Taller de Programación para Sistemas Operativos

Escuela de Ingeniería Civil Informática

- Administración del Procesador

UNIVERSIDAD DEL BÍO-BÍO

INTRODUCCIÓN

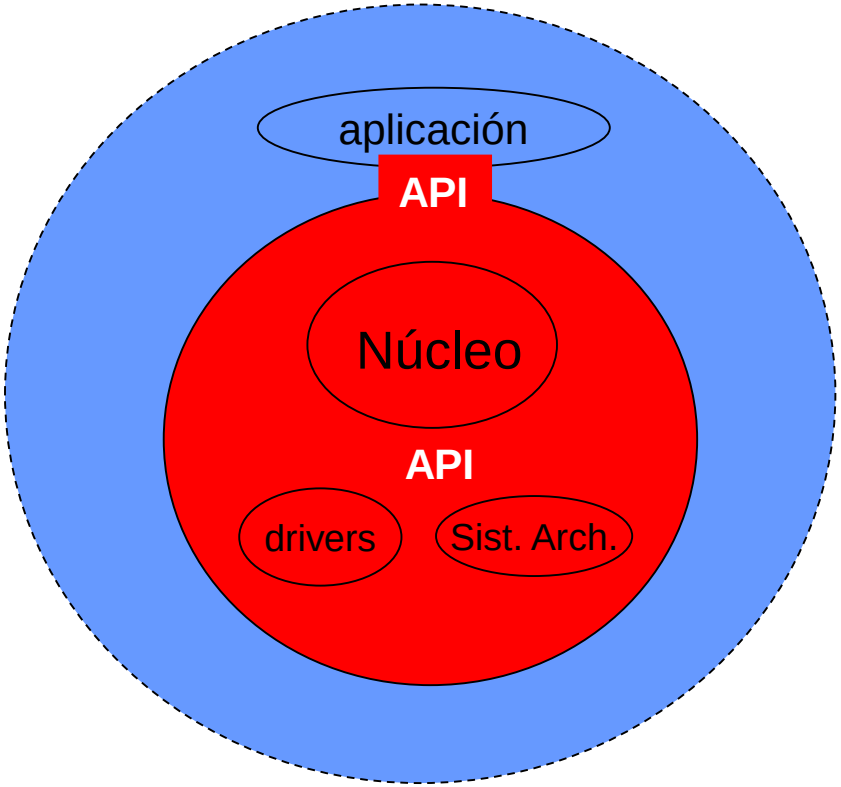
- La actividad más importante del S.O. es implementar los procesos.
- Cada proceso es un procesador virtual en donde se ejecuta una aplicación o herramienta del S.O.
- El kernel debe encargarse de administrar los recursos hardware del computador para que sea asignados convenientemente a los procesos.
- Un **programa** es una **unidad inactiva**, como por ejemplo, un archivo almacenado en disco.
- Un **proceso** (o **tarea**) es **una unidad activa**, que ha sido presentada por el usuario, y requiere un conjunto de recursos para llevar a cabo su función.
- Un proceso representa **una sola instancia** de un programa.

PARALELISMO Vs CONCURRENCIA

- **El paralelismo**
  - Técnica utilizada para hacer que un programa funcione más rápido mediante la realización de ciertos cálculos en paralelo.
  - Utiliza diferentes CPUs físicas para acelerar los cálculos.

- **La concurrencia**
  - Técnica que hace que un programa sea más usable.
  - Permite la multiplexación de una CPU física para que ejecute múltiples procesos simultáneamente.
  - El término “multitarea” es sinónimo de apoyo a la concurrencia.

ARQUITECTURA DEL S.O



fopen, fread, fwrite, malloc, free

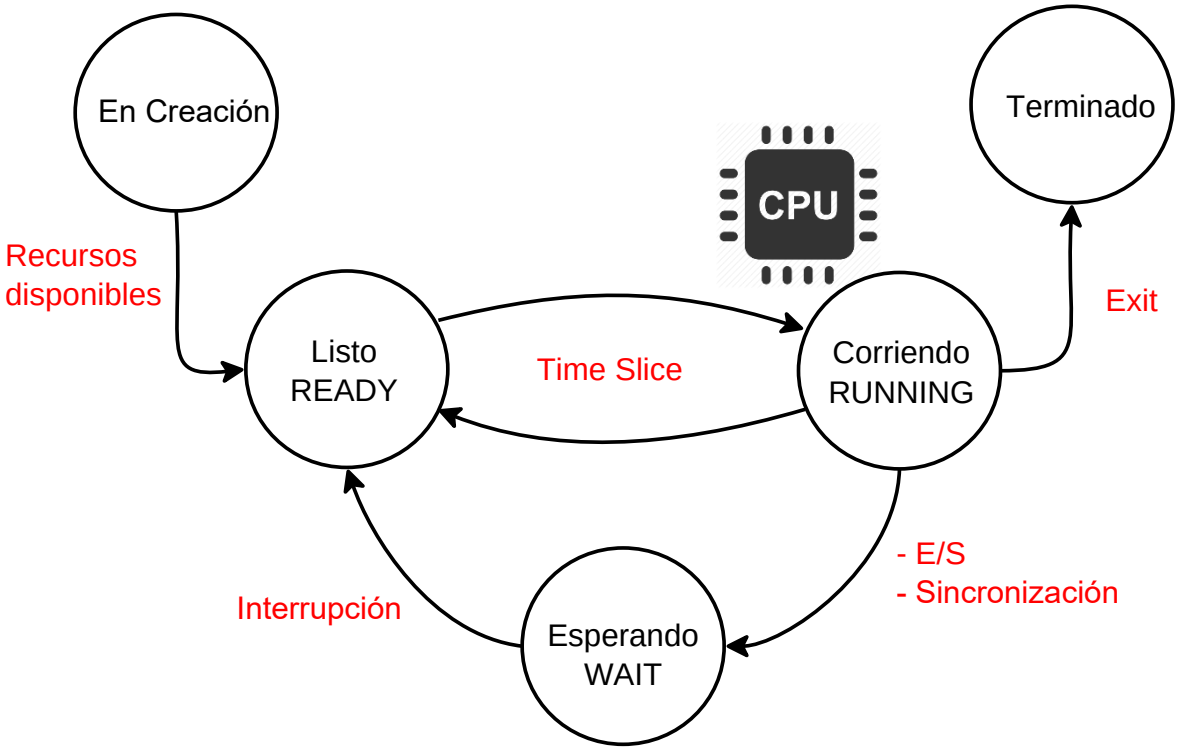
open, read, write, sbrk, fork, exec, exit

Estructuras del núcleo: Descriptores de proceso, archivos abiertos, colas de procesos, recursos pedidos, etc.

API: Application Program Interface

ESTADOS DE UN PROCESO

- Mientras un proceso se ejecuta puede pasar por diferentes estados:



- Un proceso pasa de un estado a otro constantemente y varias veces por segundo.

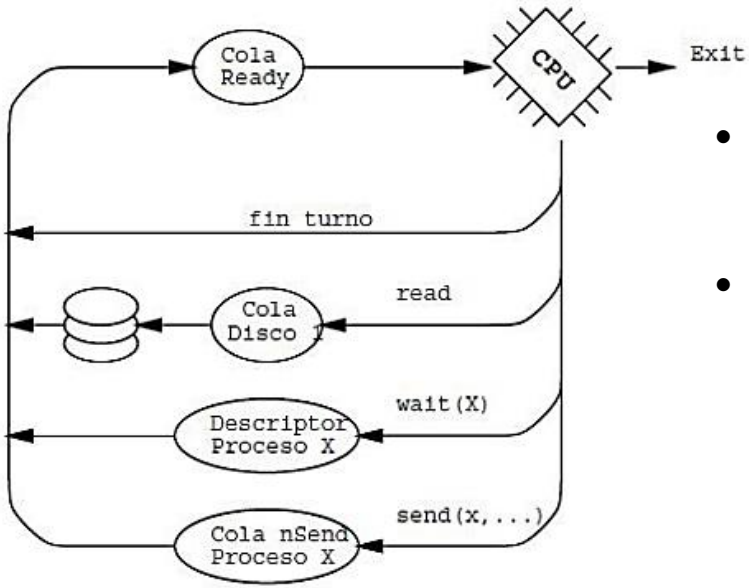
ESTADOS DE UN PROCESO

- **En creación:** El núcleo está obteniendo los recursos que necesita el proceso para correr (memoria o disco)
- **Corriendo:** o RUNNING. El proceso está en posesión del procesador, el que ejecuta sus instrucciones.
- **Esperando:** el proceso espera que se lea un sector del disco, que llegue un mensaje de otro proceso, que transcurra un intervalo de tiempo, que termine otro proceso, etc.
- **Listo:** o READY. El proceso está activo pero no está en posesión del procesador.
- **Terminado:** El proceso terminó su ejecución pero sigue existiendo para que otros procesos puedan determinar que terminó.

COLAS DE SCHEDULING

- Cuando un proceso espera por el procesador, puede permanecer en diferentes colas, las cuales se denominan Colas de Planificación.

- Estas pueden ser FIFO o colas de prioridades



- Se implementa típicamente como una lista enlazada simple.

- Otras colas de espera:
  - Scheduling de disco. E/S.
  - Mensaje síncrono a otro proceso que no está preparado para recibirlo.
  - Reloj regresivo, a la espera de que transcurra un instante de tiempo.

ESTADOS DE UNA TAREA EN NSYSTEM

- NSYSTEM también cuenta con distintos estados definidos para sus tareas, los más importantes son:
  - **READY:** elegible por el planificador o corriendo.
  - **ZOMBIE:** la tarea llamó *nExitTask* y espera *nWaitTask*.
  - **WAIT-REPLY:** la tarea hizo *nSend* y espera *nReply.*
  - **WAIT-SEND:** La tarea hizo *nReceive* y espera *nSend.*
  - **WAIT-READ:** La tarea está bloqueada en un *read*.
  - **WAIT-WRITE:** la tarea está bloqueada en un *write*.
  - Entre otros…

CAMBIO DE CONTEXTO

- Consiste en el traspaso de la CPU de un proceso a otro.
- Esto involucra ciertas acciones:
  1. Resguardar los registros
  2. Contabilizar uso de los recursos (NSystem no tiene contabilización)
  3. Cambiar el espacio de direcciones virtuales. (esto es lo más caro en el costo final del cambio de contexto).
  4. Restaurar registros del proceso receptor de la CPU.
- Estas labores del scheduler consumen algo de tiempo de procesador y por lo tanto son sobrecosto puro.

TAJADA DE TIEMPO Y RÁFAGA DE CPU

- **Tajada de tiempo:** rango de tiempo asignado por el kernel a un proceso para que ejecute sus instrucciones. En un sistema multiprogramado, el kernel, asigna seguidamente tajadas a cada proceso que se ejecuta (entremezclando sus tajadas) de manera que todos avancen “concurrentemente”.

- **Ráfaga de CPU (CPU Burst):** secuencia de instrucciones ejecutadas sin pasar a modo de espera.

  Al final de una ráfaga siempre hay que hacer un cambio de contexto.

DESCRIPTOR E IDENTIFICADOR DE PROCESO

- **Identificador de proceso:** es la manera de identificar a los procesos ente sí fuera del núcleo. Ej: PID (Process ID) en Unix/Linux.

- **Descriptor de proceso:** es la estructura de datos que representa a un proceso dentro del núcleo. Almacena:
  - Estado del proceso
  - Registros (virtuales)
  - Información de scheduling
  - Recursos asignados
  - Contabilidad
  - Otros.

DESCRIPTOR EN NSYSTEM

- NSYSTEM define el siguiente descriptor de proceso:

```
/*DESCRIPTOR DE NSYSTEM*/
typedef struct Task {
   int status; /* READY, ZOMBIE,WAIT_TASK,WAIT_REPLY,WAIT_SEND,…*/
   char *taskname;
   SP sp; /*tope de la pila cuando la tarea está suspendida*/
   SP stack; /*tope de la pila*/
   struct Task *nextTask;
   void *queue
    /*para nExitTask y nWaitTask*/
    int rc;
    struct Task *wait_task;
   /* nsend, nReceive,nReply*/
   struct queue * send-queue;
   union {void * mesg; int rc;} send;
   int wake_timer;
   int in_wait_squeue
 }*nTask;
```

SCHEDULING DE PROCESOS

- Corresponde a la planificación de procesos en un S.O.
- El kernel asigna el CPU por turnos a los procesos que pueden estar activos.
- Hay distintas estrategias para asignar estos turnos, dependiendo del objetivo que se persiga.
- La asignación estratégica del procesador a los procesos es lo que se denomina **scheduling de procesos** o **planificación de procesos**.
- La componente del núcleo que se encarga de esta labor se denomina **scheduler del procesador**.

SCHEDULING DE PROCESOS (continuación)

- Existen 3 tipo de planificación:
  - **De corto plazo:** ejecución de procesos por el kernel
  - **De mediano plazo:** relacionado con la técnica de memoria virtual
  - **De largo plazo:** usada para programar ejecución de procesos como por ejemplo: administrador de impresión, recolector de basura, etc.

- Además de scheduling de procesos, también se realiza scheduling de los accesos a disco.

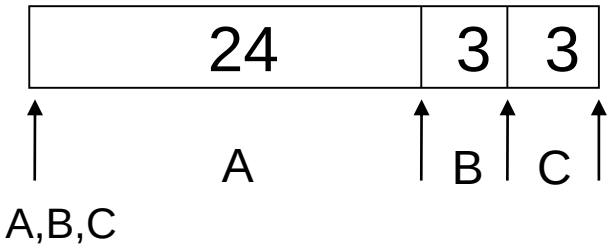
- Para ello existe un scheduler asociado a cada disco.

- Este scheduler ordena estratégicamente los mútiples accesos a disco de varios procesos con el fin de minimizar el tiempo de acceso.

SCHEDULING DE PROCESOS: FCFS

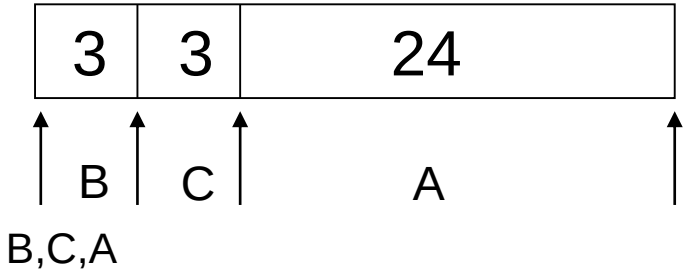
- Significa **First Come First Served**
- La idea es “minimizar los cambios de contexto”
- Es Non-Premptive
- La ráfaga se atiende de principio a fin sin cambio de contexto
- El orden de atención es FIFO
- Existe un parámetro que permite mostrar cuan mala es esta estrategia y se llama “Tiempo de despacho”
    - **Tiempo de despacho:** tiempo comprendido desde que el proceso pasa de modo READY hasta que pasa a un modo de espera (WAIT…).

SCHEDULING DE PROCESOS: FCFS



$$\text{T. Despacho} = \frac{24 + 27 + 30}{3} = 27$$

Ahora en otro orden:



$$\text{T. Despacho} = \frac{3 + 6 + 30}{3} = 13$$

Bastante mejor!

Ventajas:

- Simple de implementar (con una cola FIFO)

Desventajas:

- No sirve para procesos interactivos
- No se distribuye bien la carga entre procesos intensivos en CPU y aquellos intensivos en E/S.

SCHEDULING DE PROCESOS: SJF

- Significa **Shortest Job First** "El trabajo más corto primero"
- Minimiza el tiempo de despacho
- Atiende primero a los procesos que se "espera" que tengan ráfagas cortas.
- Ejemplo:

Proceso	Duración ráfaga
A	5
B	2
C	20
D	1

Se ejecutan en este orden según SJF:
D, B, A, C

- Problema: No sabemos cuánto durará la ráfaga de cada proceso. Necesitamos una manera de predecir.

SCHEDULING DE PROCESOS: SJF

La estrategia SJF es preemptive (S.O. quita el cpu), posibles casos:

- Cambio de contexto cuando llega un proceso con predictor menor.
- Cambio de contexto cuando se excede el predictor para el proceso sgte.

SJF con prioridades

- Se entrega la CPU al proceso con mayor prioridad (prioridad = $-T_{n+1}^{p}$)

Desventaja: hambruna

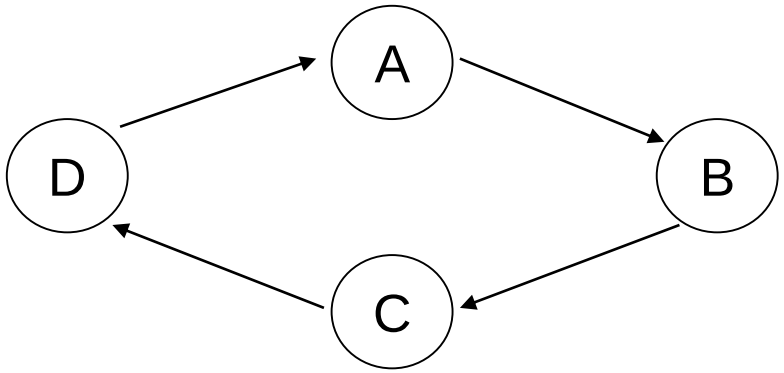
Variante: Aging (es un parche)

Cada cierto tiempo se incrementa “temporalmente” la prioridad de los procesos READY

SCHEDULING DE PROCESOS: ROUND ROBIN

- El scheduler asigna tajadas de tiempo fijo de 10 a 100 milisegundos a cada proceso.

- Los procesos se organizan en forma circular:



- Esta estrategia minimiza el tiempo de respuesta.

  En este caso, el tiempo transcurre desde que se inicia una interacción hasta que aparece el primer resultado (parcial) → tiempo de respuesta.

SCHEDULING DE PROCESOS: ROUND ROBIN

- Implementación: cola FIFO (es muy simple)
- El problema es cómo establecer el tamaño de la tajada.
  - ¿tamaño de la tajada?
    - Es ideal tener tajadas finas, así el tiempo de respuesta es más corto pero los cambios de contexto aumentan (como costo adicional).
  - El 80% de las ráfagas duran menos de una tajada

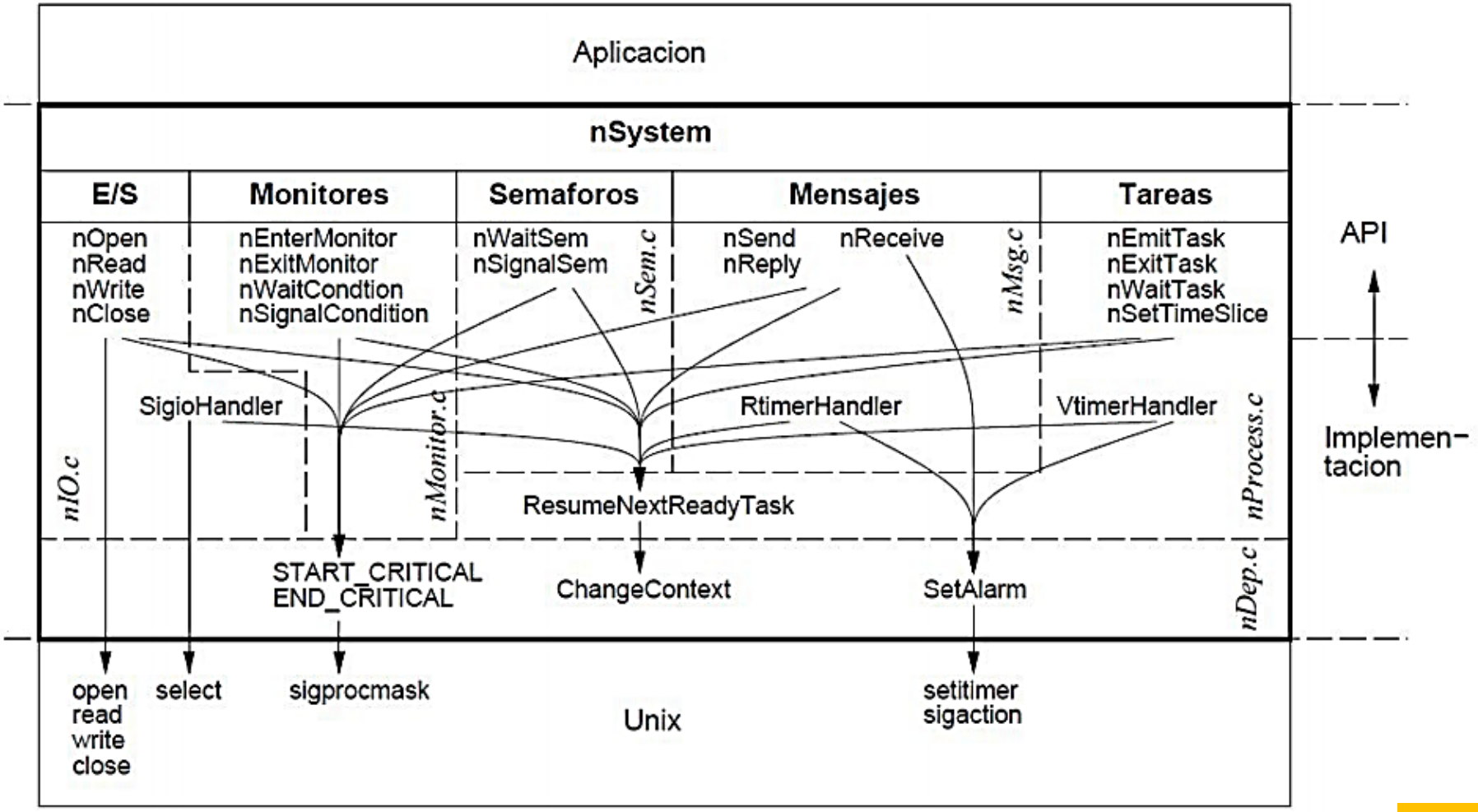
ADMINISTRACIÓN DE PROCESOS EN NSYSTEM

- El nSystem es un sistema de procesos livianos o nano tareas que corren en un solo proceso pesado de Unix.

- Las tareas de nSystem se implementan multiplexando el tiempo de CPU del proceso Unix en tajadas.

- Esto se puede hacer gracias a que:
  - un proceso Unix tiene asociado su propio reloj regresivo
  - un mecanismo de interrupciones (señales)
  - E/S no bloqueante
  - y todas las características del hardware que se necesitan para implementar multiprogramación en un solo procesador, con la salvedad de que este hardware es emulado en software por Unix.

ORGANIZACIÓN DE NSYSTEM

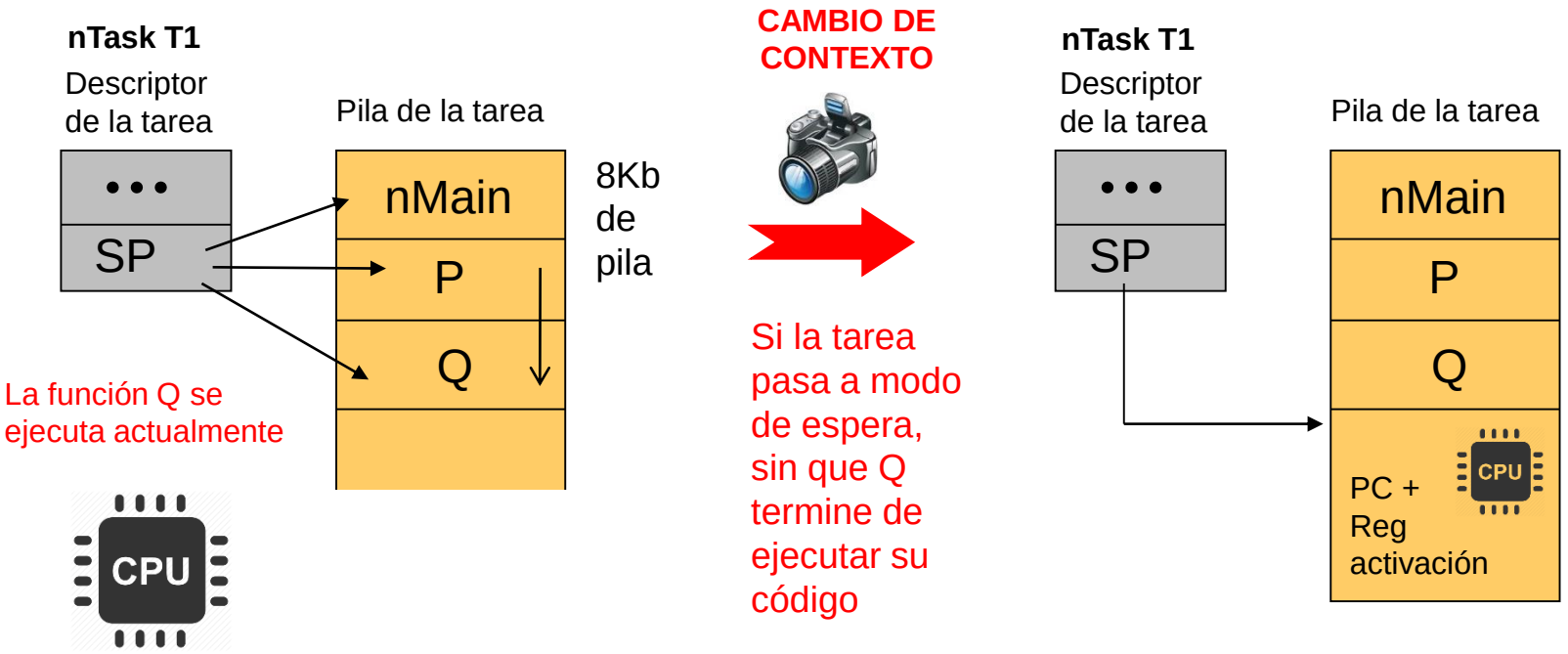


LA PILA EN NSYSTEM

- Cada nano tarea tiene su propia pila.
- Los registros de activación de los procedimientos que se invocan en una tarea se apilan y desapilan en esta estructura de datos.
- Un registro del procesador, denominado **sp**, apunta hacia el tope de la pila.
- La pila es de tamaño fijo, por lo que una recursión demasiado profunda puede desbordarla.
- El nSystem chequea el posible desborde de la pila en cada cambio de contexto. → termina con un mensaje de error.
- Sin embargo, si no hay cambios de contexto no puede detectar el desborde. En este caso, el nSystem podría terminar en un segmentation fault o bus error, sin mayor explicación de la causa de la caída.

CAMBIO DE CONTEXTO EN NSYSTEM

- Nunca hay más de una nano tarea corriendo realmente. El registro **sp** del procesador apunta al tope de la pila de la tarea que está corriendo.



- Se utiliza la función: `void ChangeContext(nTask sale, nTask entra);`

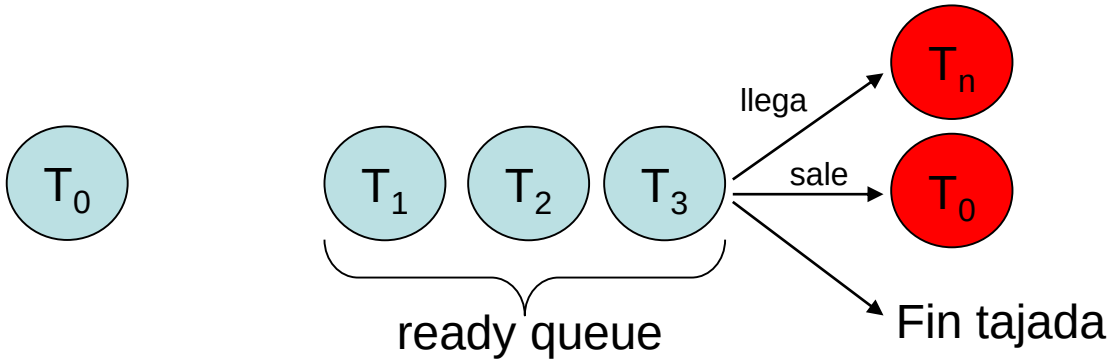
(19 instrucciones assembler en nstack-i386.s)

SCHEDULING EN NSYSTEM

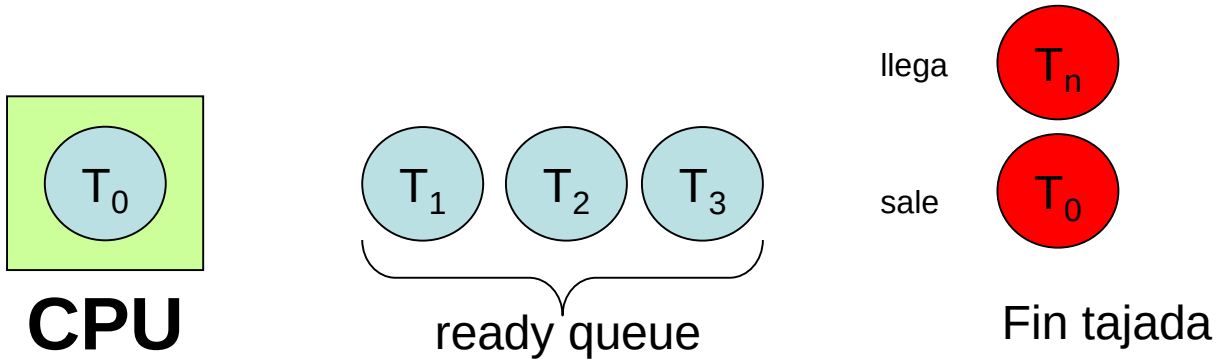
- NSYSTEM utiliza su propia estrategia de planificación de procesos y se llama: **PLCFS** o **Preemptive Last Come First Served**.

- Funciona así:

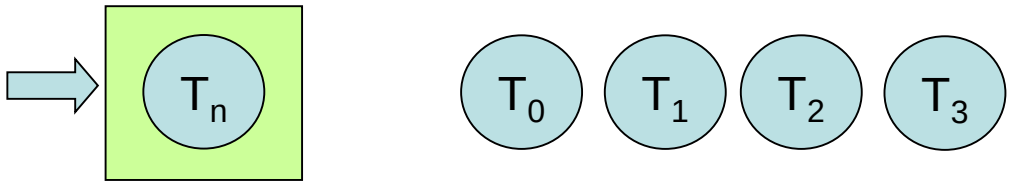
  - Las ráfagas se atienden en orden LIFO
  - La tarea que llega, recibe la CPU de inmediato
  - La tarea perdedora se coloca en primer lugar en la cola READY
  - Cada t milisegundos se suspende la tarea en ejecución y se lleva al final de la cola.



SCHEDULING EN NSYSTEM

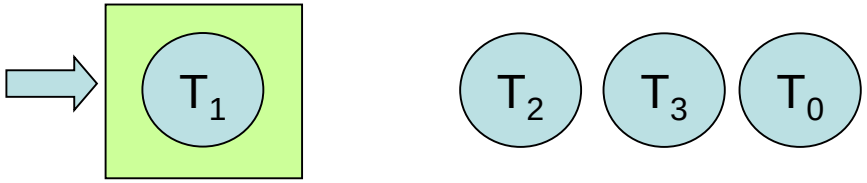


1) llega T_n



2) sale T_0

3) fin tajada



SCHEDULING EN NSYSTEM (Simplificado)

- La siguiente es una versión simplificada del código del scheduler que no considera ciertas condiciones de borde que obscurecen innecesariamente el código.

- Estas condiciones son el caso en que no hay tareas listas para correr y el caso en que el scheduling es non-preemptive.

- El scheduler mantiene las siguientes variables:

```
nTask current_task;   /*tarea actual en ejecución en el procesador*/

Queue ready_queue;    /*cola de procesos listos*/

int current_slice;    /*duración de la taja de tiempo*/
```

SCHEDULING EN NSYSTEM (Simplificado)

- Funciones para cambio de contexto por fin de tajada:

```
void Resume() { /*para no escribir ResumeNextReadytask()*/
  nTask esta = current_task;
  nTask prox = GetTask(ready_queue);
  current_task = prox;
  ChangeContext(esta, prox);
  current_task = esta;
}
```

- En cada señal periódica se invoca el siguiente procedimiento:

```
void VtimerHandler() {
  setAlarm(VIRTUALTIMER, current_slice, VtimerHandler);
  PutTask(ready_queue, current_task);
  Resume();
}
```

Para implementar `SetAlarm` se usaron las llamadas al sistema `setitimer` y `sigaction`.

SECCIONES CRÍTICAS EN NSYSTEM

- Hemos supuesto que hay un solo procesador (1 solo core).
- La única manera de que 2 tareas lleguen a entremezclar la ejecución de sus procedimientos es por medio de una señal del timer.
- Por lo tanto, la implementación más simple de la sección crítica consiste en inhibir las señales para evitar cambios de contexto dentro de la sección crítica.
- Esto se logra con los llamadas START_CRITICAL y END_CRITICAL.
- El procedimiento START_CRITICAL inhibe las señales de Unix usando la llamada al sistema `sigprocmask`.
- La misma llamada se usa en END_CRITICAL para reactivar las señales.

EJERCICIO

- Implementar el mecanismo de semáforos presente en nSystem:
  - `nSem nMakeSem(int n);`
  - `void nWaitSem(nSem s);`
  - `void nSignalSem(nSem s);`

```
typedef struct {
  int c;
  queue q;
} *nSem;

void nWaitSem(nSem s) {
  START_CRITICAL();
  if (s->c > 0)
    s->c--;
  else {
    current_task->status= WAIT_SEM;
    PutTask(s->q, current_task);
    Resume();
  }
  END_CRITICAL();
}
```

```
void nSignalSem(nSem s) {
  START_CRITICAL();
  if (EmptyQueue(s->q))
    s->c++;
  else {
    nTask w= GetTask(s->q);
    w->status= READY;
    PushTask(ready_queue, current_task);
    PushTask(ready_queue, w);
    Resume();
  }
  END_CRITICAL();
}
```