I/O**
   - Se abstraen los dispositivos (discos, impresoras, puertos, etc.) para que un proceso vea un dispositivo "virtual" exclusivo.
   - El kernel se encarga de multiplexar el acceso real al hardware.

1.3. Ventajas y desafíos

- **Aislación**: Un fallo o mal comportamiento de un proceso no suele afectar a otros, si la virtualización está correctamente implementada.
- **Seguridad**: Separar espacios de direcciones impide accesos no autorizados entre procesos.
- **Utilización**: Permite que la CPU y dispositivos de E/S estén ocupados casi todo el tiempo.
- **Complejidad**: El kernel debe gestionar:
  - Tablas de traducción de direcciones (MMU).
  - Switching de contextos (interrupciones de reloj, traps).
  - Multiplexación de I/O.
- **Overhead**:
  - *Tiempo*: Traducción de direcciones, cambios de contexto.
  - *Espacio*: Estructuras de datos (tablas de páginas, TLB).

2. Procesos

2.1. Concepto y estados básicos

- **Proceso**: Instancia en ejecución de un programa. Incluye:
  - Código y datos.
  - Pila (stack).
  - Contexto de CPU (registros, PC).
  - Estado (Ready, Running, Blocked, etc.).
- **Estados comunes**:
  a) **Nuevo (New)**: Proceso creado, pendiente de inicializar.
  b) **Listo (Ready)**: Preparado para ejecutarse; espera que el planificador le asigne la CPU.
  c) **Ejecución (Running)**: Actualmente utilizando la CPU.
  d) **Bloqueado (Blocked/Waiting)**: Espera un evento (E/S, semáforo, señal).
  e) **Terminado (Terminated/Exit)**: Ha finalizado su ejecución.

2.2. Control de procesos y PCB

- Cada proceso está representado en el kernel por una **PCB (Process Control Block)**, que contiene:
  - **PID**: Identificador único.
  - **State**: Estado actual (Ready, Running, Blocked).

- **Registers**: Valores de registros de CPU cuando el proceso no está en ejecución.
- **PC (Program Counter)**: Dirección de la siguiente instrucción a ejecutar.
- **RTC (Runtime Counters)**: Contadores de uso de CPU, tiempo de espera, etc.
- **Información de E/S**: Dispositivos/archivos abiertos.
- **Listas de sincronización**: Semáforos, variables de condición, etc.

2.3. Jerarquía de procesos (árbol de procesos)

- Cada proceso tiene un proceso padre (excepto el proceso inicial).
- Si un proceso padre finaliza antes que sus hijos, éstos pasan a estado "zombi" hasta que el padre los recoja con `wait()`.

3. Mecanismos: Ejecución Directa Limitada

3.1. Concepto de Ejecución Directa (Direct Execution)

- **Ejecución Directa**: Ejecutar el código de usuario directamente sobre el procesador, sin intervención continua del kernel, para mejorar rendimiento.
- **Problema**: Si el proceso en modo usuario ejecuta instrucciones privilegiadas (acceso a E/S, manipulación de MMU), el hardware genera una *trampa (trap)* hacia el kernel.

3.2. Kernel mínimo en las trampas

- El kernel sólo interviene cuando ocurren eventos que requieren privilegios (fallo de página, petición de E/S, interrupción de reloj).
- Mientras no haya eventos de privilegio, el proceso corre "directamente" en modo usuario.

3.3. Limitaciones y seguridad

- Es imprescindible establecer *protección de memoria* e instrucciones privilegiadas.
- Cuando se cumple el *quantum de tiempo* asignado, un temporizador genera una *interrupción de reloj* para forzar una trampa y permitir cambio de contexto.
- Sin esta limitación, un proceso en modo usuario podría monopolizar la CPU indefinidamente.

4. Planificación de Procesos

4.1. Objetivos de la planificación

- **Utilización máxima de la CPU**: Mantener la CPU ocupada.
- **Throughput**: Número de procesos completados por unidad de tiempo.
- **Tiempo de respuesta**: Desde que un proceso solicita CPU hasta que comienza a ejecutarse (crítico en sistemas interactivos).
- **Tiempo de espera**: Tiempo total que un proceso pasa en colas (Ready).
- **Fairness (Justicia)**: Evitar inanición (starvation).

4.2. Clasificación de planificadores

1) **Long-term scheduler (job scheduler)**
   - Decide qué trabajos ingresan al sistema.
   - En muchos sistemas de tiempo compartido es menos visible, pues se cargan todos los trabajos de inmediato.
2) **Short-term scheduler (CPU scheduler)**
   - Selecciona qué proceso en estado Ready pasará a Running.
   - Se ejecuta con alta frecuencia (cada interrupción de reloj, evento de E/S, etc.).
3) **Medium-term scheduler (swapper)**
   - Responsable de la *suspensión* (swapping) de procesos de memoria a disco para controlar grado de multiprogramación.
   - Reintroduce procesos suspendidos cuando hay memoria disponible.

4.3. Algoritmos de planificación

4.3.1. First-Come, First-Served (FCFS)

- Ordena según el orden de llegada a la cola Ready.
- **Ventaja**: Muy simple de implementar.
- **Desventaja**: *Efecto convoy*: un trabajo largo puede bloquear a varios cortos.
- El tiempo de espera promedio suele ser alto cuando hay ráfagas muy desiguales.
- Tiempo de espera promedio (aprox):

$$\frac{(Tturnaronda)}{q}$$

4.3.2. Shortest-Job-First (SJF)

- Selecciona el proceso con menor tiempo de CPU esperado.
- Óptimo (*minimiza tiempo de espera promedio*) si se conoce la duración exacta de las ráfagas.
- **No expropiativo**: el proceso que comienza a ejecutarse no se desaloja hasta terminar su ráfaga.
- **Expropiativo (Shortest Remaining Time First, SRTF)**: si llega un proceso con tiempo restante menor que el del proceso en ejecución, se desaloja.
- **Problema**: Dificultad para predecir el tiempo de ráfaga.

4.3.3. Priority Scheduling

- A cada proceso se le asigna una *prioridad* (puede ser estática o dinámica).
- Expropiativo: si llega un proceso con prioridad más alta, desaloja al actual.
- **Problema**: Inanición de procesos de baja prioridad.
- **Solución**: Envejecimiento (*aging*), incrementando la prioridad de procesos que esperan mucho tiempo.

4.3.4. Round Robin (RR)

- Variante de time-sharing: cada proceso recibe un *quantum* de tiempo (q).
- Si no termina en q, se coloca al final de la cola Ready.
- **Ventaja**: Buen tiempo de respuesta en sistemas interactivos.
- **Desventaja**:
  - Si q es muy grande, se parece a FCFS.
  - Si q es muy pequeño, el overhead de cambio de contexto crece.
- Tiempo de espera promedio (aprox):

$$\frac{(N-1)\,q}{2},$$

donde N es el número de procesos con ráfaga de CPU mayor que q.

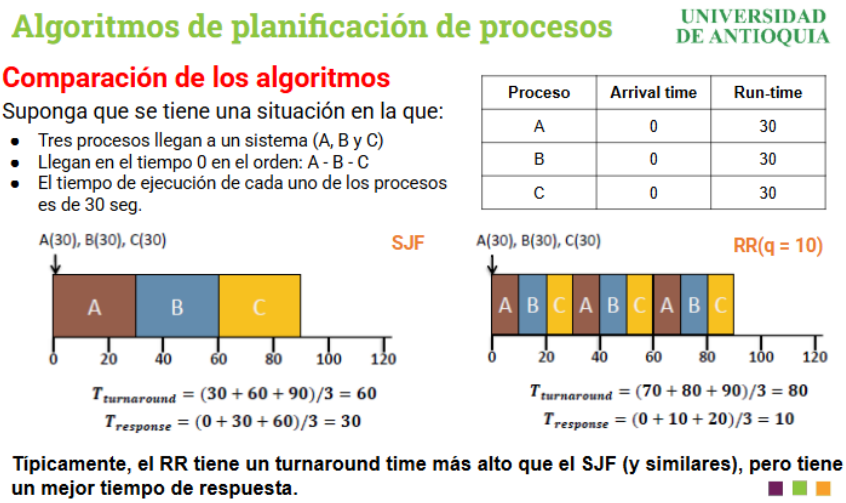

Figura 1: Comparación de algoritmos: SJF vs Round Robin con $q = 10$.

4.3.5. Multi-level Queue (Colas multinivel)

- Se definen varias colas con *diferentes niveles de prioridad* y *políticas propias*.
- Ejemplo:
  - Cola 0 (alta prioridad): Procesos interactivos (RR, quantum pequeño).
  - Cola 1: Procesos batch (FCFS).
- Cada proceso, al llegar, se asigna a una cola fija según categoría.
- **Desventaja**: Rígido; un proceso no migra entre colas.

4.3.6. Multi-level Feedback Queue (Colas de retroalimentación)

- Permite *movimiento entre colas* según comportamiento de CPU/E/S.
- Mecanismo típico:
  1) Nuevos procesos entran en la cola de mayor prioridad con quantum pequeño.
  2) Si agotan su quantum sin bloquearse, se degradan a cola de menor prioridad.
  3) Si un proceso se bloquea por E/S, al volver se reincorpora a la misma cola (o se le puede premiar).
- **Ventaja**: Adapta prioridades dinámicamente, logra buen balance entre throughput y tiempo de respuesta.
- **Parámetros clave**: Número de colas, tamaño inicial del quantum, reglas de escalonamiento (ascenso/descenso).

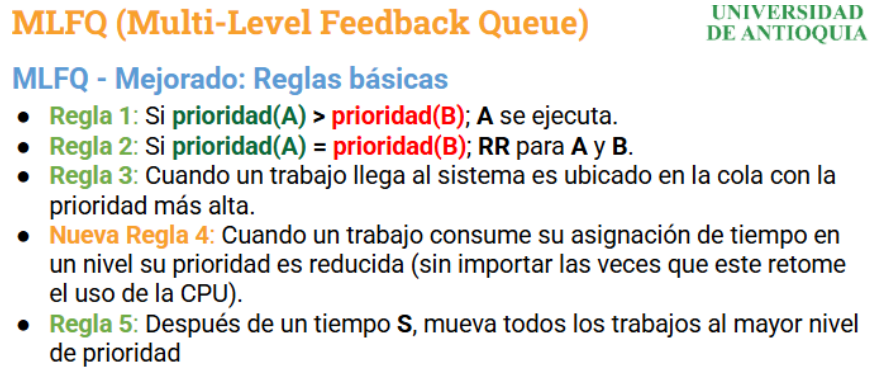

Figura 2: Reglas basicas para MLFQ

5. Planificación multinivel

5.1. Motivación y características

- Existen distintas *categorías de procesos* que requieren políticas diferentes (interactivo, batch, tiempo real).
- Se usan *colas separadas*, cada una con su propio algoritmo de planificación y quantum.
- Se define una *prioridad de cola*: los procesos en colas de nivel superior se atienden antes que los de nivel inferior.

5.2. Esquemas típicos

1) **Colas estáticas (Multi-level Queue Scheduling)**
   - Cada proceso, al crearse, se asigna a una cola fija.
   - Ejemplo de asignación:
     - Cola 0: Procesos de sistema (prioridad alta, FCFS).
     - Cola 1: Procesos interactivos (prioridad media, RR).
     - Cola 2: Procesos batch (prioridad baja, FCFS).
   - No hay movimiento entre colas.

2) **Colas con retroalimentación (Multi-level Feedback Queue)**
   - Permiten *movimiento entre colas* según comportamiento de CPU/I/O (como en la sección anterior).

5.3. Problemas comunes y cómo abordarlos

- **Inanición de cola baja**: Si siempre hay procesos en cola alta, colas inferiores nunca se atienden.
  - *Soluciones*:

- Envejecimiento: Desplazar procesos de cola inferior a superior tras cierto tiempo.
- Reservar un porcentaje de CPU para colas bajas.

- **Balanceo de prioridades**: Ajustar quantum y reglas de escalonamiento para que procesos I/O-bound (que liberan CPU pronto) no se degraden demasiado rápido.

6. Virtualización de Memoria

6.1. Espacio de direcciones

- Cada proceso utiliza *direcciones virtuales* que deben mapearse a direcciones físicas.
- **Objetivos**:
  - Proporcionar un espacio contiguo para cada proceso, aunque la memoria física esté particionada o fragmentada.
  - Permitir ubicación no contigua en memoria física.
  - Aislamiento entre procesos.

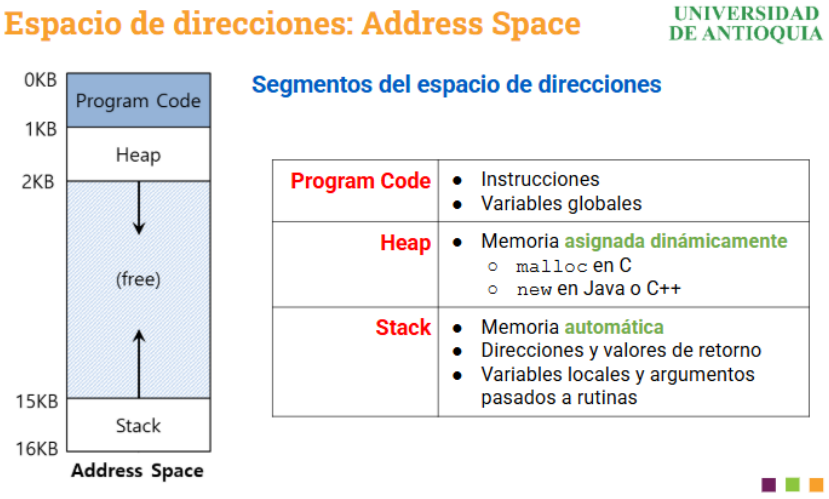

Figura 3: Address space figure

6.2. Tabla de páginas

- **Páginas**: Bloques de tamaño fijo (por ejemplo, 4 KB).
- **Marcos de página (frames)**: Bloques de memoria física del mismo tamaño.
- Cada proceso tiene su *tabla de páginas* con, para cada página virtual:

- **Bit de presencia (P)**: Indica si la página está cargada en memoria física.
- **Número de marco (frame number)**: Si $P = 1$, apunta al frame; si $P = 0$, señala ubicación en disco (swap).
- **Bits de control**:
  - Lectura/Escritura (R/W).
  - Bit de referencia (R).
  - Bit de modificado (dirty).
  - Otros según arquitectura.

6.3. Proceso de traducción de direcciones

1) El proceso genera una *dirección virtual*, dividida en:
   - **Índice de página (p)**: Bits altos para indexar en la tabla de páginas.
   - **Desplazamiento (offset)**: Bits bajos que indican la posición dentro de la página.
2) El MMU (Memory Management Unit) consulta la tabla de páginas del proceso:
   - Si $P = 1$: concatena el número de frame con el offset → dirección física.
   - Si $P = 0$: genera un *fallo de página* (page fault). El SO debe:
     1) Cargar la página desde disco a memoria.
     2) Actualizar la tabla de páginas (bit $P = 1$, asignar frame).
     3) Posiblemente expulsar otra página (según política de reemplazo).
3) La CPU accede finalmente a la dirección física resultante.

Traducción de direcciones

UNIVERSIDAD DE ANTIOQUIA

Ejemplo:

Un proceso con un espacio de direccionamiento de 4KB fue asignado a una memoria física a partir de 16KB. Teniendo en cuenta lo anterior, realizar la traducción de direcciones virtuales a físicas mostradas en la siguiente tabla:

Virtual Address (VA)	Physical Address (PA)
0	16 KB
1 KB	17 KB
3300	19684
4400	Fault (Out of bounds)

Figura 4: Ejemplo Traducción

7. Paginación

7.1. Ventajas de la paginación

- **Elimina la fragmentación externa**: La memoria física se organiza en *frames* fijos, y los procesos en *páginas* del mismo tamaño.
- **Asignación no contigua**: Un proceso puede ocupar frames dispersos.
- **Compartición de páginas**: Varias instancias de un programa pueden compartir páginas de solo-lectura (código) en frames comunes.

7.2. Diseño de la tabla de páginas

- **Paginación invertida**:
  - Se mantiene una *tabla global* con entradas (frame $\rightarrow$ proceso, número de página).
  - Reduce el espacio de tablas pero hace más costosa la búsqueda.
- **Tabla multinivel**:
  - Cuando el espacio de direcciones virtuales es grande (64 bits), la tabla completa no cabe en memoria.
  - Se divide en niveles:
    1) Nivel 1: Directorio de páginas.
    2) Nivel 2: Tabla de páginas propiamente dicha.
    3) (Para 64 bits, pueden haber 3 o más niveles.)
  - Solo las partes usadas de la tabla se crean en memoria, reduciendo consumo.

7.3. Ejemplos sobre direccionamiento de memoria

Ejemplo de clase: 5.

1. **¿Cuántos bits se necesitan para direccionar la memoria física?**
   Se tiene un espacio de direcciones físicas de 128 bytes, por lo tanto:

   $$2^n = 128 \Rightarrow n = \log_2(128) = 7$$

   Se requieren **7 bits** para direccionar la memoria física.

2. **¿Cuántos bits se necesitan para hacer referencia a los marcos (frames)?**
   El número de marcos es:

   $$\frac{128}{16} = 8 \Rightarrow 2^n = 8 \Rightarrow n = \log_2(8) = 3$$

   Se requieren **3 bits** para referenciar los marcos.

3. **¿Cuántos bits se necesitan para el desplazamiento (offset)?**
   Sabemos que:
   $$n(\text{PA}) = n(\text{PFN}) + n(\text{offset})$$
   $$7 = 3 + n(\text{offset}) \Rightarrow n(\text{offset}) = 4$$
   Se requieren **4 bits** para el offset.

4. **¿Cómo es el formato de la dirección para este ejemplo?**
   La dirección física se compone de 7 bits, divididos en:
   - **Bits altos (PFN)**: 3 bits.
   - **Bits bajos (offset)**: 4 bits.

   La estructura se muestra a continuación:

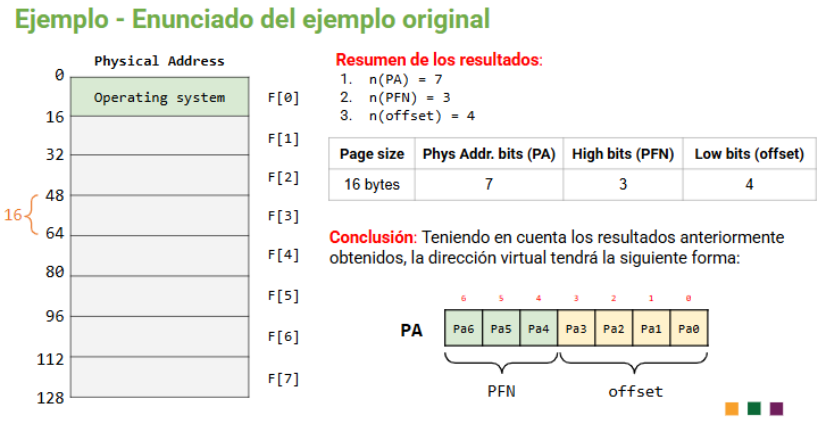

Figura 5: Ejemplo de formato de dirección física con 7 bits, considerando un tamaño de página de 16 bytes.

7.4. Tasa de fallos de página y working set

- **Working set**: Conjunto de páginas que un proceso utiliza activamente en un intervalo de tiempo (por ejemplo, páginas referenciadas recientemente).
- Si el número de frames asignados a un proceso es menor que su working set, aumenta la *tasa de fallos de página*, generando overhead por swapping.
- Algoritmo del Working Set:
  - El SO calcula (o aproxima) el conjunto activo y ajusta la asignación de frames para mantener baja la tasa de fallos.
- **Thrashing**:
  - Situación en la que los procesos pasan la mayor parte del tiempo haciendo *swapping* (fallos de página), en lugar de ejecutar instrucciones.
  - Síntoma: CPU inactiva, pero mucho tráfico de E/S al disco.

7.5. Ejemplo de traducción de direcciones

A continuación se muestra una memoria virtual y una física que almacenan datos de 1 byte. Teniendo en cuenta la información de la figura 6, se responde lo siguiente:

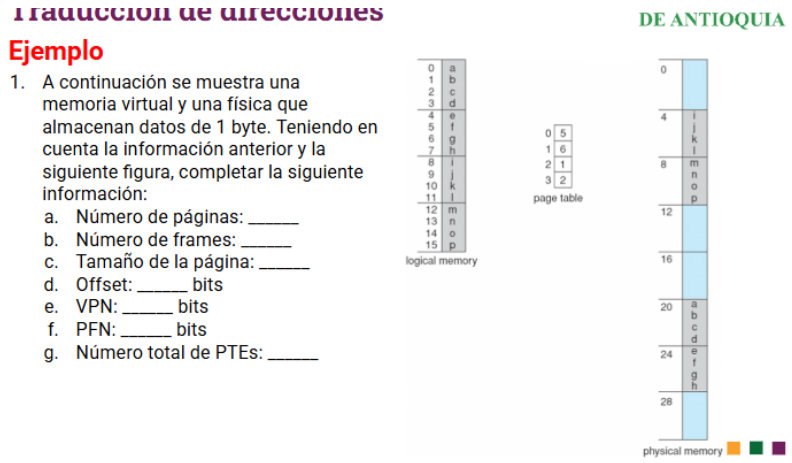

Figura 6: Ejemplo de memoria lógica, tabla de páginas y memoria física.

1. **Número de páginas:** La memoria lógica contiene 16 bytes, y el tamaño de cada página es de 4 bytes. Por tanto:

$$\frac{16 \text{ bytes}}{4 \text{ bytes/página}} = 4 \text{ páginas}$$

2. **Número de marcos (frames):** La memoria física contiene 32 bytes. Según la organización observada, cada frame tiene 8 bytes:

$$\frac{32 \text{ bytes}}{8 \text{ bytes/frame}} = 4 \text{ frames}$$

3. **Tamaño de la página:** De acuerdo con la segmentación de la memoria lógica:

$$\frac{16 \text{ bytes}}{4 \text{ páginas}} = 4 \text{ bytes por página}$$

4. **Offset:** Para direccionar 4 posiciones dentro de una página se necesitan:

$$2^n = 4 \Rightarrow n = 2 \text{ bits}$$

5. **VPN (Virtual Page Number):** Existen 4 páginas, por tanto:

$$2^n = 4 \Rightarrow n = 2 \text{ bits}$$

6. **PFN (Page Frame Number):** Existen 4 frames, por tanto:

$$2^n = 4 \Rightarrow n = 2 \text{ bits}$$

7. **Número total de entradas en la tabla de páginas (PTEs):** Se observan 4 entradas numeradas del 0 al 3.

8. Acelerando la traducción de direcciones: TLB (Translation Lookaside Buffer)

8.1. Concepto y funcionamiento

- El **TLB** es una memoria caché en hardware que almacena entradas recientes de traducción ⟨virtual-page, physical-frame⟩.
- Al generar una dirección virtual, el MMU primero busca en el TLB:
  - **Hit**: La traducción se obtiene inmediatamente, sin acceder a la tabla de páginas en memoria.
  - **Miss**: Se consulta la tabla de páginas en memoria principal, se actualiza el TLB y luego se procede.
- **Ventaja**: Reduce drásticamente los accesos a memoria para traducciones, acelerando la ejecución.

8.2. Políticas de reemplazo en TLB

- Debido al tamaño reducido del TLB (por ejemplo, 64 o 128 entradas), se aplican algoritmos de reemplazo similares a caches:

1) **LRU (Least Recently Used)**: Reemplaza la entrada menos recientemente utilizada.

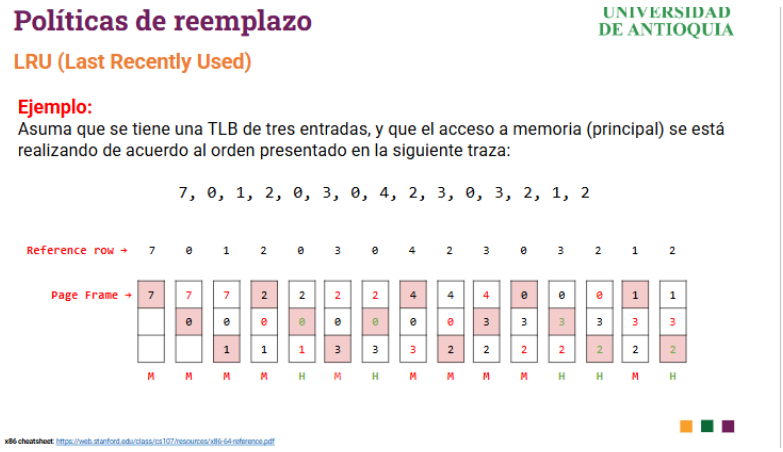

Figura 7: Ejemplo de politica LRU.

2) **FIFO**: Reemplaza la entrada más antigua.
3) **Random**: Selección aleatoria; a veces competitivo y fácil de implementar.

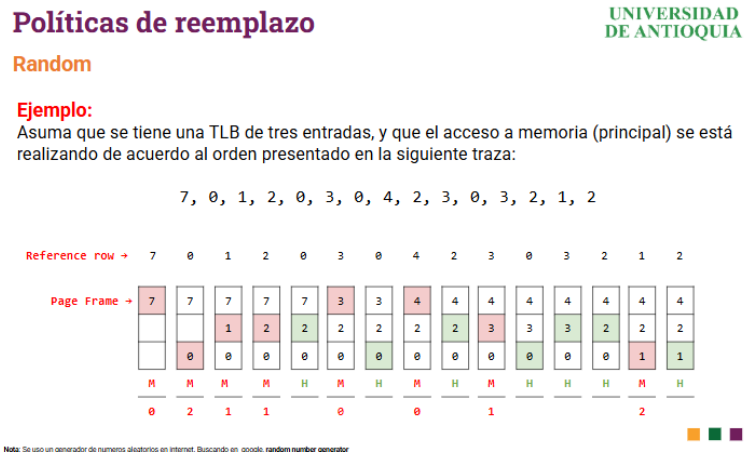

Figura 8: Ejemplo de politica random.

8.3. Consistencia y tablas multinivel

- En arquitecturas con *tablas multinivel*, la traducción normal implica múltiples accesos (uno por cada nivel).
- El TLB almacena la traducción final (nivel $k \rightarrow$ frame), omitiendo accesos intermedios.
- **Actualización del TLB**:
  - Cuando un proceso se bloquea o la asignación de páginas cambia (fallo de página, swapping), el SO debe invalidar (flush) entradas del TLB.
  - Alternativamente, usar *TLB tags* que incluyan el PID para identificar traducciones por proceso.

9. Más allá de la memoria física: mecanismos

Cuando la memoria física no es suficiente para alojar todas las páginas activas de los procesos, el SO emplea estrategias para extender el espacio de direcciones.

9.1. Swapping (Intercambio)

- Consiste en pasar *completamente* un proceso de la memoria principal al disco (área de swap), liberando todos sus frames.
- Cuando se reanuda el proceso, se lee nuevamente del disco.
- **Desventaja**: Gran overhead en E/S, pero fácil de implementar.

9.2. Paging (Paginación por demanda)

- Al iniciar un proceso, solo se cargan en memoria las páginas estrictamente necesarias (código y datos).
- El resto de páginas permanece en swap o archivo de paginación en disco.
- Cuando el proceso accede a una página no cargada, se produce un *fallo de página*, y el SO la trae desde disco.

9.3. Segmentación y paginación combinadas

- Algunos SOs emplean *segmentación* (división por módulos lógicos: código, pila, datos) junto con paginación interna en cada segmento.
- Proceso de traducción:
  1) Traducir (segmento, offset) → dirección virtual.
  2) Aplicar paginación a la dirección virtual resultante (índice de página → consulta en tabla de páginas).

9.4. Archivos de paginación (Swap files)

- Alternativa a la partición de swap: archivos normales en disco que actúan como área de intercambio.
- **Ventaja**: Flexibilidad para redimensionar.
- **Desventaja**: Velocidad ligeramente menor que particiones dedicadas (depende del sistema de ficheros).

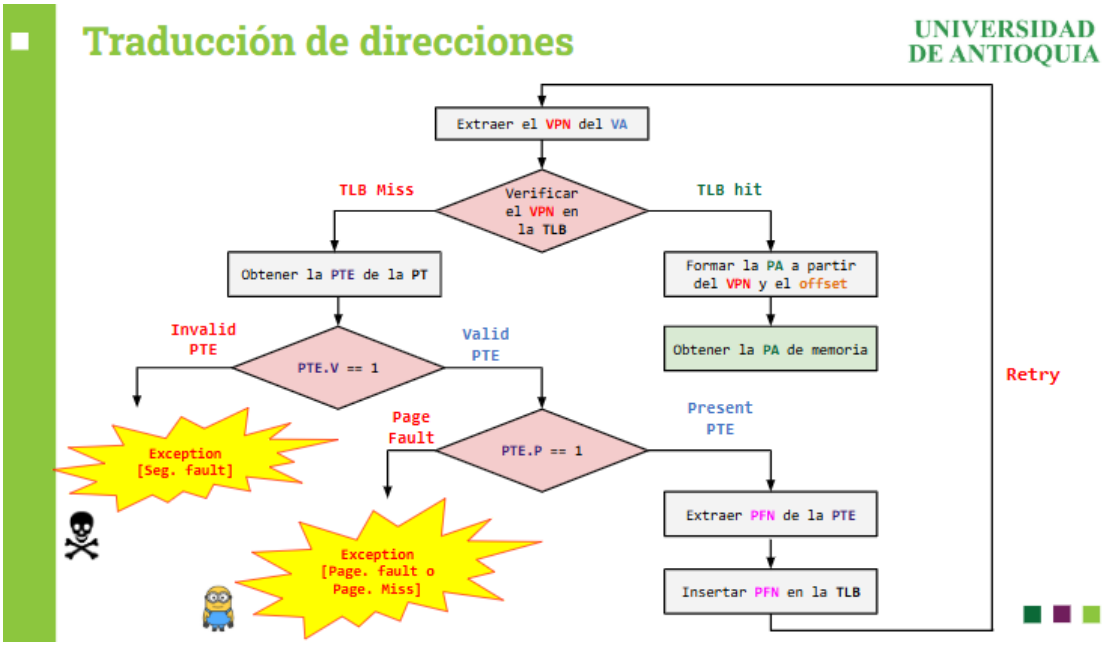

Figura 9: Diagrama de traducción de direcciones.

10. Más allá de la memoria física: políticas

Cuando ocurre un fallo de página y no hay frames libres, el SO elige una página "víctima" para expulsar. A continuación se describen políticas principales:

10.1. Políticas de reemplazo de páginas

1) **FIFO (First-In, First-Out)**

- Expulsa la página que lleva más tiempo en memoria.
- Fácil de implementar con una cola.
- No considera recientidad ni frecuencia de uso.

2) **LRU (Least Recently Used)**

- Expulsa la página a la que *no se ha accedido durante más tiempo.*
- Óptimo teórico si la referencia pasada predice la futura.
- **Costo**: Mantener seguimiento exacto de recientidad es caro (requiere relojes, contadores, bits de referencia).
- *Aproximaciones*:
  - **Clock o Second Chance**: Página circular con bit de referencia R. Si $R = 1$, se limpia a 0 y se mueve al final; si $R = 0$, se expulsa.
  - **NFU (Not Frequently Used)** con envejecimiento: Contador periódico de referencias.

3) **Optimal (Belady's Algorithm)**

- Expulsa la página que no se usará hasta más tarde en el futuro.
- Teórico: Minimiza tasa de fallos global.
- Impracticable: Requiere conocer patrones futuros de referencia.

10.2. Local vs Global Replacement

- **Local**: Cada proceso sólo puede expulsar páginas de su propia asignación de frames.
- **Global**: En un fallo de página, el reemplazo puede elegir cualquier página de cualquier proceso.
- **Consecuencias**:
  - *Global* tiende a mejores tasas de uso de frames, pero puede ser injusto: un proceso puede acaparar frames de otro.
  - *Local* respeta límites, pero si un proceso tiene pocos frames y alta demanda, tendrá más fallos de página.

10.3. Cuánto asignar

- **Asignación fija**: Cada proceso recibe un número fijo de frames (por ejemplo, tamaño del proceso ÷ número total de procesos).
- **Asignación dinámica**: Basada en working set, tamaño real del proceso, prioridad.

11. Ejemplos de resolución de problemas

11.1. Cálculo de tiempo de espera en Round Robin

Enunciado: Dado un conjunto de procesos con tiempos de CPU (en milisegundos):

- P1: 24 ms
- P2: 3 ms
- P3: 3 ms

Con un *quantum* de 4 ms, calcular el tiempo de espera promedio.

Solución paso a paso

- Todos los procesos llegan en $t = 0$. Cola Ready inicial: [P1, P2, P3].
- Se va creando la tabla de Gantt y actualizando los tiempos:

Tiempo	Ejecución	Cola Ready después	Comentario
0 – 4	P1 (4 ms)	[P2, P3, P1(20 ms restantes)]	P1 usado 4 ms, quedan 20 ms.
4 – 7	P2 (3 ms)	[P3, P1, P2(0 ms restantes → terminado)]	P2 termina en $t = 7$.
7 – 10	P3 (3 ms)	[P1, P3(0 ms restantes → terminado)]	P3 termina en $t = 10$.
10 – 14	P1 (4 ms)	[P1(16 ms restantes)]	P1 usado otros 4 ms, quedan 16 ms.
14 – 18	P1 (4 ms)	[P1(12 ms restantes)]	P1 usado, quedan 12 ms.
18 – 22	P1 (4 ms)	[P1(8 ms restantes)]	P1 usado, quedan 8 ms.
22 – 26	P1 (4 ms)	[P1(4 ms restantes)]	P1 usado, quedan 4 ms.
26 – 30	P1 (4 ms)	[P1(0 ms restantes → terminado)]	P1 finaliza en $t = 30$.

Cuadro 1: Ejecución Round Robin con quantum = 4 ms

Tiempos de finalización

- P2: 7 ms
- P3: 10 ms
- P1: 30 ms

Tiempos de retorno (Turnaround)

$$\begin{aligned}\text{P2: } &7\,\text{ms} - 0\,\text{ms} = 7\,\text{ms}\\ \text{P3: } &10\,\text{ms} - 0\,\text{ms} = 10\,\text{ms}\\ \text{P1: } &30\,\text{ms} - 0\,\text{ms} = 30\,\text{ms}\end{aligned}$$

Tiempos de espera

$$\text{Tiempo de espera} = \text{Turnaround} - \text{Burst (ráfaga)}.$$

$$\begin{aligned}\text{P2: } &7\,\text{ms} - 3\,\text{ms} = 4\,\text{ms}\\ \text{P3: } &10\,\text{ms} - 3\,\text{ms} = 7\,\text{ms}\\ \text{P1: } &30\,\text{ms} - 24\,\text{ms} = 6\,\text{ms}\end{aligned}$$

Tiempo de espera promedio

$$\frac{4+7+6}{3} = \frac{17}{3} \approx 5{,}67\,\text{ms}.$$

Consejo : En examen, construir siempre la tabla de Gantt para Round Robin y extraer tiempos de espera, retorno y respuesta sistemáticamente.

11.2. Fallos de página con algoritmo FIFO

Enunciado: Secuencia de referencias de páginas: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$. Número de frames: 3. Calcular tasa de fallos usando FIFO.

Solución

Partimos con todos los frames vacíos. Llevamos un puntero FIFO al frame más antiguo.

1) Ref 1 → fallo (cargar 1). Frames: [1, –, –]
2) Ref 2 → fallo (cargar 2). Frames: [1, 2, –]
3) Ref 3 → fallo (cargar 3). Frames: [1, 2, 3]
4) Ref 4 → fallo → reemplazar FIFO: expulsa 1. Frames: [4, 2, 3]
5) Ref 1 → fallo → expulsa 2. Frames: [4, 1, 3]
6) Ref 2 → fallo → expulsa 3. Frames: [4, 1, 2]
7) Ref 5 → fallo → expulsa 4. Frames: [5, 1, 2]
8) Ref 1 → hit (está en memoria). Frames: [5, 1, 2]
9) Ref 2 → hit. Frames: [5, 1, 2]
10) Ref 3 → fallo → expulsa 5. Frames: [3, 1, 2]
11) Ref 4 → fallo → expulsa 1. Frames: [3, 4, 2]
12) Ref 5 → fallo → expulsa 2. Frames: [3, 4, 5]

- **Total referencias**: 12.
- **Fallos**: 10 (en las referencias 1, 2, 3, 4, 5, 6, 7, 10, 11, 12).
- **Tasa de fallos**: $\frac{10}{12} \approx 0{,}83$ (83 %).

Consejo : Mantener una lista con los frames y actualizar tras cada referencia. Llevar un puntero FIFO para la expulsión.

11.3. Traducción de direcciones con TLB

Enunciado: Sistema con páginas de 4 KB (12 bits de offset). La TLB contiene la siguiente entrada:

Virtual Page (VP)	Physical Frame (PF)	Bits de control
0x004	0x00A	P = 1, R = 1, W = 0

Cuadro 2: Entrada ejemplo en la TLB

Si el proceso genera la dirección virtual `0x004123`, ¿cuál es la dirección física resultante? ¿Qué ocurre si la TLB falla?

Solución

1) **Extraer campos**:
   - Dirección virtual: `0x004123` (hexadecimal).
   - Dividir en:
     - *VP* (bits altos): `0x004`.
     - *Offset* (12 bits bajos): `0x123`.
2) **Buscar en TLB**:
   - La VP `0x004` está en la TLB con PF = `0x00A`.
3) **Cálculo de dirección física**:

   $$\text{Dirección física} = (\text{PF} \ll 12) + \text{offset} = \texttt{0x00A000} + \texttt{0x123} = \texttt{0x00A123}.$$

4) **Si la TLB falla** (Miss):
   1) El MMU consulta la tabla de páginas en memoria principal.
   2) Supongamos que la tabla de páginas indica PF = `0x00A`, P = 1.
   3) Se inserta en la TLB la entrada (`0x004` → `0x00A`).
   4) Luego se obtiene la misma dirección física `0x00A123` y se continúa la ejecución.

12. Resumen de conceptos clave y consejos de examen

13. Recomendaciones para resolver problemas en el examen

1) **Organiza tus ideas**: Al leer el enunciado, subraya datos clave: número de frames, secuencia de referencias, tiempos de CPU, prioridades, etc.
2) **Construye tablas de seguimiento**:
   - Para planificación Round Robin: dibuja la línea de tiempo (Gantt chart) y marca cuándo cada proceso entra y sale.
   - Para fallos de página: dibuja la lista de frames y actualiza tras cada referencia, anotando fallo/acierto y página expulsada.
3) **Cálculos importantes**:
   - Ten clara la fórmula: Tiempo de espera = Turnaround − Burst.
4) **Distingue políticas y mecanismos**:
   - *Algoritmo* (¿qué hacer en cada evento?) vs *mecanismo* (¿cómo lo implementa el SO a bajo nivel?).

Tema	Conceptos clave
Virtualización de recursos	Multiprogramación, aislamiento, trampas, ejecución directa limitada.
Procesos	PCB, estados (Ready, Running, Blocked), cambio de contexto.
Planificación	FCFS, SJF, RR, Prioridad, Multinivel. Cálculo de tiempos de espera, retorno, respuesta.
Planificación multinivel	Colas estáticas vs colas de retroalimentación, envejecimiento, evitar inanición.
Virtualización de memoria	Direcciones virtuales, MMU, tablas de páginas, bits P/R/W, fallos de página.
Paginación	Eliminación de fragmentación externa, frames, working set, thrashing.
TLB	Caché de traducciones, hit/miss, reemplazo (LRU, Clock).
Más allá de memoria física	Swapping, paginación por demanda, segmentación + paginación.
Políticas de reemplazo de páginas	FIFO, LRU, Second-Chance, Optimal, NFU, local vs global.
Mecanismos de sustitución	Asignación fija vs dinámica (working set, WSClock).

Cuadro 3: Resumen de temas y conceptos clave

- Ejemplo: Traducción de direcciones (mecanismo hardware/software) vs política de reemplazo de páginas (FIFO, LRU, etc.).

5) **Memoriza tamaños y dominios de bits**:

- Páginas de 4 KB $\Rightarrow$ 12 bits de offset.
- Si aparece un ejemplo con 32 bits de dirección y páginas de 4 KB, la tabla de páginas necesita $32 - 12 = 20$ bits de índice.

6) **Interpretación de TLB**:

1) Extraer VP y offset.
2) Buscar en TLB $\rightarrow$ hit/miss.
3) Si miss, consultar tabla de páginas.
4) Calcular dirección física.

Resumen hecho por Emmanuel Valbuena y un poquito de IA jeje