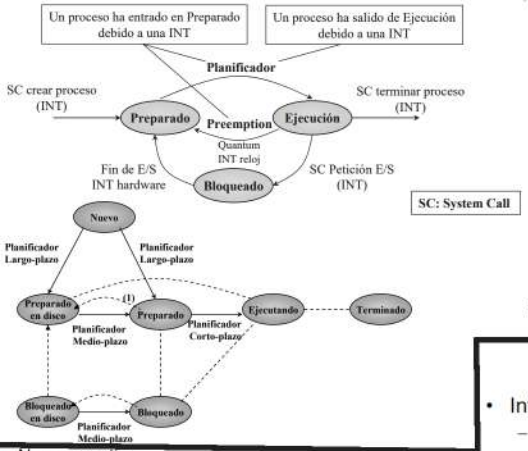
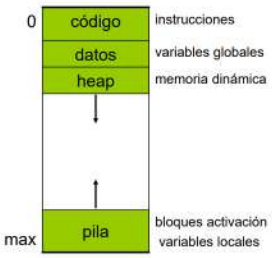


Partes de un Proceso



+ registros CPU (pc, sp, cpsr(psw), ...)

Bloque de Control de Proceso (PCB)

- Process Control Block: contiene toda la información relativa a un proceso
  - Identificador
  - Estado
  - Prioridad
  - Padre e hijos
  - contexto I
    - Punteros a zonas de memoria
  - contexto II
    - pc, cprs, registros
  - Info E/S
    - Peticiones pendientes
    - Disp. E/S asignados
    - Ficheros abiertos
  - Info Contabilidad
    - Tiempos acumulados
    - Fechas
    - Cuotas (memoria, tiempo, ...)
- Cada vez que un proceso cambia de estado, el SO anota los cambios en los campos apropiados de su PCB
- Tabla de Procesos: un struct PCB por proceso

Asignación de prioridades

- Interna / Externa
  - Definida a partir de parámetros obtenidos por el propio SO
  - Definida por el administrador en base a tipo de usuario, importancia del trabajo, cuota que se paga, …
- Estática / Dinámica
  - Estable durante toda la vida del proceso
  - Varía durante la ejecución en función del patrón de uso de recursos, de la ocupación del sistema, …
- Se gana o se compra
  - El proceso gana o pierde prioridad dependiendo de su comportamiento
  - El proceso adquiere un cierto nivel de prioridad pagando por ello al propietario de la máquina

Tipos de Sistema Operativo

- SO por Lotes
  - Se agrupan programas,datos y órdenes de control para formar "trabajos" a ejecutar por el ordenador
    - No permiten interacción hombre - máquina
    - Buscan la máxima eficiencia del sistema
- SO de Tiempo Compartido
  - Pensados para entornos muy interactivos
  - El sistema pretende dar la imagen a cada usuario de que tiene toda la máquina para él
- SO de Tiempo Real
  - Entornos en los que han de ser aceptados y procesados un gran número de sucesos externos con mayor o menor urgencia
    - Control industrial, comunicaciones, control de vuelo, …
- SO de Plazo Fijo
  - Ciertos trabajos tienen un tiempo específico para ser terminados. Se han de terminar antes de ese momento

Tipos de Planificador

- Largo plazo
  - Decide sobre la creación de nuevos procesos, determina qué programas son admitidos por el sistema para ser procesados
  - Controla el grado de multiprogramación
  - Mayor número de procesos en el sistema implica menor porcentaje de tiempo dedicado a cada proceso
- Medio plazo
  - Swapping: intercambio de procesos entre memoria y disco
- Corto plazo
  - Se invoca cada vez que ocurre un evento
    - Interrupción de reloj, Interrupción de E/S, Llamada al sistema operativo, señales
  - Decide qué proceso de los preparados será el siguiente en ejecutarse

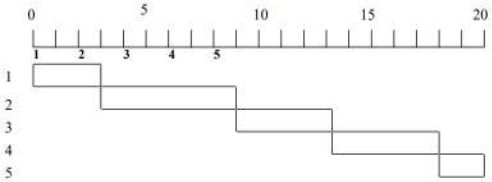
- No preemption
  - Cuando un proceso consigue el procesador, no lo abandona hasta que termina o se bloquea en espera de una operación de E/S
- Preemption
  - Un proceso en estado de ejecución puede ser interrumpido por el Sistema Operativo, pasando a estado preparado
  - Evita que un proceso pueda monopolizar el procesador durante demasiado tiempo

Ejemplo

Proceso	Tiempo de llegada	Tiempo de servicio
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

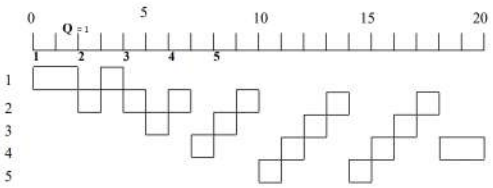
First-Come-First-Served (FCFS)

-Proceso corto puede ser obligado a esperar mucho
-Miniminimiza el tiempo de cambios de contexto



Round-Robin

- Mayor quantum, mejor throughput(Terminación de proceso
- Menor quantum, mejor tiempo de respuesta



- Usa preemption basada en un reloj
- Quantum (Q): tiempo máximo de permanencia en ejecución que se da a un proceso

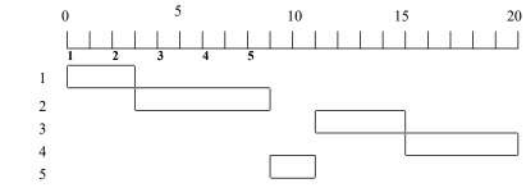
Porcentaje de tiempo perdido en Cambio de Contexto

$$\%Tcc = 100 * \frac{Tcc}{Tcc + Q}$$ (Mínimo)

Tcc = Tiempo de cambio de contexto
Q = Tiempo de quantum asignado

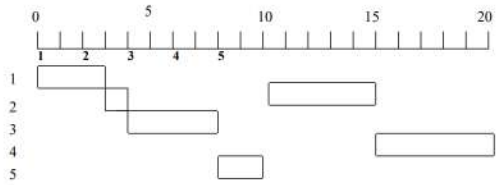
Shortest Process Next

-Procesos largos pueden sufrir inanición



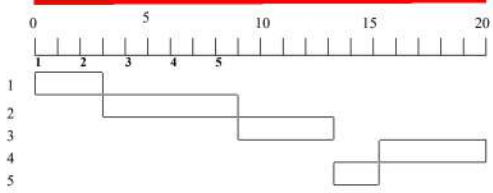
- Política no preemption
- Se selecciona al proceso con menor tiempo en ejecución previsto
- Los procesos cortos se adelantan a los largos

Shortest Remaining Time



- Versión preemptive de la política shortest process next
- Debe estimar el tiempo de proceso
- Posible inanición de procesos largos

Highest Response Ratio Next (HRRN)



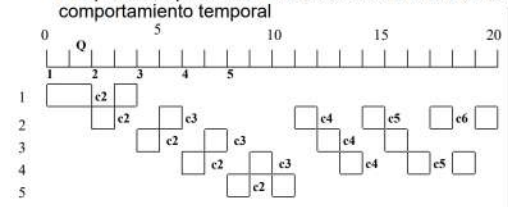
- Elige el siguiente proceso según mayor ratio

$$\frac{\text{Tiempo de espera en las colas + tiempo de servicio}}{\text{tiempo de servicio}}$$

- No preentive
- Resuelve problemas inanición

Colas con realimentación

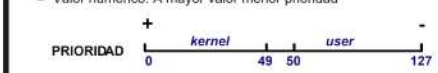
- Varias colas con distinta prioridad
  - Los procesos pasan de unas a otras en función de su comportamiento temporal



Penaliza a los procesos que más tiempo llevan en ejecución

Planificación en UNIX

- Prioridad variable + Round Robin
- Cálculo de prioridad
  - Valor numérico: A mayor valor menor prioridad



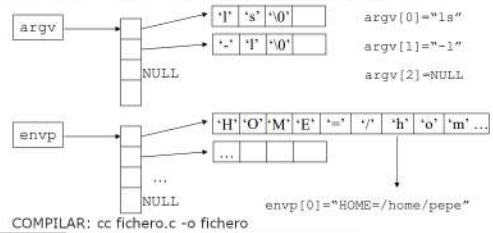
  - Prioridad base + NICE + prioridad variable
  - *Prioridad base* de usuario: 50
  - *NICE*: valor positivo que el usuario puede añadir a sus programas para "molestar menos"
  - *Prioridad variable*: función de la carga del sistema y del uso de CPU que el proceso ha hecho recientemente

Estructura de Programa en C

```
main(int argc, char *argv[], char *envp[])
```

- Lista de argumentos: argv
- Lista de variables de entorno: envp

En el caso de ejecutar el comando ls -l, el valor de argc será 2



COMPILAR: cc fichero.c -o fichero

Información de hijo a padre

- Caso más frecuente
  - Proceso hijo termina y padre está esperando en wait()
    - Padre recibe información sobre la causa de la muerte del hijo continúa ejecutando
    - Hijo desaparece
- Casos especiales
  - Proceso termina y padre no está en wait()
    - Proceso Zombie hasta que padre ejecute wait() o termine
  - Padre termina dejando hijos activos
    - Hijos adoptados por init (pid=1)
  - Padre ejecuta wait() y no existen hijos
    - Wait devuelve -1 y errno=10 (ECHILD: "No children")

Resumen de Identificadores y Marcas (1)

Para cada usuario
- UID (user id.): id. de usuario (nº natural)
  - UID=0 -> superusuario (root)
- GID (group id.): id. de grupo al que pertenece (nº natural)

Para cada fichero
- Número de I-node: identificador numérico
- UID del propietario
- GID del propietario
- Permisos de acceso (user-group-other)
- Bits set-user-id, set-group-id: Permiten cambiar el usuario/grupo efectivo de un fichero.
- Sticky bit: Permite gestión adecuada de ficheros de distintos usuarios en directorios compartidos (por ejemplo /tmp)

PIDs: 0 -> scheduler, 1-> init

- Para cada proceso
  - PID: identificador de proceso - getpid()
  - Process Group ID: Id. de grupo de procesos getpgrp()
    - setpgrp() PGID=PID (proceso líder)
    - setpgid(pid, pgrp)
  - Real User ID: UID del usuario que ha ejecutado el proceso getuid()
  - Real Group ID: GID del mismo usuario getgid()
  - Effective User ID: al ejecutar un programa, SI (set-user-id==1 en el fichero ejecutable) EUID = UID del propietario del fichero (Ej: passwd) SINO EUID = RUID. geteuid()
  - Effective Group ID: igual que EUID aplicado a grupo getegid()

SEÑALES:

- Comunican eventos especiales a procesos en ejecución
  - Son interrupciones software
  - No contienen información. El proceso señalado no conoce la identidad del proceso señalador
  - No es la forma habitual de comunicar procesos. Pueden servir para sincronizar
- Problemas que dan las señales no seguras:
  1. Al capturar una señal el comportamiento de la misma cambia a SIG_DFL
  2. Posibles bloqueos por llegadas de señales en momentos no adecuados (por ej. antes de un pause)
  3. Interrupción de SCs bloqueantes

Tipos de Dispositivos

- De bloque. *Discos*
  - direccionable (bloque), con operaciones de localización
- De caracteres. *Terminales*, *impresoras*, *ratón*…
  - no direccionables, sin operaciones de localización

Directorio:

- Un nivel
  - Todos los ficheros en un único directorio
- Dos niveles
  - Un directorio por usuario
- Jerarquía
  - Cada usuario puede organizar sus ficheros en subdirectorios

Fichero: operaciones

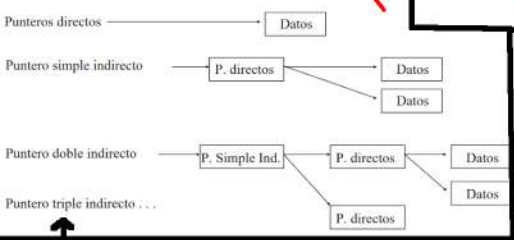
Un fichero es un TAD.

- Operaciones vs. Tipo de acceso
  - Secuencial:
    - rebobinar, leer_siguiente, escribir_siguiente
  - Directo:
    - leer(n), escribir(n), posicionar(n)
    - leer_siguiente, escribir_siguiente
  - Indexado, Hash
    - Leer(clave), escribir(clave), posicionar(clave)
    - leer_siguiente, escribir_siguiente

UNIX: I-node

Estructura de datos que almacena toda la información relativa a un fichero

- Número de I-node
- Tipo fichero
  - Tipos de ficheros
    - Ordinario: definido por el usuario
    - Directorio: contiene una lista de parejas nombres de fichero – Inodo
    - Dispositivo: usado para el acceso a dispositivos de entrada/salida
    - FIFO, pipes o "named pipes": Ficheros para comunicación entre procesos
- Protección
- UID, GID
- Tamaño
- Fecha último cambio
- Número de links
- Información sobre localización de los bloques del fichero
  - Asignación indexada con varios niveles



Asignación contigua

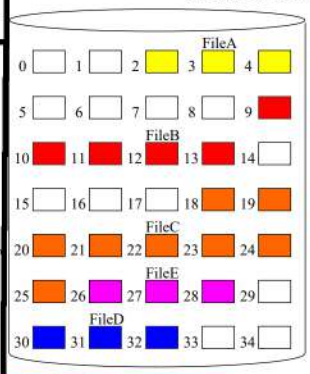


Tabla de localización

File Name	Start Block	Length
FileA	2	3
FileB	9	5
FileC	18	8
FileD	30	3
FileE	26	3

- Fácil localización: inicio y tamaño
- Fácil acceso secuencial y directo
- Difícil asignación
- Pérdida de espacio
- Problemas con cambios de tamaño

Asignación encadenada

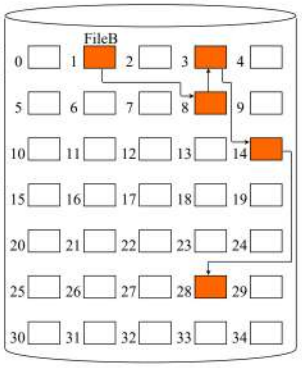


Tabla de localización

File Name	Start Block	Length
…	…	…
FileB	1	5
…	…	…

- Localización: inicio y tamaño
- Acceso secuencial
- Imposible acceso directo
- Fácil asignación
- Sin pérdida de espacio

Asignación indexada

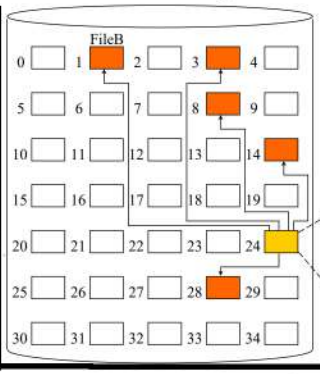


Tabla de localización

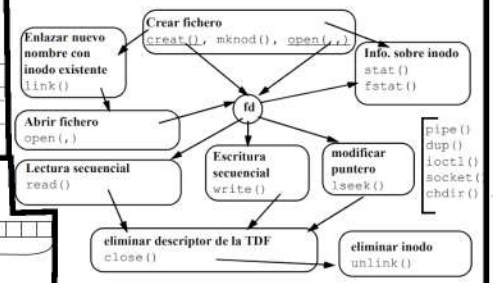
File Name	Index Block
…	…
FileB	24
…	…

1
8
3
14
28

- Localización: índice
- Acceso secuencial
- Acceso directo
- Fácil asignación
- Sin pérdida de espacio

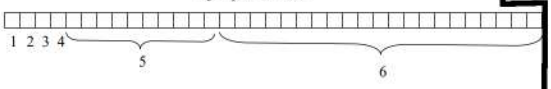
Problema: tamaño del índice

Esquema de llamadas sobre ficheros



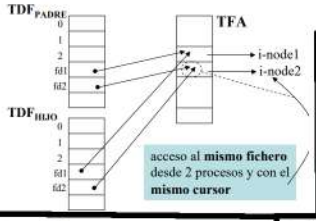
UNIX: Estructura de un disco

Ejemplo: MINIX



- 1. Bloque de arranque
- 2. Superbloque (Cómo está distribuida la información)
- 3. Mapa de bits de nodos-I (Bits para saber si estructura de I-nodos se utiliza o no)
- 4. Mapa de bits de bloques (o zonas) (Bits para indicar qué bloques de datos tienen datos o no)
- 5. Nodos-I
- 6. Bloques de datos

- Tras hacer un **fork( )** el proceso HIJO recibe una copia de la Tabla de Descriptores de Fichero del proceso PADRE:
  => - la TDF se *duplica* entera
  - TODAS las entradas de la TDF_{HIJO} apuntan a la misma entrada en TFA que la correspondiente entrada de la TDF_{PADRE}

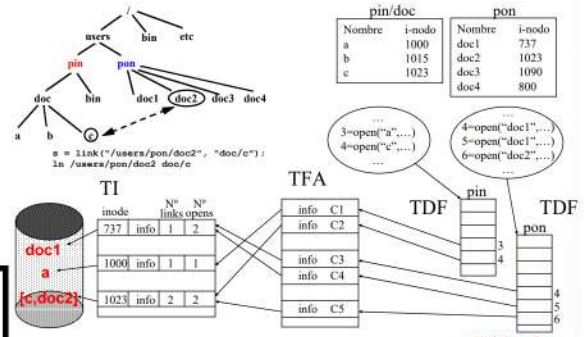


- Tras hacer un **exec( )** el proceso *ejecutado* mantiene la TDF del proceso *ejecutor*

Al hacer "int close(int fd)":
(fd = descriptor de fichero)

queda libre la entrada fd de TDF
decrementa nfd en TFA
si (nfd == 0)
  queda libre entrada en TFA
  decrementa nopens en TI
  si (nopens == 0)
    si (nlinks == 0) borra fichero;
    elimina inodo de TI;

Tablas en memoria del Sistema de Ficheros



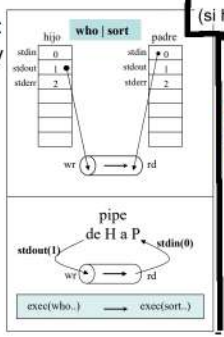
- Por defecto
  - stdin y stdout son:
    - Line buffered si están dirigidas a una terminal o
    - Fully buffered en caso contrario
  - stderr es unbuffered

EL PROBLEMA DE LOS FINES DE LÍNEA SE DA TANTO EN fprintf() COMO EN printf()

Redirección

- Desde el intérprete de comandos:
  - Redirección de la salida:
    who **>** quien_hay.tmp
  - Redirección de la entrada:
    sort **<** quien_hay.tmp
  - Redirección de la salida de errores:
    cc –o ej10 ej10.c **2>** error_10
  - Redirección de la entrada y la salida:
    sort **<** quien_hay.tmp **>** quien_hay

- Qué ocurre con los bufferes cuando:
  - **fork**: se copia el $Buffer_{PADRE}$ al $Buffer_{HIJO}$
    (si había cosas pendientes de escribir, se escriben 2 veces)
  - **exec**: desaparece el Buffer
    (si había cosas pendientes de escribir, se pierden)
  - **exit**: se vacía el Buffer (equivalente a fflush)
  - **terminación involuntaria**:
    desaparece el Buffer
    (si había cosas pendientes de escribir, se pierden)

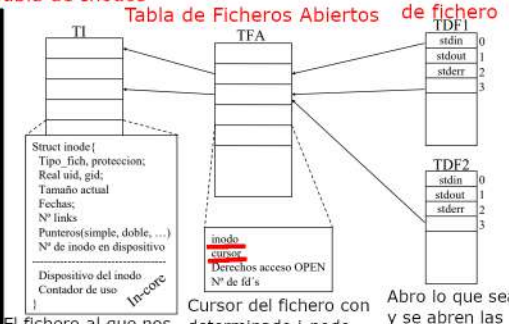


Tuberías

DIFERENCIAS

UNAMED	NAMED
comunican procesos emparentados	comunican cualquier proceso
No están representados en el Sistema de Ficheros mediante un nombre	Tienen nombre en el Sistema de Ficheros, como un fichero cualquiera
Se crean con **pipe( )**	Se crean con **mknod( )**
Muy utilizados, especialmente en el *shell*: who | sort | lp -oq	Muy poco utilizados

Tablas de Inodos / Tabla de Ficheros Abiertos / Tabla de Descriptores de fichero

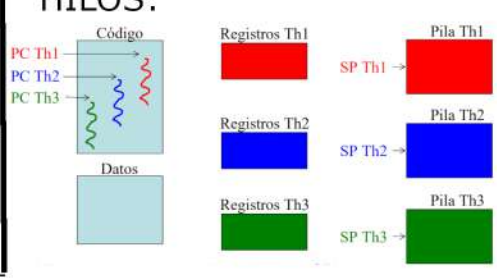


El fichero al que nos referimos (i-nodo)

Cursor del fichero con determinado i-nodo

Abro lo que sea y se abren las entradas y salidas standard

HILOS:



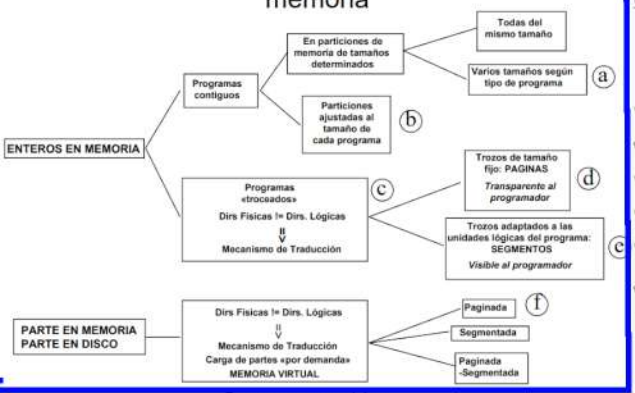
Si un hilo se bloquea, no bloquea a todo el proceso. El kernel puede poner en ejecución otros hilos del mismo proceso

En un sistema multiprocesador, el kernel puede poner simultáneamente en ejecución varios hilos del mismo proceso

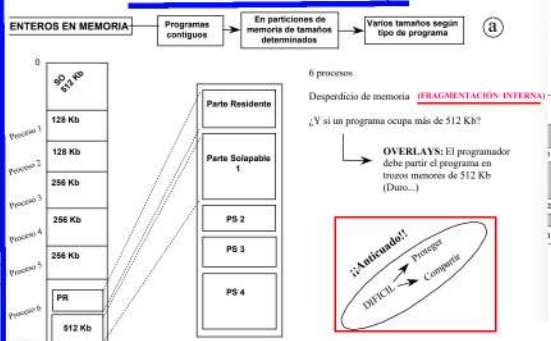
Tienen mayor coste en cambio de contexto de hilo

Tienen mayor coste para crear, señalizar/esperar, …

Cómo pueden residir los programas en memoria

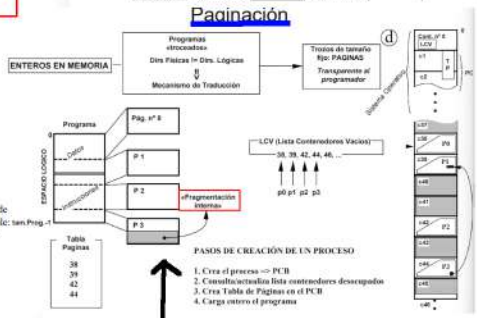


Particiones de tamaño fijo

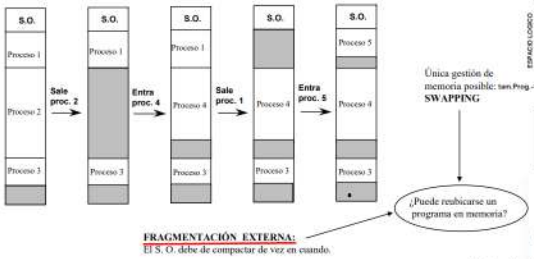


Programas troceados

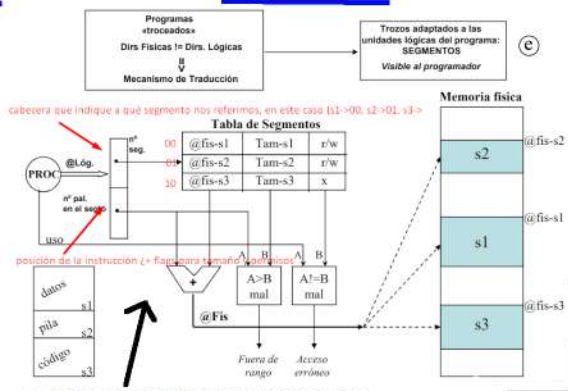
Paginación



Particiones de tamaño variable

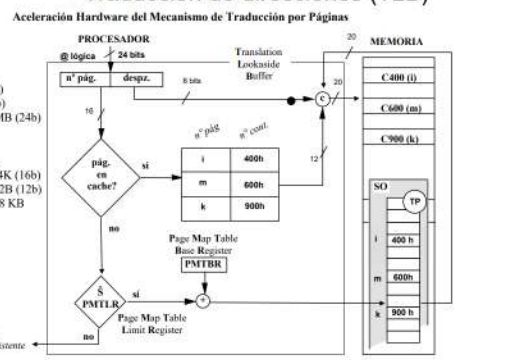


Segmentación



- Segmentos definidos por el programador
- Fragmentación Externa
- Crecimiento dinámico fácil (pila, heap)
- Cada segmento puede estar en un fichero distinto
- El espacio lógico está formado por DOS dimensiones, el programador «ve» las dos dimensiones:
  - nº de segmento.
  - nº de palabra dentro del segmento.
- Coste temporal:
  - 2 accesos a tabla (< l/e, tam_s1> ;<@Fís_s1>)
  - 1 SUMA (Inicios de Segmento NO ALINEADOS)
  - 1 acceso a @ Física

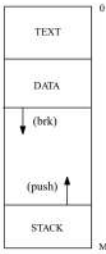
Traducción de direcciones (TLB)



Gestión de Memoria en UNIX

- Sistema basados en SWAP
- Sistema basados en Demand Paging
- Gestión de memoria en Regiones

Región:
- Area contigua del espacio virtual de un proceso
- Tratada como un único objeto
- Uniforme en cuanto a permisos, shared...



- Gestión de memoria en Tablas