1. Sistema Operativo Unix

1.1 Introducción al S.O. Unix y su entorno

1.2 Subsistema de Archivos

1.3 Subsistema de Procesos

1.4 Políticas de Gestión de Memoria

1.3 Subsistema de procesos: Introducción

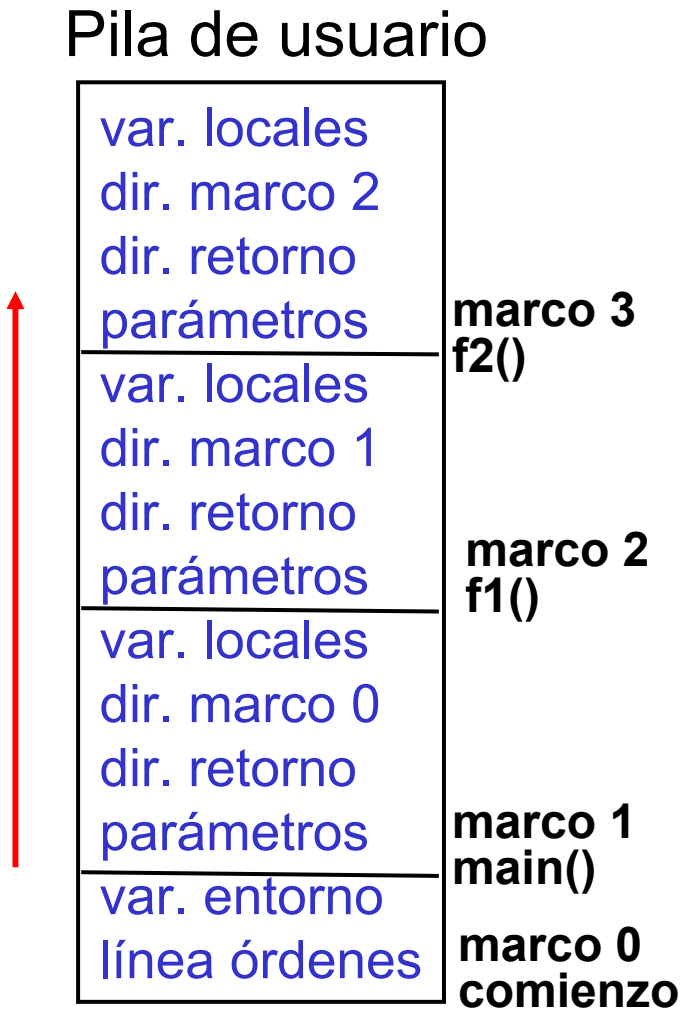
- **Programa** ➔ archivo ejecutable
- **Proceso** ➔ instancia de un programa en ejecución
- El núcleo identifica a los procesos por su **PID**
- En Unix, proceso es la entidad que se crea con la llamada al sistema ***fork*** (excepto el proceso 0)
- Procesos especiales que existen durante la vida del sistema:
  - **Proceso 0**: creado "a mano" cuando arranca el sistema. Crea al proceso 1 y se convierte en el proceso **intercambiador**
  - **Proceso 1 (Init)**: antecesor de cualquier proceso del sistema

Introducción al subsistema de procesos (y II)

- **Partes de un programa ejecutable**
  - conjunto de cabeceras
  - texto del programa
  - datos inicializados y no inicializados (*bss*)
  - otras secciones (ej. tabla de símbolos)
- El núcleo divide el espacio de un proceso en **regiones** (o segmentos)
  - región de texto: sólo lectura
  - región de datos
  - región de pila

Introducción al subsistema de procesos (y III)

- La pila de usuario se crea automáticamente y su tamaño se ajusta dinámicamente
- Consta de **marcos de pila** lógicos que se insertan cuando se llama a una función y se eliminan cuando finaliza
- Como un proceso puede ejecutarse en modo supervisor y modo usuario, existe **una pila para cada modo**



Estructuras de datos para los procesos

- Cada proceso tiene una **entrada en la tabla de procesos** y una zona privada llamada **u-área**: ambas contienen información de control y estado sobre el proceso
- Cada proceso tiene una **tabla de regiones privada** (*pregión*) cuyas entradas apuntan a entradas de una **tabla de regiones global**
- El núcleo tiene una variable global donde almacena la dirección de la u-área del proceso en ejecución

Estructuras de datos para los procesos (y II)

- La tabla de regiones global permite la **compartición** de regiones entre procesos

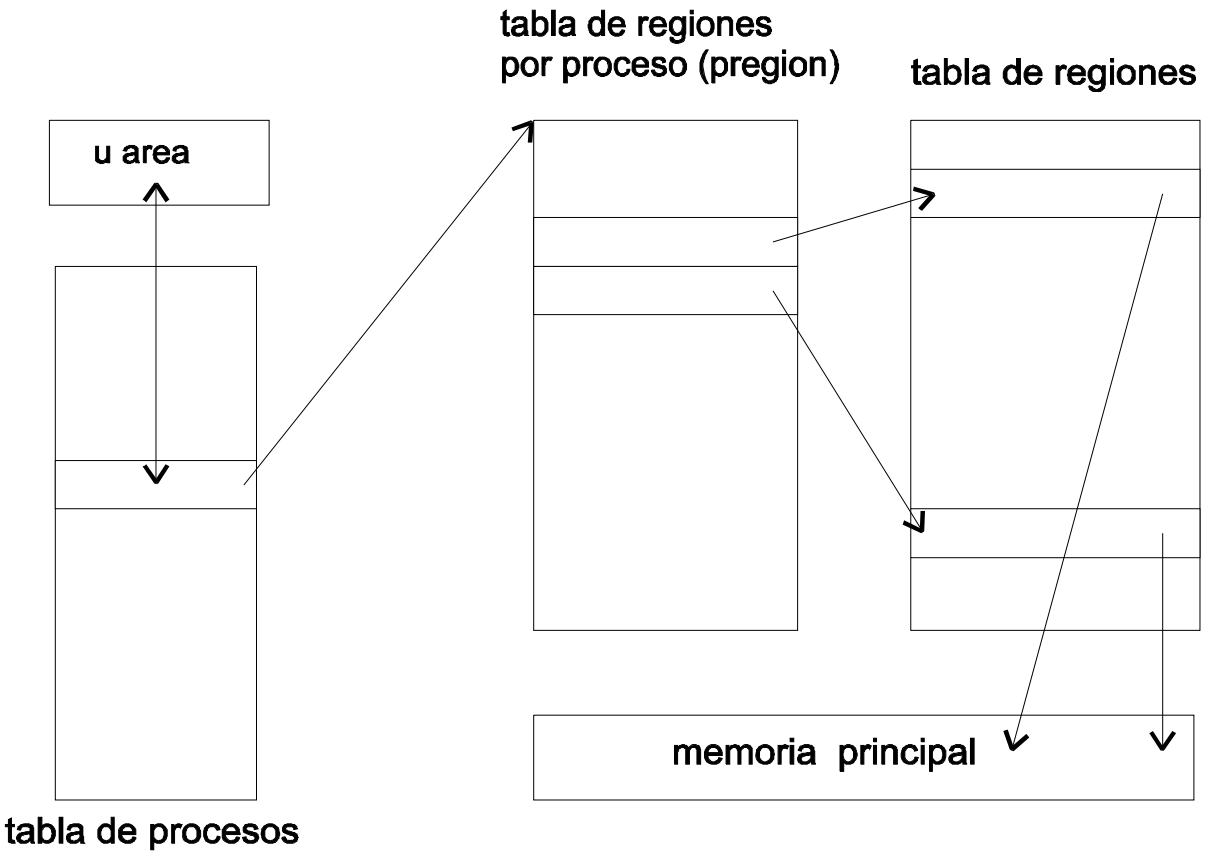


Tabla de procesos

- **Estado** del proceso
- Campos para localizar el proceso y su u-área, e información sobre el tamaño del proceso
- Algunos UID's que determinan distintos privilegios del proceso
- PID's que especifican las relaciones entre procesos
- Descriptor de eventos que espera un proceso
- Parámetros de planificación

Tabla de procesos (y II)

- Señales recibidas pero no tratadas aún
- Temporizadores indicando el tº de ejecución del proceso y el tº de uso de los recursos
- Punteros para enlazar el proceso en la cola del planificador, o si está bloqueado, en la cola de bloqueados
- Punteros para las colas hash en base a su PID

U-área

- Puntero a la entrada de la tabla de procesos
- UIDs y GIDs real y efectivo
- Tiempos que los procesos y sus descendientes gastan ejecutándose en modo usuario y modo supervisor
- Matriz que almacena la reacción ante las señales
- Terminal asociado al proceso, si existe
- Campos de error y del valor devuelto (para las llamadas al sistema)

U-área (y II)

- Tabla de descriptores de archivos
- Parámetros de las operaciones de E/S
- Directorio actual y raíz
- Tamaños límites de procesos y archivos
- Campo modo de permisos
- Pila del núcleo

Tabla de regiones por proceso

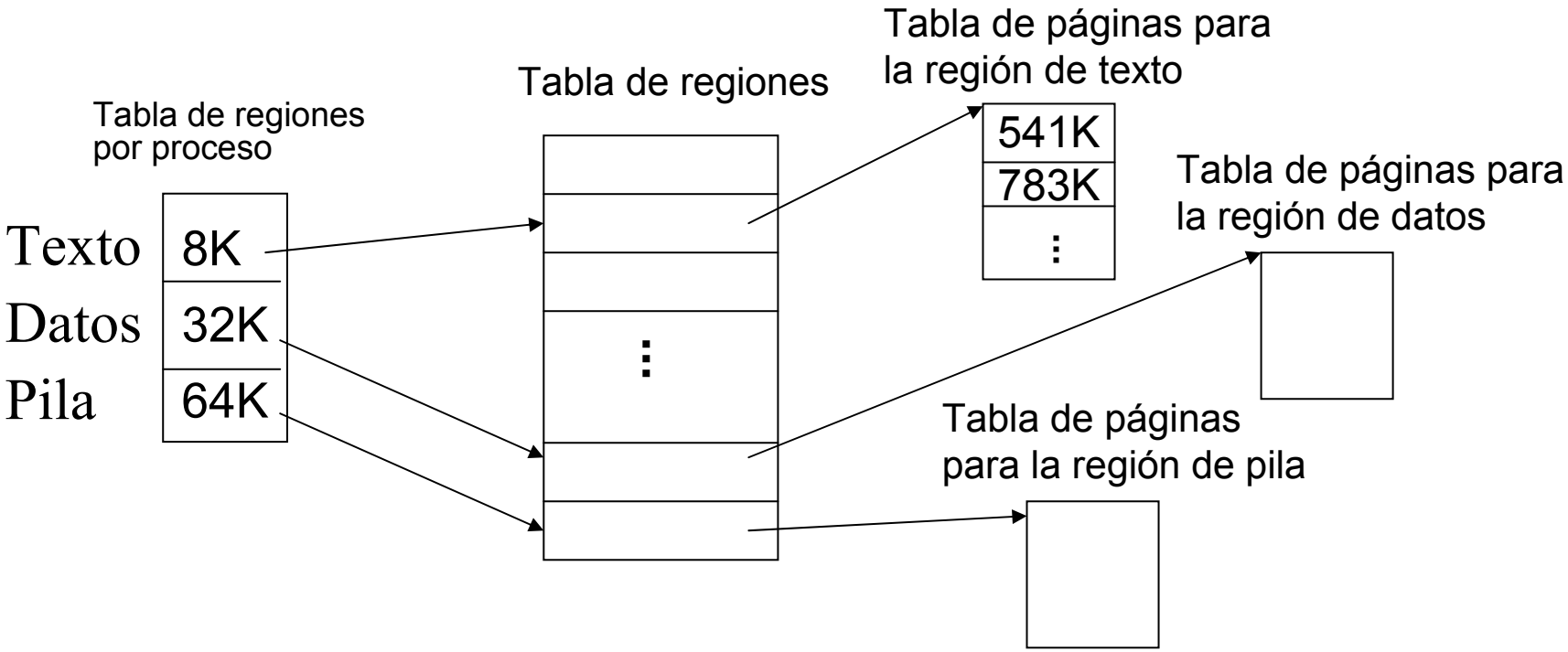
- Puede estar en distintos lugares, depende de la implementación:
  - en la tabla de procesos
  - en el u-área
  - en una zona de memoria asignada separadamente
- Contenido de cada entrada
  - puntero a la correspondiente entrada de la tabla de regiones
  - dirección virtual de comienzo de la región
  - campos de permiso para el tipo de acceso del proceso
- El concepto de región es independiente de las políticas de gestión de memoria utilizadas

Tabla de regiones

- Puntero al **i-nodo del archivo ejecutable** desde donde se cargaron originalmente sus contenidos
- **Tipo de región**: texto, memoria compartida, pila, ...
- **Tamaño** de la región
- **Localización** de la región en memoria principal
- **Estado** de la región (bloqueada, en demanda, cargándose en memoria o válida)
- **Contador** de procesos que referencian la región

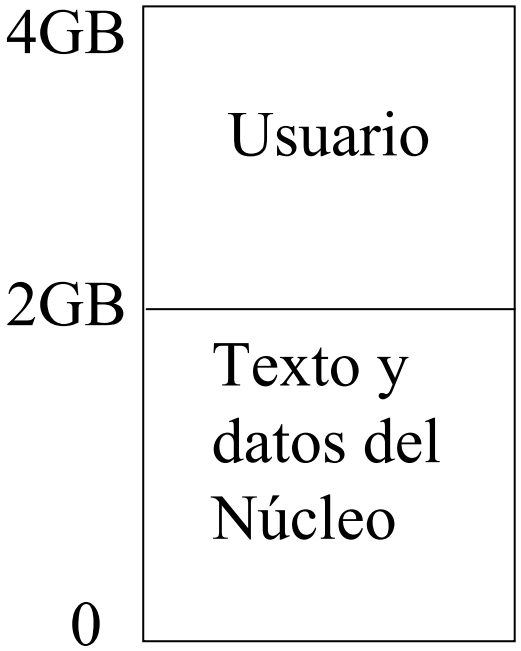
Diseño de Memoria del Sistema

- Se utiliza un sistema de paginación por demanda: cada región tiene asociada una tabla de paginas con información de las páginas que la forman



Diseño de Memoria del Sistema (y II)

- La estructura del espacio de memoria virtual de un proceso se divide en dos partes:

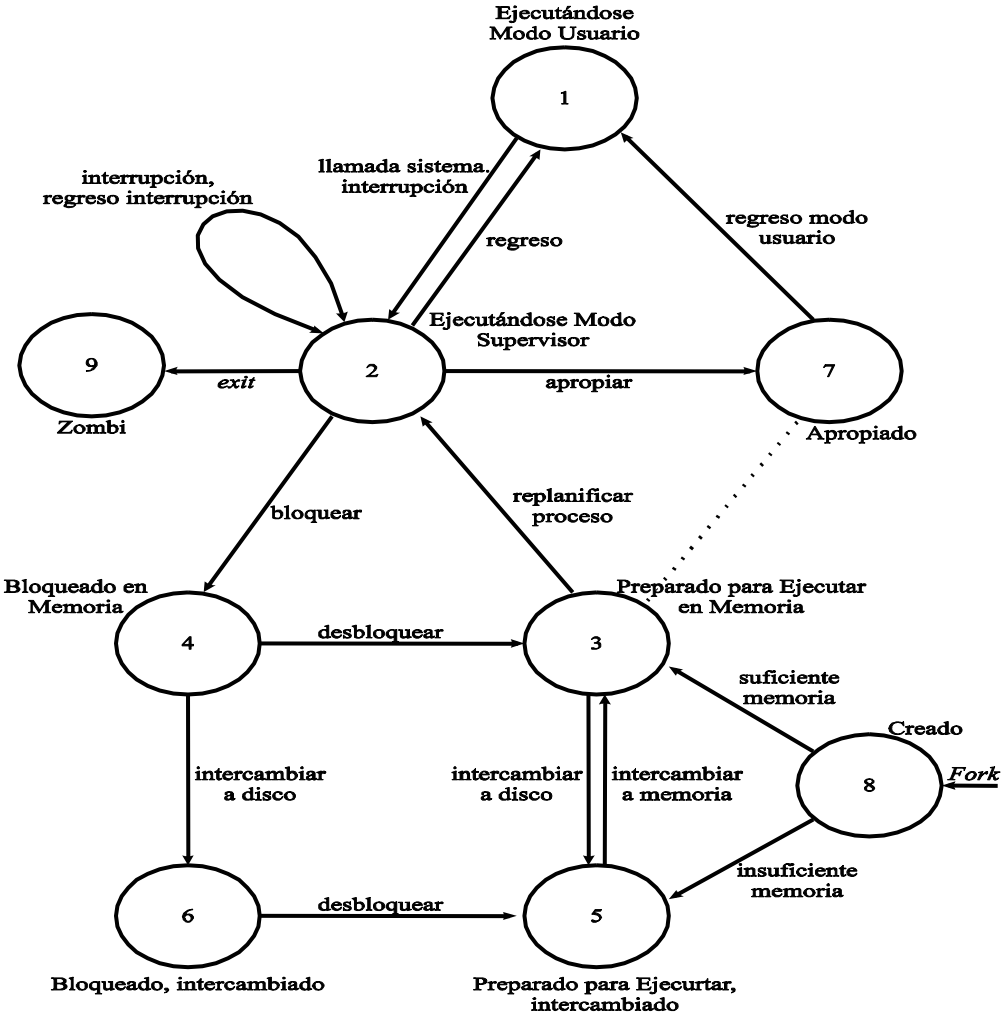


- Con direcciones de 32 bits, el espacio virtual máximo para un proceso es de 4GB

- Los primeros 2GB se ocupan con el código y las estructuras de datos del núcleo

Estados de los procesos y transiciones

- El proceso tiene control sobre algunas transiciones
- Ningún proceso puede desplazar a otro que se está ejecutando en modo supervisor
- El 3 y el 7 son realmente el mismo estado



Consistencia de las estructuras del núcleo

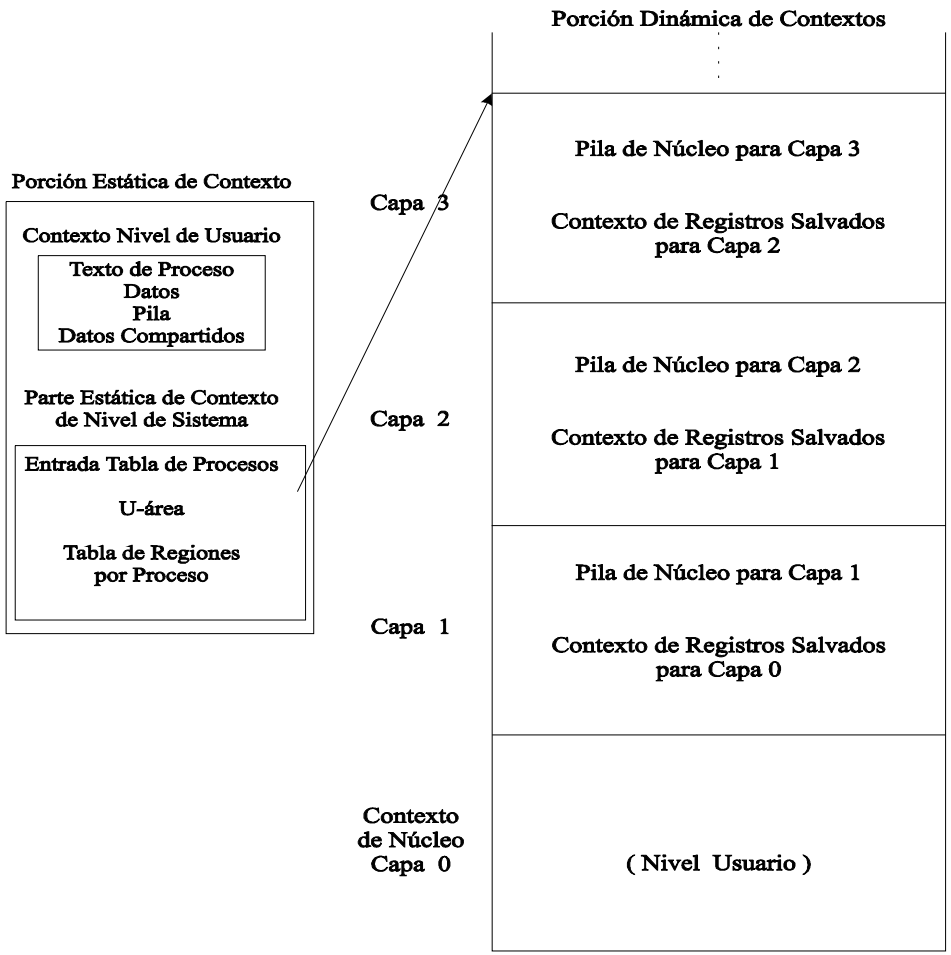
- Es un problema de **Exclusión Mutua**
- En Unix tradicional
  - **enmascaramiento** de interrupciones
  - núcleo **no apropiativo**: los procesos que se ejecutan en modo supervisor no pueden ser apropiados
  - **bloquear** a los procesos cuando una estructura del núcleo está en uso por otro proceso (sleep/wakeup)
- Los núcleos actuales son apropiativos (total, Solaris o parcialmente, SVR4)

Contexto de un proceso

- Es el **estado** del proceso, la unión de:
  1. Contexto a nivel de **usuario** (texto, datos, pila)
  2. Contexto a nivel de **registros**
     - *PC*, dirección de la siguiente instrucción
     - Registro de estado del procesador (estado hardware)
     - *SP*, puntero de pila
     - Registros de propósito general
  3. Contexto a nivel de **sistema**
     - Parte estática, fija (entrada TP, u-área, pregion)
     - Parte dinámica, variable (pila de capas de contexto)

Contexto de un proceso (y II)

- Se introduce una capa:
  - ocurre una interrupción
  - llamada al sistema
  - cambio de contexto
- Se saca una capa:
  - regreso de interrupción
  - regreso a modo usuario
  - cambio de contexto
- Un proceso se ejecuta dentro de su capa de contexto actual



Control de procesos

- Llamadas al sistema para trabajar con los procesos

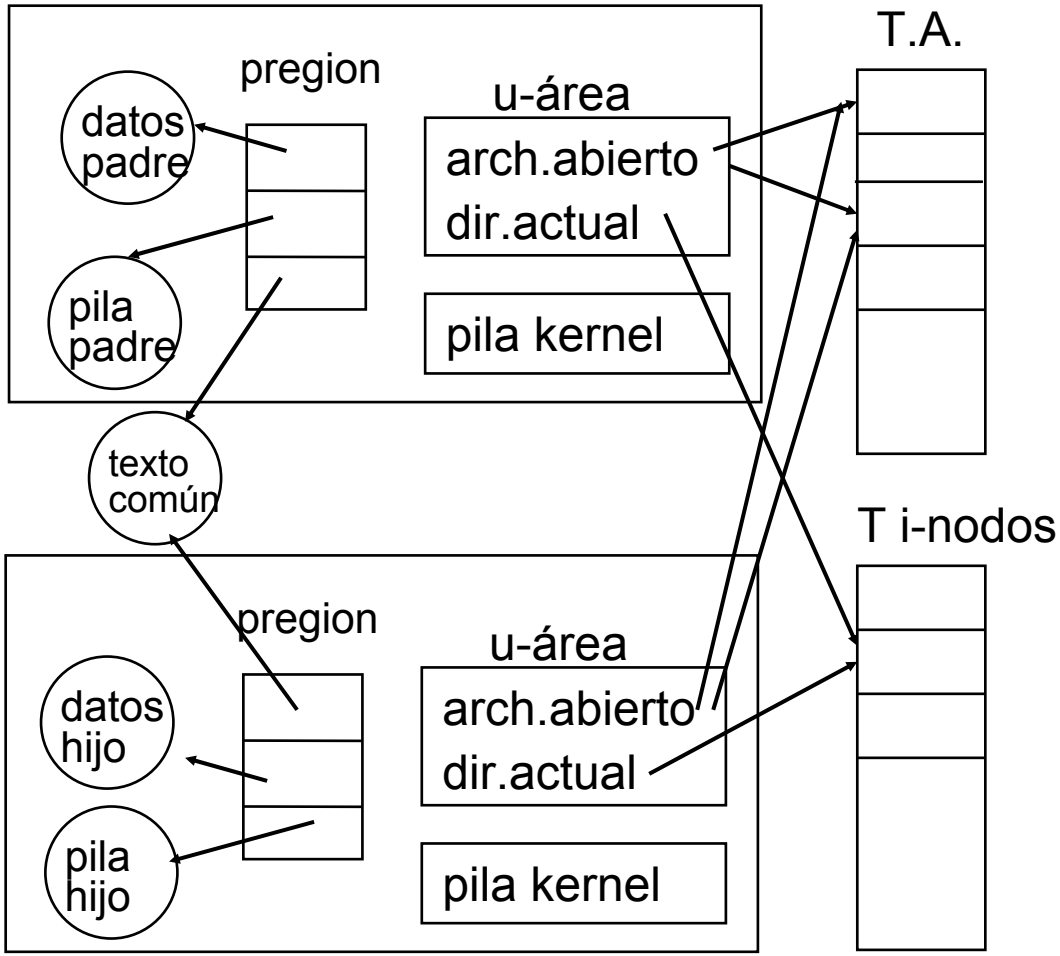
Relacionadas con la Gestión de Memoria			Relacionadas con la sincronización			
fork	exec	brk	exit	wait	signal	kill
algoritmos internos (sleep, wakeup, ...)						

Llamada al sistema fork

- Crea un nuevo proceso, copia casi idéntica del padre
- Devuelve al padre el PID del hijo y al hijo 0
- Sintaxis:

```
pid = fork()
```

- Después del fork, ambos se ejecutan concurrentemente



Llamada al sistema exec

- Ejecuta el programa que se le pasa como argumento
- Existen seis funciones exec que se diferencian únicamente en la forma de pasar los argumentos (execl, execv, execvp, ...)
- Si la llamada tiene éxito, el espacio de direcciones del proceso que la inició se ha sustituido completamente por un nuevo programa
- Hay que asegurar que el nuevo programa se ejecutará correctamente. Ej. puede ser necesario redirigir la entrada o la salida o ambas

Llamadas al sistema exit y wait

- **exit**
  - Pone fin a la ejecución de un proceso
  - Devuelve el estado de finalización al padre y el proceso pasa a estado *zombie*
- **wait**
  - Espera de la teminación del primer hijo
  - Si no ha terminado ningún hijo, el proceso (padre) se bloquea
- Si un padre finaliza antes que sus hijos, los hijos se conectan al proceso *Init*
- Si un hijo finaliza sin que el padre ejecute wait, se queda en estado *zombie*

Señales

- Informan a los procesos de la ocurrencia de eventos (mecanismo de comunicación-sincronización)
- Los procesos y el núcleo pueden mandar señales
- Categorías

  1) Eventos síncronos producidos por errores generados por la ejecución de un proceso. Ej. violación de segmentación (SIGSEGV) o instrucción ilegal (SIGILL)

  2) Eventos asíncronos que ocurren externamente a la ejecución del proceso pero que tienen alguna relación. Ej. terminación de un hijo (SIGCHLD) o bloqueo desde el manejador del terminal tty (SIGHUP)

Señales (y II)

- Cada señal tiene asociado un número entero positivo
- Cuando se envía una señal a un proceso, el núcleo activa un bit del **campo de señales** correspondiente al tipo de señal (entrada tabla de procesos)
- Un proceso puede bloquearse de dos formas:
  - **interrumpible**: el suceso que espera no se sabe cuando ocurrirá o si ocurrirá (ej. pulsación de una tecla). Si un proceso está bloqueado de forma interrumpible, el envío de la señal provoca su desbloqueo
  - **no interrumpible**: el suceso ocurrirá pronto (ej. fin E/S)
- Un proceso puede recordar diferentes tipos de señales, pero no cuántas de cada tipo ha recibido

Señales (y III)

- Hay dos fases en el proceso de señalización:

1) Generación: el núcleo genera señales en base a varios eventos:

- Excepciones
- Un proceso puede enviar una señal a otro u otros
- Interrupciones del terminal
- Control de tareas: la shell usa señales para manipular los procesos en foreground y background
- Quotas: exceso en límites de CPU o tamaños de archivos
- Notificaciones solicitadas por un proceso al sistema
- Alarmas

Señales (y IV)

2) Reparto o manejo: cuando el proceso va a pasar del estado ejecutándose en modo supervisor a ejecutándose en modo usuario se comprueba si tiene señales pendientes. Posibles acciones que realizaría el núcleo:

- Abortar o terminar el proceso
- Ignorar
- Ejecutar una función concreta
- Suspender al proceso
- Reanudar un proceso suspendido

Cada señal tiene una acción por defecto que lleva a cabo el núcleo si no se especifica una alternativa.

- Hay señales especiales: SIGKILL o SIGSTOP

Señales (y V)

- Para especificar la acción deseada

  funcion_vieja = signal (num_señal, funcion)

  funcion puede ser:
  - dirección de la función a invocar
  - ignorar la ocurrencia de la señal
  - terminar el proceso

- Para enviar una señal

  kill (pid, num_señal)

  Dependiendo del valor de pid, el núcleo envía la señal a:
  - pid > 0: al proceso con ese nº de pid
  - pid = 0: a todos los procesos del grupo del emisor
  - pid = -1: a todos los procesos cuyo UID real = UID efectivo del emisor. Si el emisor tiene un UID efectivo de superusuario se manda a todos excepto al proceso 0 y 1

Tipos de procesos

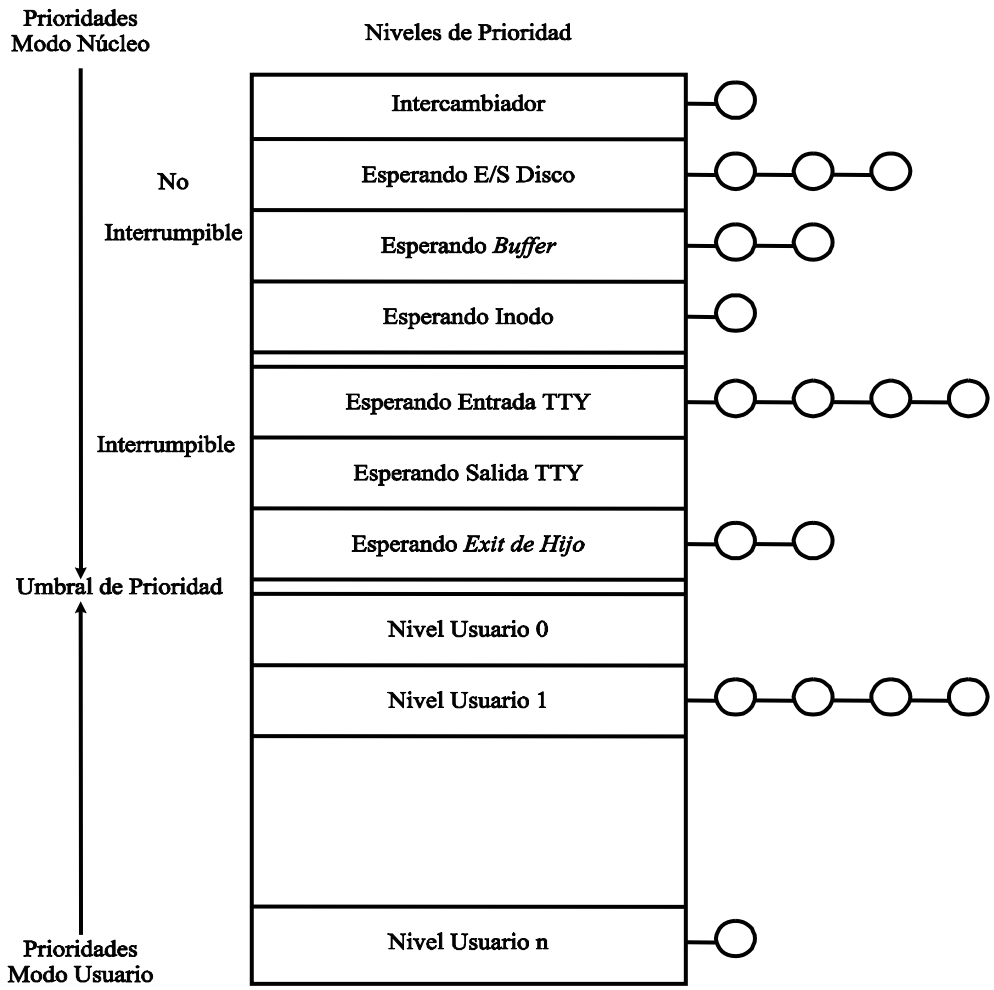
- Procesos de **usuario**
- Procesos **demonios** (*daemons*)
  - No están asociados a ningún terminal
  - Realizan funciones del sistema (spooler de impresión, administración y control de redes, ...)
  - Pueden crearlos el proceso **Init** o los procesos de usuario
  - Se ejecutan en modo usuario
- Procesos del **sistema**
  - Se ejecutan en modo supervisor
  - Los crea el proceso 0
  - Proporcionan servicios generales del sistema
  - No son tan flexibles como los demonios (recompilación)

Planificación de procesos

- Cada proceso activo tiene una prioridad de planificación
- El planificador utiliza la política de **colas múltiples con realimentación**, donde cada cola se gestiona por **Round Robin**
- El **algoritmo** que se sigue es:
  - selecciona el proceso de más alta prioridad de aquellos que están en estado apropiado o preparado para ejecutarse en memoria
  - si existe más de un proceso, elige el más antiguo
  - si no hay procesos, espera a la siguiente interrupción y después de su tratamiento, intenta planificar un proceso

Rango de prioridades de un proceso

- Se divide en dos clases:
  - de usuario
  - del núcleo
- Cada clase tiene distintos valores de prioridad y cada prioridad tiene una cola asociada
- Existe un valor umbral que las separa
- La prioridad de usuario depende de la **cantidad de tº de CPU consumida** y su valor ***nice***



Cálculo de la prioridad de un proceso

- El núcleo calcula la prioridad de un proceso cuando:
  - un proceso se va a bloquear: le asigna un valor de prioridad según la razón del bloqueo
  - un proceso pasa de modo supervisor a modo usuario
  - el manejador del reloj ajusta las prioridades de todos los procesos en modo usuario periódicamente (cada 1sg.) y provoca que el núcleo llame al algoritmo de planificación

Cálculo de la prioridad de un proceso (y II)

- El reloj puede interrumpir a un proceso varias veces durante su quantum
- En cada interrupción, el manejador de reloj incrementa el campo "**uso reciente de CPU**" de la tabla de procesos
- Cada segundo, el manejador de reloj ajusta el uso reciente de CPU de cada proceso preparado:

decay(CPU) = uso reciente CPU/2

y recalcula la prioridad de todo proceso preparado:

prioridad = (uso reciente CPU)/2 + umbral + nice

Ejemplo de planificación

- Un valor numérico bajo implica una alta prioridad
- No se usa el valor nice
- Interrupción de reloj 60 veces por segundo
- Los procesos no hacen llamadas al sistema ni se bloquean
- El proceso A es el primero que se elige
- quantum = 1 sg.
- No existen otros procesos

Tiempo	Proc A		Proc B		Proc C	
	Prioridad	Contador CPU	Prioridad	Contador CPU	Prioridad	Contador CPU
0	60	0	60	0	60	0
		1 ... 60				
1	75	30	60	0	60	0
				1 ... 60		
2	67	15	75	30	60	0
						1 ... 60
3	63	7	67	15	75	30
		8 ... 67				
4	76	33	63	7	67	15
				8 ... 67		
5	68	16	76	33	63	7

Funciones del manejador del reloj

- Toda máquina Unix tiene un reloj hardware que interrumpe al sistema en intervalos de tº fijos (típicamente se establece el tick de reloj en 10ms)
- Las funciones del manejador de la interrupción de reloj son (no todas en cada tick):
  - Reiniciar el reloj (si es necesario)
  - Planificar funciones internas del núcleo basadas en temporizadores
  - Reunir estadísticas del sistema y de los procesos
  - Mantener la hora
  - Enviar las señales de alarma a los procesos
  - Controlar la planificación de procesos
  - Despertar al *intercambiador* y *stealer* periódicamente

Análisis del algoritmo de planificación

- Es simple y efectivo. Adecuado para sistemas generales de tiempo compartido
- **Limitaciones**:
  - No es adecuado cuando crece mucho el número de procesos porque recalcular las prioridades puede ser costoso
  - No hay forma de garantizar un tiempo específico de CPU para un proceso o un conjunto de procesos
  - No garantiza tiempos de respuesta para aplicaciones con características de tiempo real
  - Procesos de alta prioridad pueden tener que esperar por ser el núcleo no apropiativo (inversión de prioridad)

Planificador SVR4

- Se introducen **clases de planificación**: definen la política de planificación para los procesos. Hay dos:

  1) **Tiempo compartido** (0 - 59):

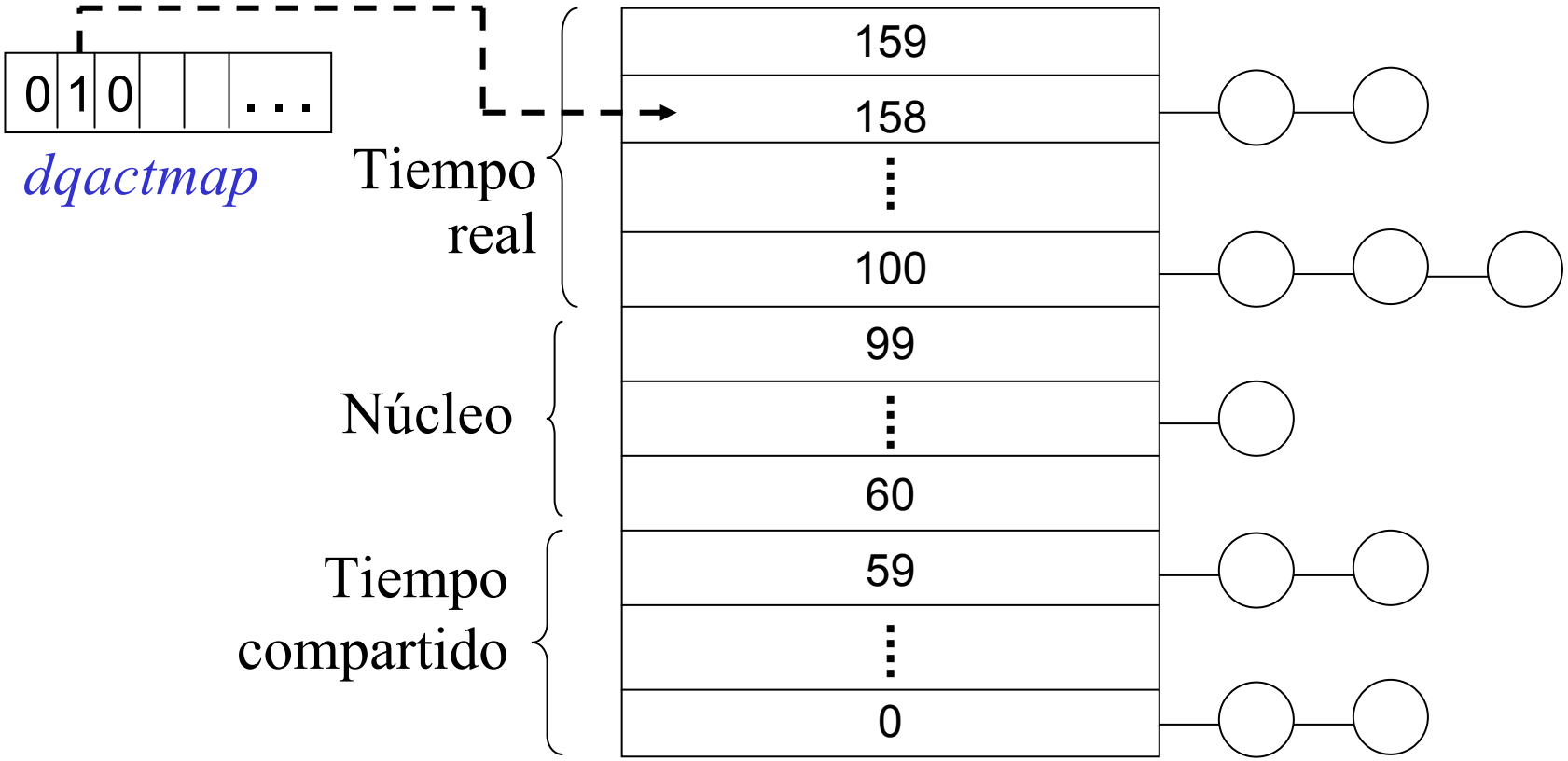
     - Cambia la prioridad de un proceso en respuesta a eventos específicos relacionados con ese proceso
     - Existe una tabla de parámetros del planificador que define cómo los distintos eventos cambian la prioridad de un proceso
     - El quantum depende de la prioridad

Planificador SVR4 (y II)

- Disminuye la prioridad cuando consume quantum y aumenta cuando se bloquea o tarda en usar la CPU
- Cuando un proceso se bloquea, se le asigna una prioridad a nivel de núcleo (60 - 99)
- Valor *nice* entre -20 a +19 (por defecto, 0)

2) **Tiempo real** (100 - 159)

- Los procesos tienen prioridad y quantum fijos especificados explícitamente
- Sólo el superusuario puede tener procesos en esta clase

Planificador SVR4 (y III)

- Existe una interfaz para ciertas rutinas dependiendo de la clase (calculoPrioridad, ...)