udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Calendarizador

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er Semestre 2015

Adaptado de Dave Eckhardt

¿Qué vamos a discutir hoy?

- Calendarización (OS:P+P capítulo 7, OSC capítulo 6)
- Terminología
  - Tiempo de espera significa tiempo gastado en estado ejecutable pero atorado en una cola de calendarizador (no el tiempo esperando que un evento nos despierte)
  - Tarea significa algo que el calendarizador calendariza (hilo, proceso, o algo ejecutable)
- Concepto de calendarización
- Algoritmos
- Sistemas en tiempo real

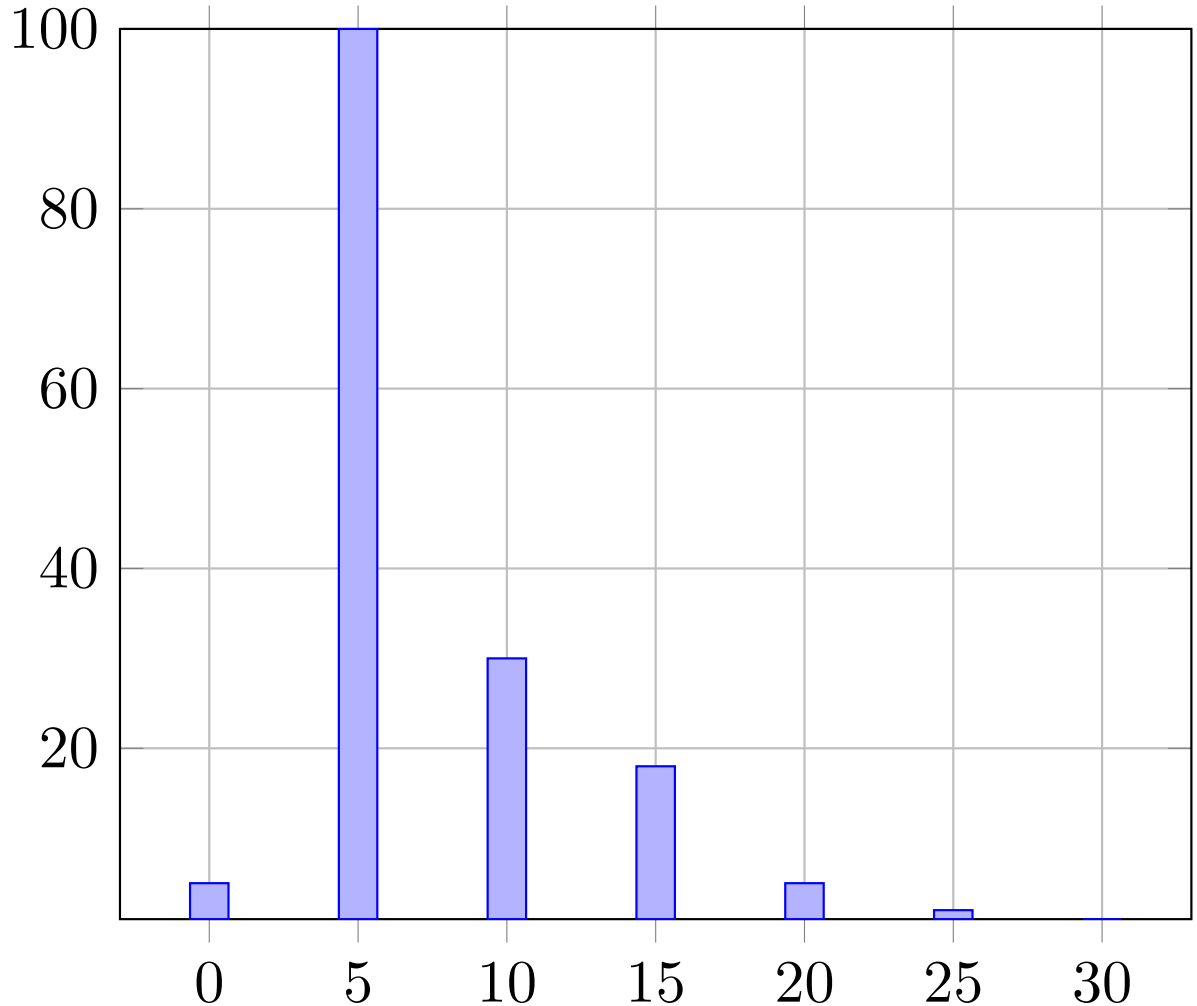
Ciclos CPU I/O

- Vista de procesos: 2 estados
  - Ejecutando (que corre)
  - Bloqueado en I/O
  - Ciclo de vida: I/O (cargando el ejecutable), CPU, I/O, CPU, ..., CPU (`exit()`)
- Vista del sistema
  - Ejecutando (que corre)
  - Bloqueado en I/O
  - Ejecutable (que puede correr, i.e., esperando) —no hay suficientes procesadores ahora
- Ejecutando $\Rightarrow$ los bloqueos dependen del proceso mayormente
  - ¿Cuánto tiempo se ejecuta un proceso antes de bloquearse?

Longitud de ráfagas de CPU

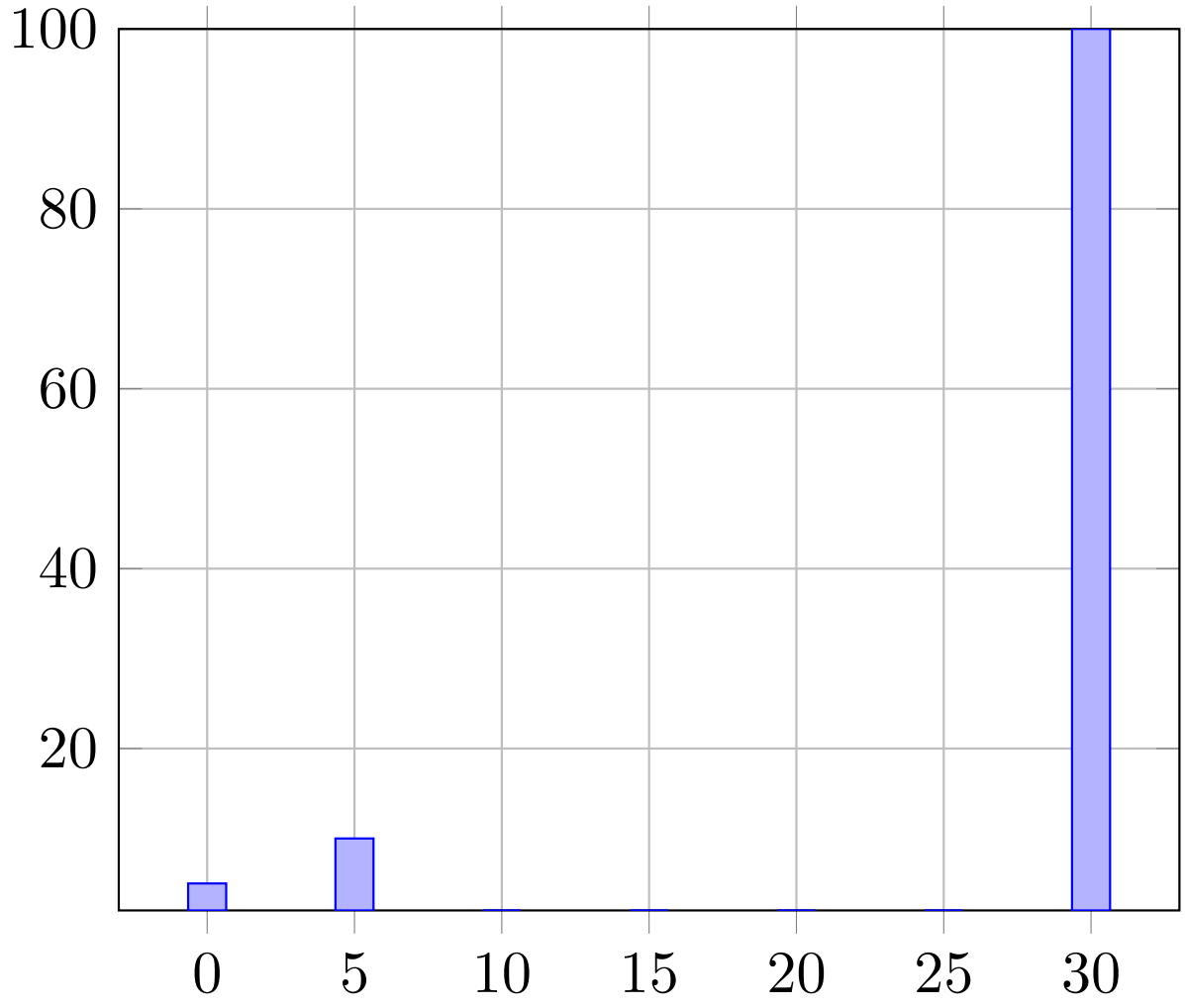
General

- Caída exponencial en la longitud de las ráfagas del CPU



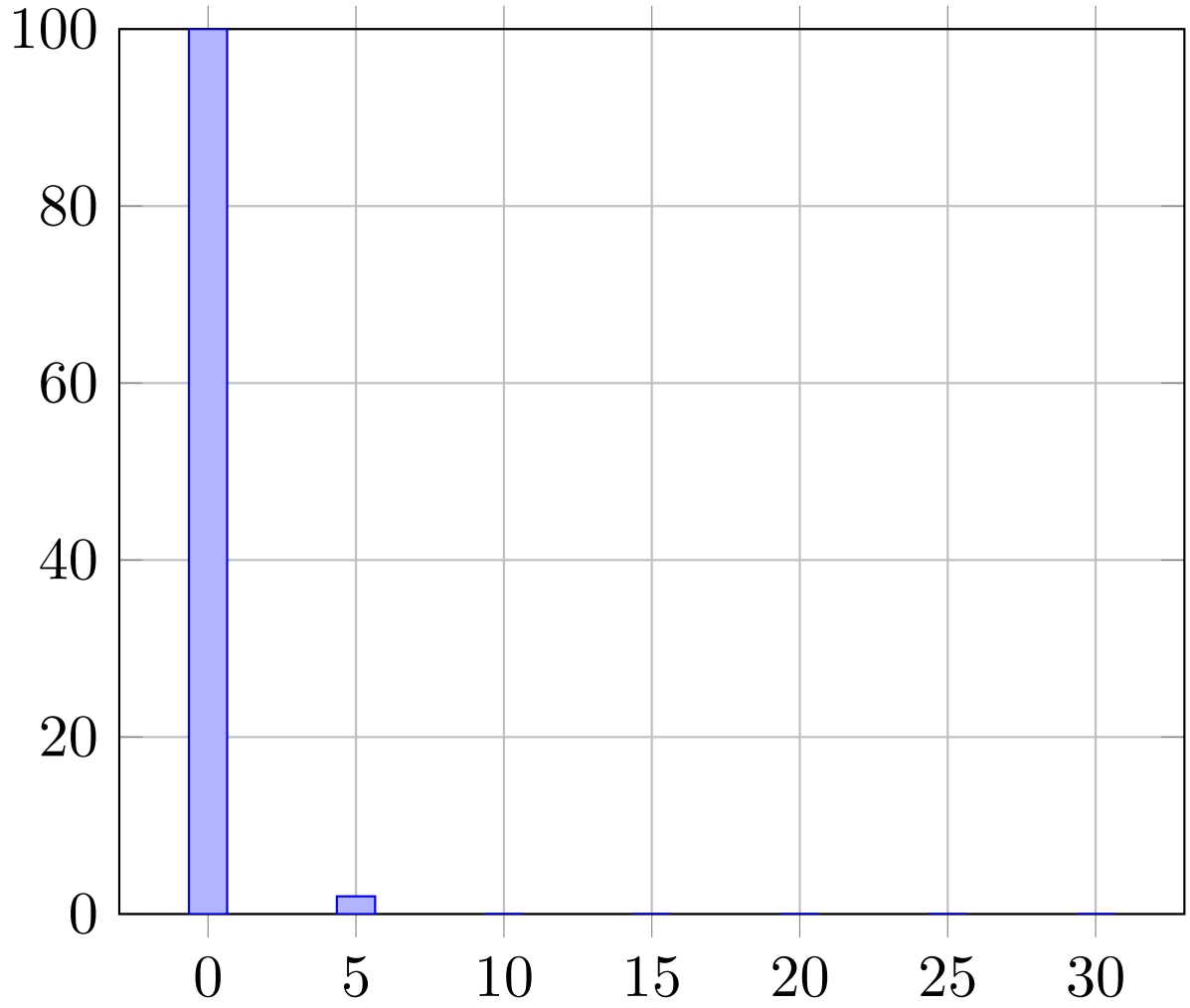
Programa acotado por el CPU

- Trabajo en batch
- Altas ráfagas de CPU



Programa acotado por el I/O

- Copiar, adquirir datos
- Muy pequeñas ráfagas de CPU entre llamadas de sistema



¿Por qué calendarizar?

- ¿Qué pasa si permitimos que un programa acotado por el CPU se ejecute hasta terminar?
  - ¿Qué pasa a los programas acotados por I/O?
- ¿Qué pasa si permitioms que un programa acotado por el I/O se ejecute hasta terminar?
  - ¿Qué pasa a los programas acotados por CPU?

¿Puede apropiar?

- Cuatro oportunidades para calendarizar
  - Un procesos que se está ejecutando se bloquea (I/O, error de página, `wait()`, etc.)
  - Un proceso que se está ejecutando termina
  - Un proceso bloqueado se ejecuta (I/O completado)
  - Otra interrupción (reloj)
- Tipos multitarea
  - Totalmente interrumpible: todos los tipos anteriores causan calendarización
  - Cooperativo: solamente los dos primeros

Kernel apropiativo

- Multitarea apropiativa
  - Todos los cuatro casos causan cambios de contexto
- Kernel apropiativo
  - Todos los cuatro casos causan cambio de contexto en el modo de kernel
  - Las llamadas al sistema inhabilitan las interrupciones solo cuando es realmente necesario
  - Las interrupciones del reloj deben de suspender la ejecución de llamadas del sistema
    - E.g., `fork()` debe de parecer atómico, pero no ejecutarse de esa manera

Calendarizador del CPU

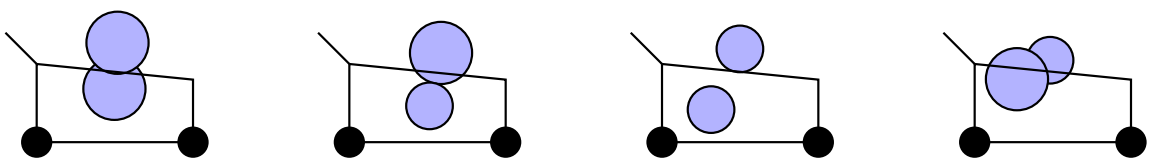
- Invocado cuando el CPU se encuentra en ocioso y/o ha pasado cierto tiempo
  - La tarea actual se bloquea
  - El reloj interrumpe
- Seleccionamos la siguiente tarea
  - Rápidamente
  - PCB está en FIFO, cola de prioridad, árboles, etc.
- Cambiamos (utilizamos un despachador)
  - El término puede cambiar

Despachador

- Deja la tarea a ejecutarse
  - Salva el estado de los registros
  - Actualiza la información de uso del CPU
  - Almacena el PCB en una "cola de ejecución"
- Toma la tarea designada
  - Activa la memoria de la nueva tarea
    - Protección, mapeo
  - Restaura el estado de los registros
  - Regresa a cualquier tarea que estaba realizando anteriormente

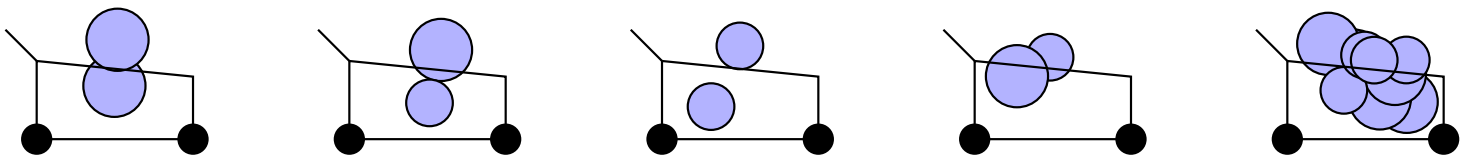
Consideremos

- ¿Quién pasa de primero? ¿Último?



Consideremos

- ¿Quién pasa de primero? ¿Último?
- Ahora, ¿quién pasa primero? ¿Y último?



Criterios de calendarización

- Vista del administrador del sistema
  - Maximizar/comprometer
  - Utilización del CPU (que tan ocupado)
    - Era importante cuando las computadoras eran caras
    - Ahora calor y poder son más costosos que el silicio (*silicon*)
  - Rendimiento (trabajos por segundo)
- Vista del proceso
  - Minimizar
  - Tiempo de respuesta (todo, `fork()` a `exit()`)
  - Tiempo de espera (ejecutable pero no ejecutando)
- Vista del usuario (procesos interactivos)
  - Minimizar tiempo de respuesta (latencia de entradas y salidas)
  - Tiempo de respuesta predecible (saber "por qué está lenta hoy")

Algoritmos

- No intenten estos en casa
  - FCFS —*First come, first served*
  - SJF —*Shortest job first*
  - Prioridad
- Razonables
  - Round Robin
  - Multi nivel (más retroalimentación)
- Multiprocesador
  - Balanceado de carga
  - Afinidad de procesador
- Tiempo real

FCFS —*First come, first served*

- Idea básica
  - Ejecutar una tarea hasta que entregue el CPU
  - Cuando sea ejecutable, colocar al final de la cola FIFO
- Tiempo de espera depende de la mezcla de los procesos
  - Algunos procesos se ejecutan por poco tiempo
  - Otros por mucho más tiempo
- Efecto convoy
  - N tareas hacen 1 solicitud I/O, demoran (e.g., copia de archivos)
  - 1 tarea ejecuta una ráfaga muy larga en el CPU
    - Todas las tareas I/O se convierten en ejecutables durante este tiempo
  - Después, repetimos
    - Resultado: N tareas acotadas por I/O no pueden tener los dispositivos I/O ocupados

SJF —*Shortest job first*

- Idea básica
  - Escoger la tarea con la ráfaga de CPU más pequeña
  - Entregará el CPU más rápidamente, es amable con las otras tareas
- Probablemente óptimo
  - Minimiza la espera promedio entre tareas
- Prácticamente imposible
  - ¿Podemos predecir la siguiente ráfaga de CPU?
  - El texto sugiere promediar las ráfagas recientes
  - Algunas tareas pueden decirnos cuanto trabajo les falta por realizar
    - Harchol-Balter et al., "Size-Based Scheduling to Improve Web Performance", ACM TOCS 21:2, 5/2003

Prioridad

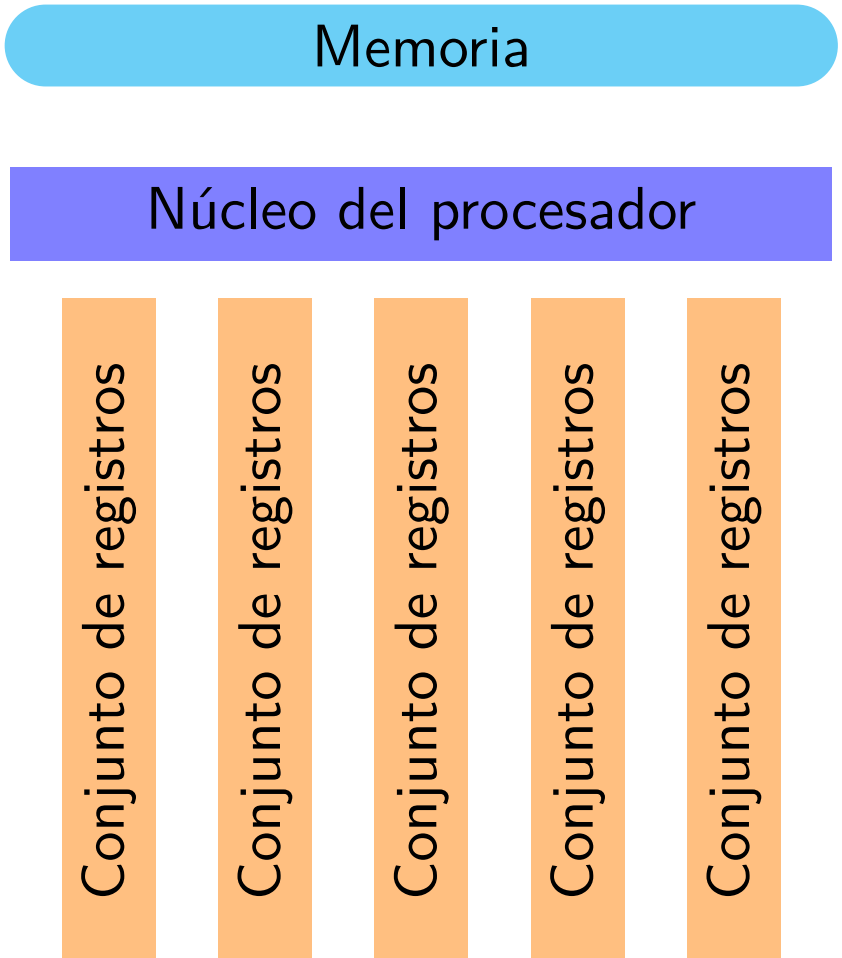
- Idea básica
  - Escoger la tarea que espera más importante
  - Según el sistema, la prioridad mayor puede ser $p = 0$ o $p = 255$
- Asignación de la prioridad
  - Estática: propiedad fija (¿creada?)
  - Dinámica: función del comportamiento de la tarea
- Problema: inanición
  - "La tarea más importante" se ejecuta a menudo
  - "La tarea menos importante" puede nunca ejecutarse
  - Hack común: prioridad según envejecimiento

Round Robin

- Idea básica
  - Ejecutar cada tarea por un tiempo determinado
  - Cuando el período expire, agregamos la tarea a la cola FIFO
- "Justo"
  - Probablemente "no óptimo"
- Escoger la longitud del período
  - Infinito (hasta que el proceso haga I/O), equivalente a FCFS
  - Infinitesimal (1 instrucción), equivalente "compartir procesador"
  - Balancear justicia vs. costo de cambio de contexto

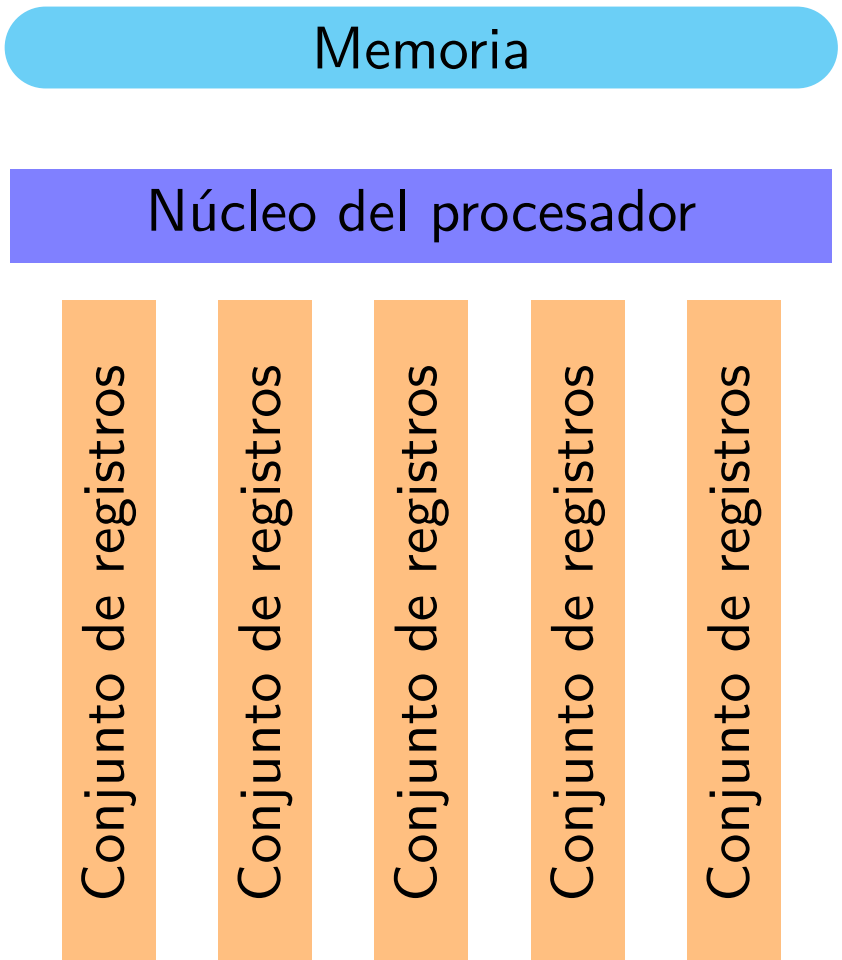
Verdadero procesador compartido

- Latencia de memoria
  - Larga, constante fija
  - Cada instrucción tiene un operando de memoria
- Solución: round robin
  - Período: 1 instrucción
  - Un proceso ejecutándose
  - $N - 1$ procesos esperando en memoria



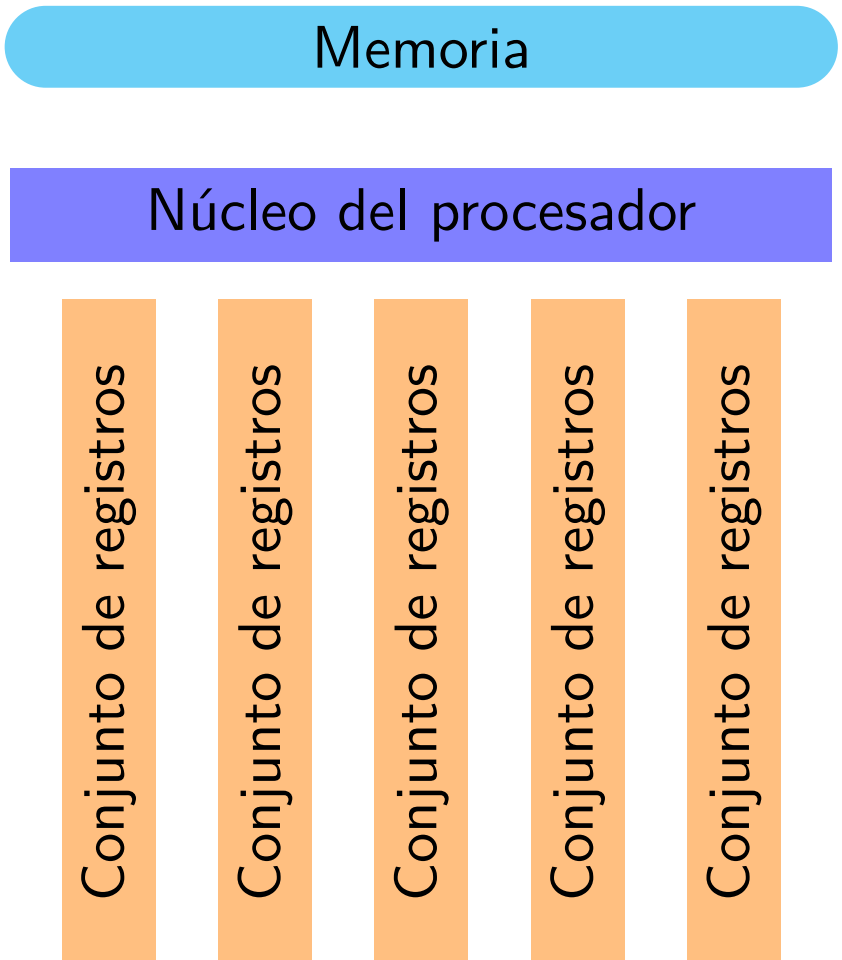
Verdadero procesador compartido

- Cada instrucción
  - Computación breve
  - Una carga o un almacenamiento
    - Duerme a los procesos N ciclos
- Estado estable
  - Ejecutar cuando está listo
  - Está listo cuando es su turno



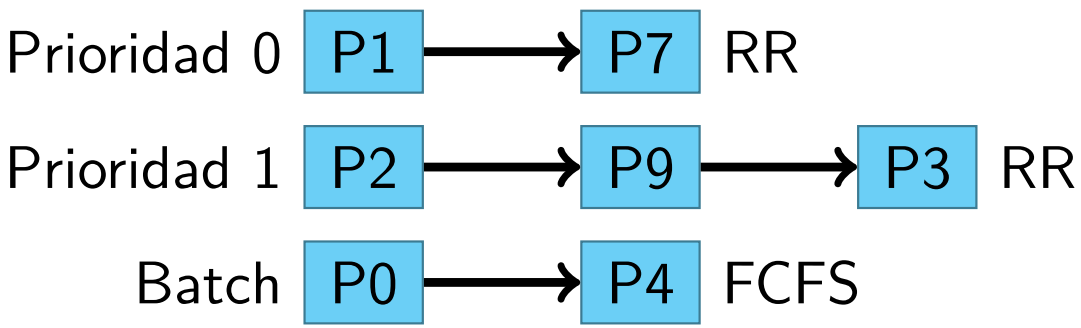
Todo lo viejo es nuevo de nuevo

- Intel *hyperthreading*
- N conjuntos de registros
- M unidades funcionales
- Cambiar en operaciones de larga ejecución
- Compartir es menos regular
- La ilusión de compartir es más torpe
  - Bueno para algunas mezclas de aplicaciones
  - Malo para otras
  - "Hyperthreading Hurts Server Performance, Say Developers", ZDNet UK, 18-11-2005



Cola multi nivel

- N colas independientes de procesos
  - Una por prioridad
  - Un algoritmo por cola



¿Calendarización inter cola?

- Prioridad estricta
  - Prioridad 0 se ejecuta antes que Prioridad 1
  - Prioridad 1 se ejecuta antes que batch
  - Cada vez
- Rebanadas de tiempo (e.g., round robin ponderado)
  - Prioridad 0 toma dos porciones
  - Prioridad 1 toma una porción
  - Batch obtiene una porción

Cola multi nivel con retroalimentación

- N colas, diferentes períodos
- ¿Bloqueamos o dormimos antes de que expire el período?
  - Agregada al final de nuestra cola ("buen ejecutable")
- ¿Terminamos nuestro período?
  - Nos degradan a una cola más lenta ("mal ejecutable")
  - Una prioridad más baja equivale a un período más largo
- ¿Nos pueden promover de nuevo?
  - Tal vez I/O nos promueva
  - Tal vez envejezcamos, y nos promuevan
- Calendarizador popular para tiempo compartido

Calendarización multiprocesador

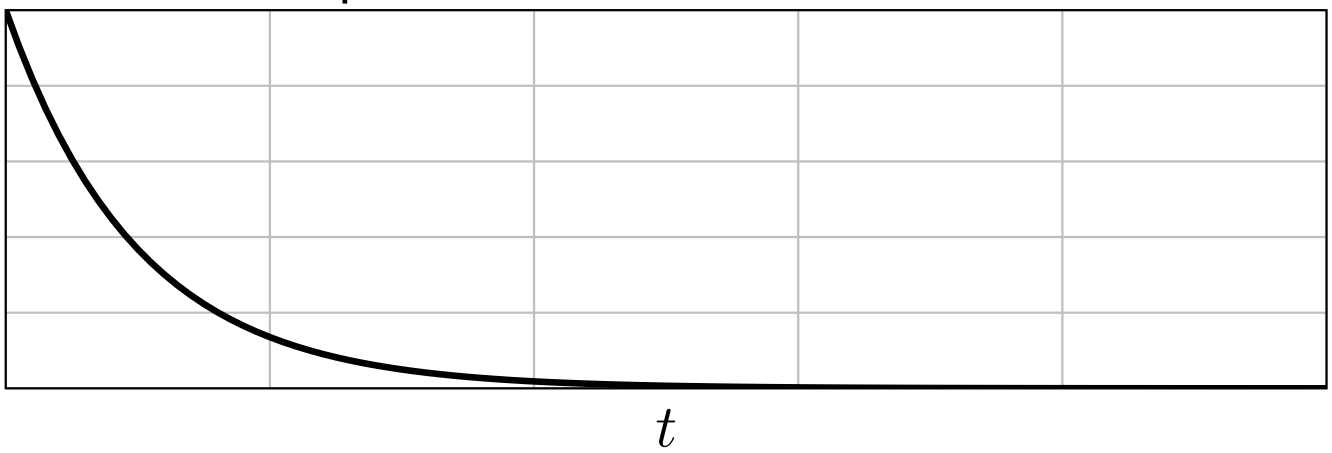
- Suposiciones comunes
  - Procesadores homogéneos (misma velocidad)
  - Acceso a la memoria uniforme
- Objetivo: carga compartida, y balanceo de carga
  - Fácil: una cola global de "listos" —no hay falsa ociosidad
- Pero
  - La cola global de "listos" es un punto de contención "caliente"
  - Afinidad de procesador: algún procesador puede ser más deseable o necesario
    - Dispositivo especial I/O
    - Cambio rápido de hilos
    - Continuar en el CPU más reciente puede encontrar algunas cosas todavía en el cache
    - $1/N$-ésimo de la memoria puede ser más rápido

Enfoques de evaluación de la calendarización

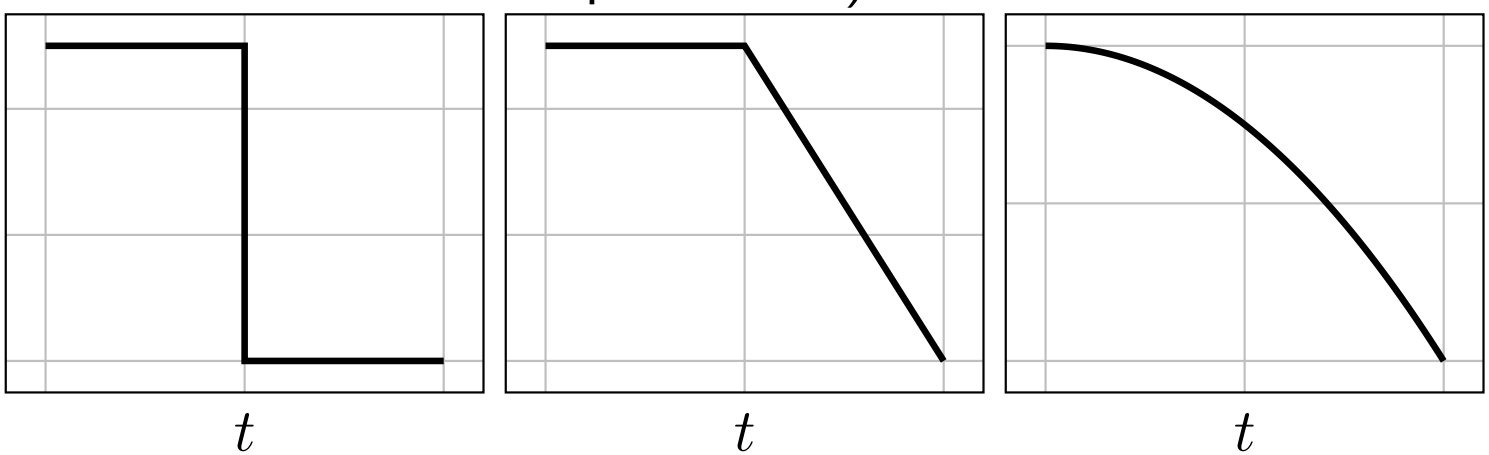
- Modelado determinista
  - Ejecución manual
- Teoría de colas
  - A menudo entrega aproximaciones rápidas y útiles
  - La matemática se vuelve compleja rápidamente
  - La matemática es sensible a suposiciones
    - Pueden ser irreales, por lo tanto equivocadas
- Simulación
  - Modelo de carga de trabajo o dirigido por trazas
  - Peligro de GIGO (*garbage in, garbage out*)

Calendarización en tiempo real

- Valor de computación

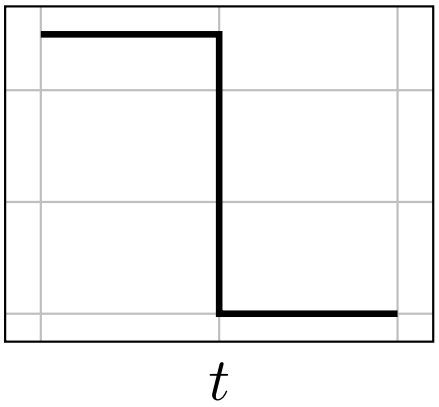


- Tiempo real: no valor extra si es temprano (o en algunos casos la curva decae rápidamente)

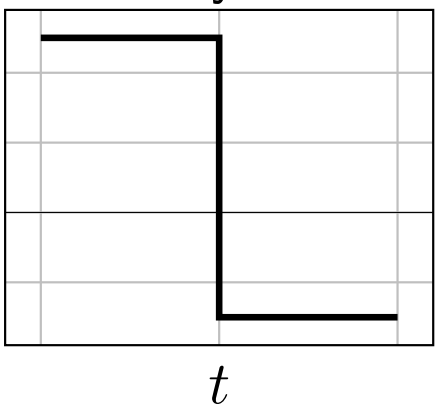


Tiempo real duro

- Múltiples definiciones utilizadas
  - Tiempo rápido de respuesta —¿$10\,\mu s$?
  - No hay valor si los resultados llegan tarde
  - Muy costoso si llegamos tarde
  - Nunca es tarde



No hay valor



Muy costoso

Calendarización en tiempo real duro

- Los diseñadores deben describir los requerimientos de las tareas
  - Secuencia de instrucciones del peor caso de ejecución (según el tiempo)
- Demostrar el tiempo de respuesta del sistema
  - Argumentado o un verificador automático
- No puede utilizar tecnologías de tiempo indeterminado
  - Discos, redes, etc.
- Las soluciones involucran
  - Diseño simplificado
  - Sistemas con sobre ingeniería
  - Hardware dedicado
  - Sistemas operativos especializados

Calendarización en tiempo real relajado

- La realización de la computación aún tiene valor después del límite
  - Pensemos en "interfaces de usuario"
  - Varios sistemas de control
    - Si el sistema no ha movido el elevador en 50 ms, probablemente es bueno moverlo dentro de los 100 ms
- El rendimiento no es crítico (nadie morirá)
  - Un video de youtube
  - Skype
    - Aunque los paquetes que llegan tarda causan una pérdida de audio
  - Software que escribe en CDs
    - Puede resultar en CD corruptos

Conclusiones

- Round robin está bien para casos simples
- Sistemas “reales”
  - Retroalimentación multi nivel
  - Probablemente algunos son tiempo real relajado
  - Calendarización en multi-procesador es importante
- Conceptos de sistemas en tiempo real
  - Terminología: relajado (suave), duro, tiempo límite
  - Problema principal: “inversión de prioridad” (ver el texto)