Instituto de Matemática e Estatística

USP

MAC422
Sistemas Operacionais

Prof. Marcel P. Jackowski

Aula #12

Escalonamento de processos e threads

Até agora...

- Interrupções, processos, threads e sincronização
  - Mecanismos
- Recursos: elementos que processos e threads utilizam
  - Tempo de CPU, memória, espaço em disco, etc.
  - Políticas

Tipos de recursos

- Preemptible
  - OS can take resource away, use it for something else, and give it back later
    - E.g., CPU
- Non-preemptible
  - OS cannot easily take resource away; have to wait after the resource is voluntarily relinquished
    - E.g., disk space
- Type of resource determines how to manage

Algumas decisões

- Allocation: which process gets which resources
  - Which resources should each process receive?
  - Space sharing: Controlled access to resource
  - Implication: resources are not easily preemptible

- Scheduling: how long process keeps resource
  - In which order should requests be serviced?
  - Time sharing: more resources requested than can be granted
  - Implication: Resource is preemptible

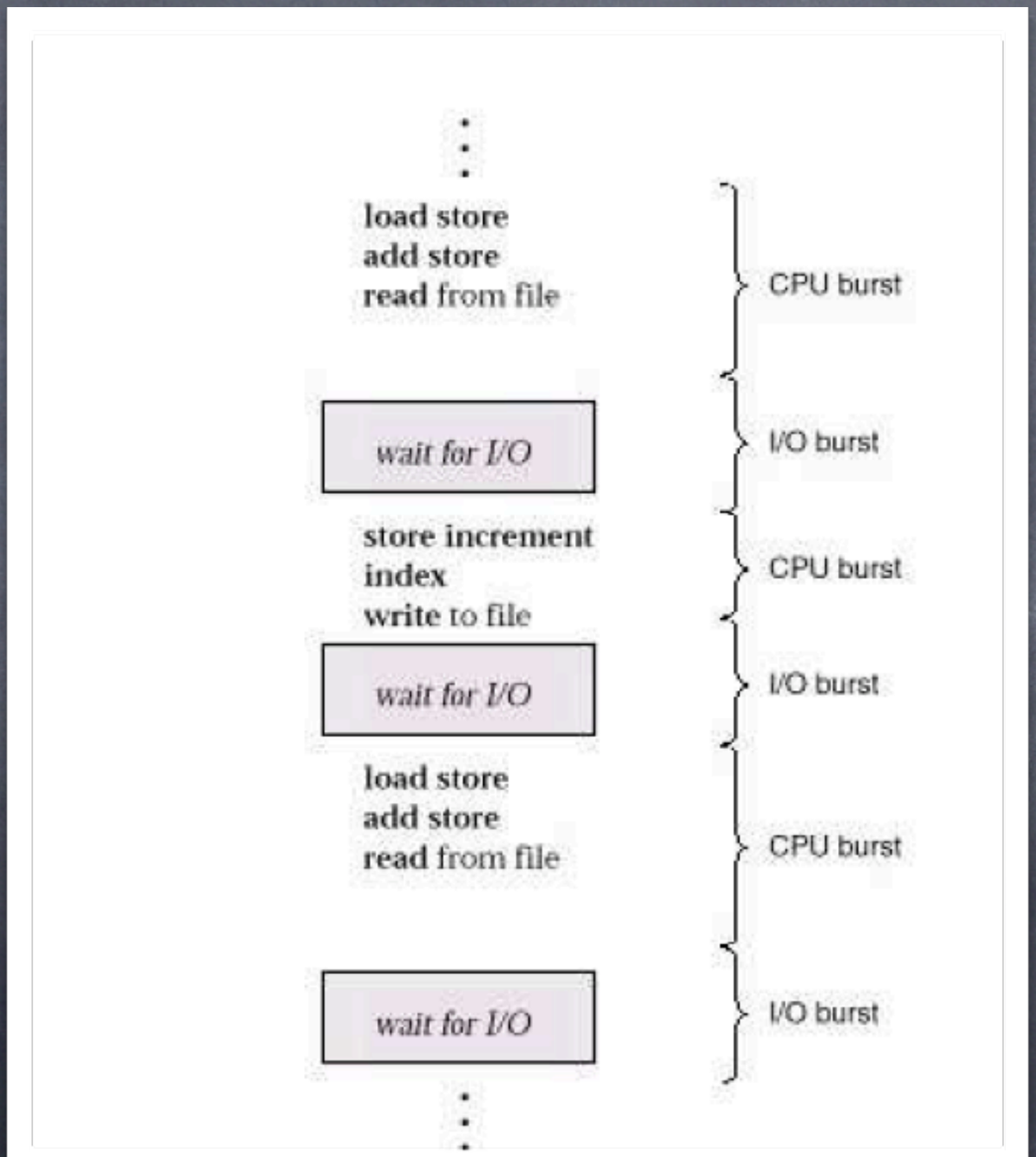
Despachante vs. escalonador

- Dispatcher
  - Low-level mechanism
  - Responsibility: context switch
- Scheduler
  - High-level policy
  - Responsibility: deciding which process to run

Quando escalonar ?

- When does scheduler make decisions?
  - When a process

Ciclos de uso de CPU ou E/S



Critérios

- Criteria: workload and environment
- Workload
  - Process behavior: alternating sequence of CPU and I/O bursts
  - CPU bound v.s. I/O bound
- Environment
  - Batch v.s. interactive?
  - Specialized v.s. general?

Escalonamento

- ***Short term scheduling***
  - Decide qual processo é o próximo a tomar o tempo da CPU
- ***Long term scheduling***
  - Decide qual processo sai do dispositivo de armazenamento secundário e entra na fila de Pronto.
  - Multiprogramação.

Métricas de performance

- Min waiting time: don't have process wait long in ready queue
- Max CPU utilization: keep CPU busy
- Max throughput: complete as many processes as possible per unit time
- Min response time: respond immediately
- Fairness: give each process (or user) same percentage of CPU

First Come, First Served (FCFS)

- Também chamado de FIFO (First In, First Out)

- Simplest CPU scheduling algorithm
  - First job that requests the CPU gets the CPU
  - Nonpreemptive

- Implementation: FIFO queue

Exemplo de FCFS

Process	Arrival Time	Burst Time
P_1	0	7
P_2	0	4
P_3	0	1
P_4	0	4

- Gantt chart

Schedule: P_1 P_2 P_3 P_4

- Average waiting time: (0 + 7 + 11 + 12)/4 = 7.5

Outro exemplo de FCFS

Arrival order: P_3 P_2 P_4 P_1

- Gantt chart

Schedule:

- Average waiting time: (9 + 1 + 0 + 5)/4 = 3.75

Vantagens e desvantagens do FCFS

- Advantages
  - Simple
  - Fair

- Disadvantages
  - waiting time depends on arrival order
  - Convoy effect: short process stuck waiting for long process
  - Also called head of the line blocking

Shortest Job First (SJF)

- Priorizar processos com o menor tempo de processamento

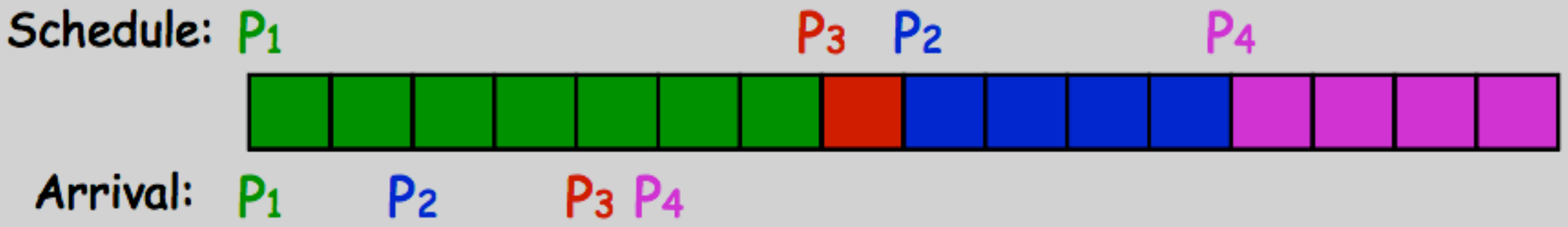
- Schedule the process with the shortest time
- FCFS if same time

- Também não-preemptivo.

Exemplo de SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Gantt chart



- Average waiting time: (0 + 6 + 3 + 7)/4 = 4

Resumo do SJF

- Schedule the process with the shortest time
  - FCFS if same time
- Advantages
  - Minimizes average wait time. Provably optimal
- Disadvantages
  - Not practical: difficult to predict burst time
    - Possible: past predicts future
  - May starve long jobs

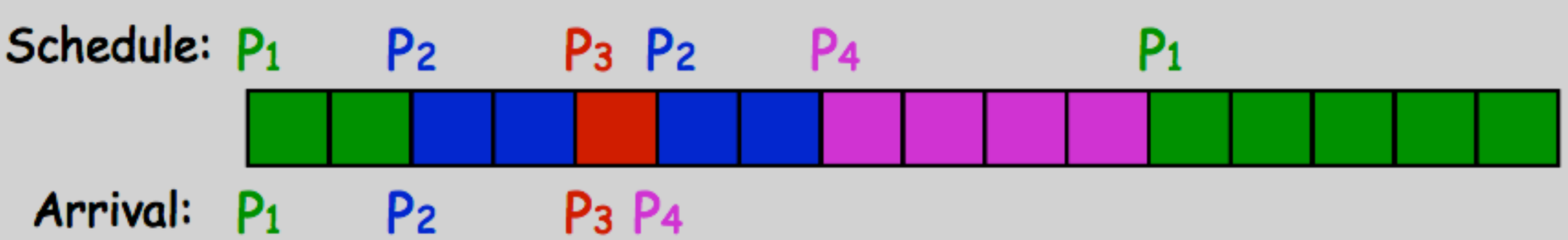
Shortest Remaining Time First (SRTF)

- If new process arrives w/ shorter CPU burst than the remaining for current process, schedule new process
  - SJF with preemption

- Advantage: reduces average waiting time

Exemplo de SRTF

- Gantt chart



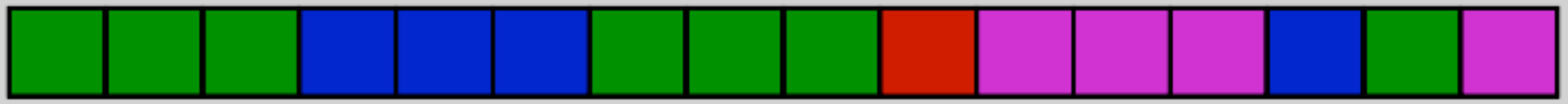
- Average waiting time: (9 + 1 + 0 + 2)/4 = 3

Round Robin (RR)

- Practical approach to support time-sharing
- Run process for a time slice, then move to back of FIFO queue
- Preempted if still running at end of time-slice
- How to determine time slice?

Exemplo de RR: time slice=3

- Gantt chart with time slice = 3



Arrival: P_1 P_2 P_3 P_4

Queue:

- Average waiting time: (8 + 8 + 5 + 7)/4 = 7
- Average response time: (0 + 1 + 5 + 5)/4 = 2.75
- # of context switches: 7

Exemplo de RR: time slice=1

- Gantt chart with time slice = 1

Arrival: P_1 P_2 P_3 P_4

Queue:

P_1	P_1	P_2	P_1	P_2	P_3	P_1	P_4	P_2	P_1	P_4	P_2	P_1	P_4	P_1	P_4
		P_1	P_2	P_3	P_1	P_4	P_2	P_1	P_4	P_2	P_1	P_4	P_1	P_4	
				P_1	P_4	P_2	P_1	P_4	P_2	P_1	P_4				
					P_2										

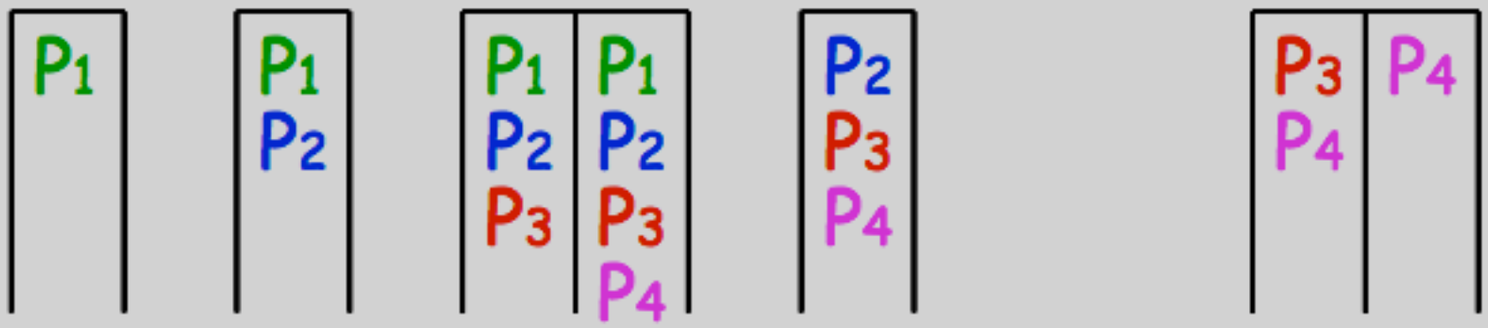
- Average waiting time: (8 + 6 + 1 + 7)/4 = 5.5
- Average response time: (0 + 0 + 1 + 2)/4 = 0.75
- # of context switches: 14

Exemplo de RR: time slice=10

- Gantt chart with time slice = 10

Arrival: P_1 P_2 P_3 P_4

Queue:



- Average waiting time: (0 + 5 + 7 + 7)/4 = 4.75
- Average response time: same
- # of context switches: 3 (minimum)

Vantagens/desvantagens do RR

- Advantages
  - Low response time, good interactivity
  - Fair allocation of CPU across processes
  - Low average waiting time when job lengths vary widely

- Disadvantages
  - Poor average waiting time when jobs have similar lengths
    - Average waiting time is even worse than FCFS!
  - Performance depends on length of time slice
    - Too high → degenerate to FCFS
    - Too low → too many context switches, costly

Adicionando prioridades

- A priority is associated with each process
  - Run highest priority ready job (some may be blocked)
  - Round-robin among processes of equal priority
  - Can be preemptive or nonpreemptive

- Representing priorities
  - Typically an integer
  - The larger the higher or the lower?

Atribuindo prioridades

- Priority can be statically assigned
  - Some always have higher priority than others
  - Problem: starvation
- Priority can be dynamically changed by OS
  - Aging: increase the priority of processes that wait in the ready queue for a long time

Escalonador genérico

- No one-size-fits-all scheduler
  - Different workloads
  - Different environment

- Building a general scheduler that works well for all is difficult!

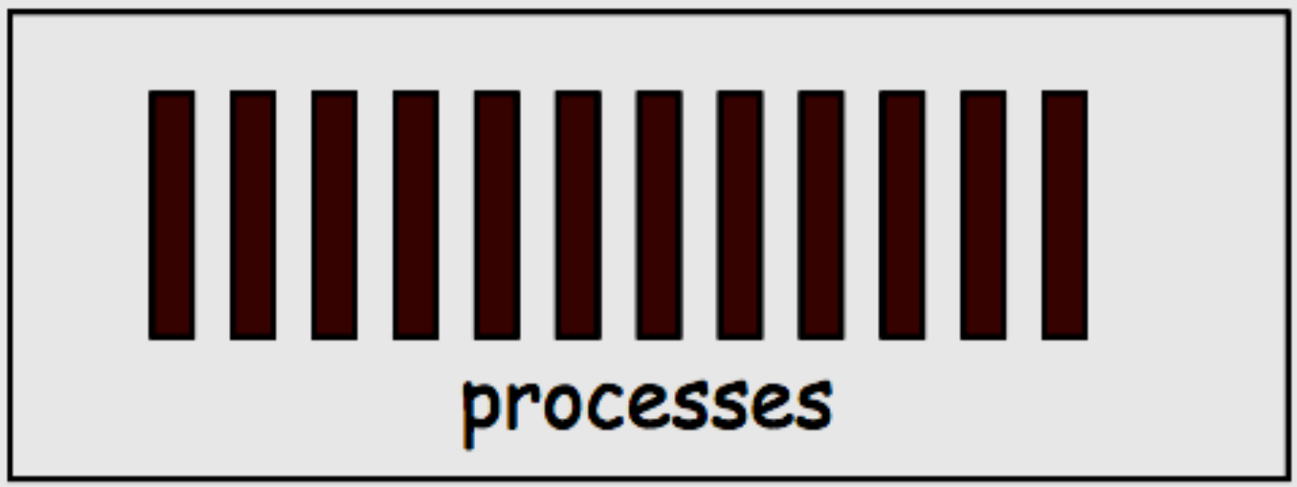
- Real scheduling algorithms are often more complex than the simple scheduling algorithms we've seen

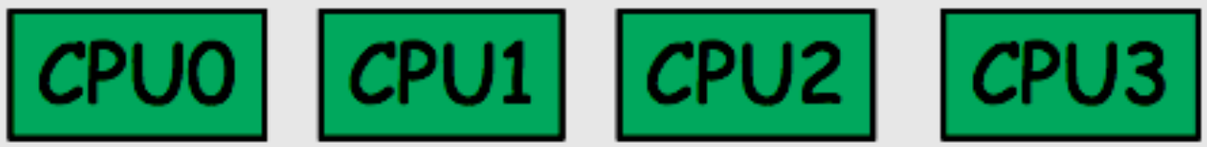
Combinando algoritmos de escalonamento

- Multilevel queue scheduling: ready queue is partitioned into multiple queues
- Each queue has its own scheduling algorithm
  - Foreground processes: RR
  - Background processes: FCFS
- Must choose scheduling algorithm to schedule between queues. Possible algorithms
  - RR between queues
  - Fixed priority for each queue

Sistemas multiprocessados

- Shared-memory Multiprocessor

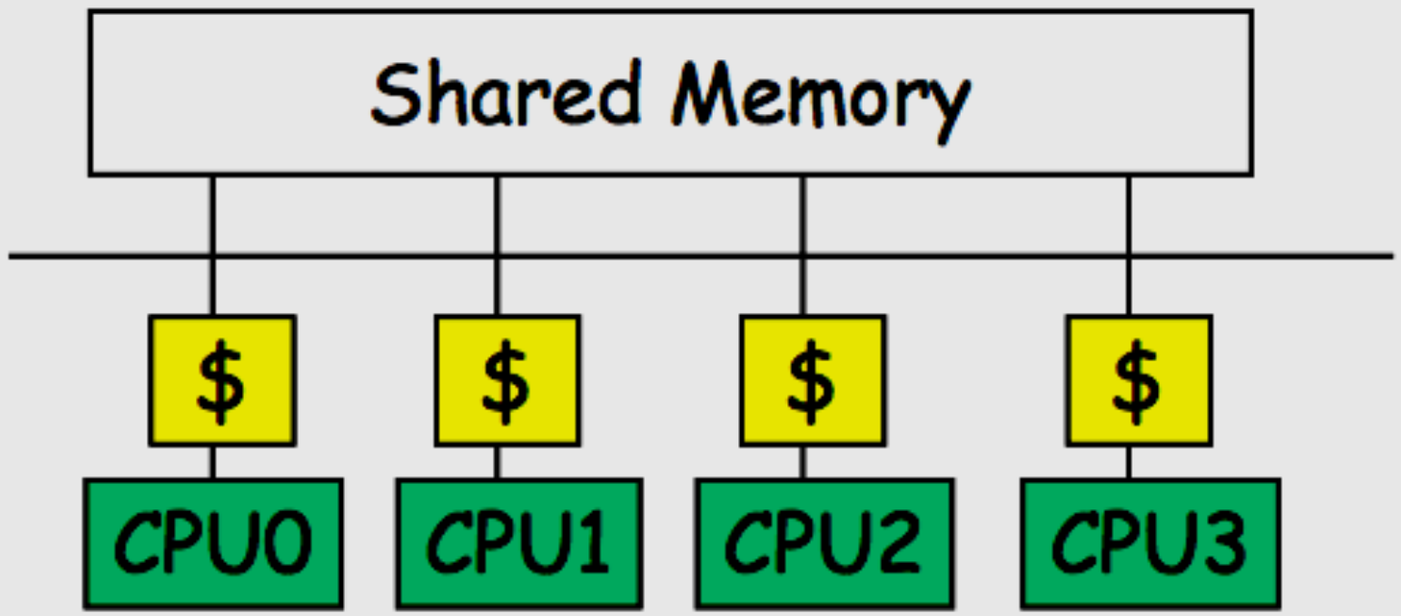




- How to allocate processes to CPU?

Multiprocessamento simétrico

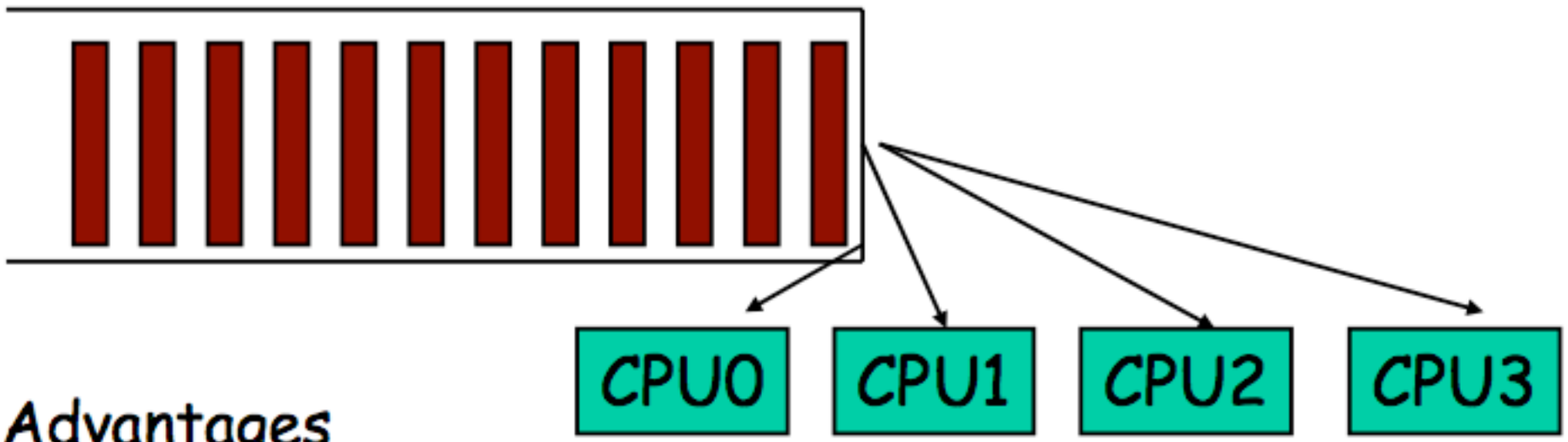
- Architecture



- Small number of CPUs
- Same access time to main memory
- Private cache

Fila global de processos

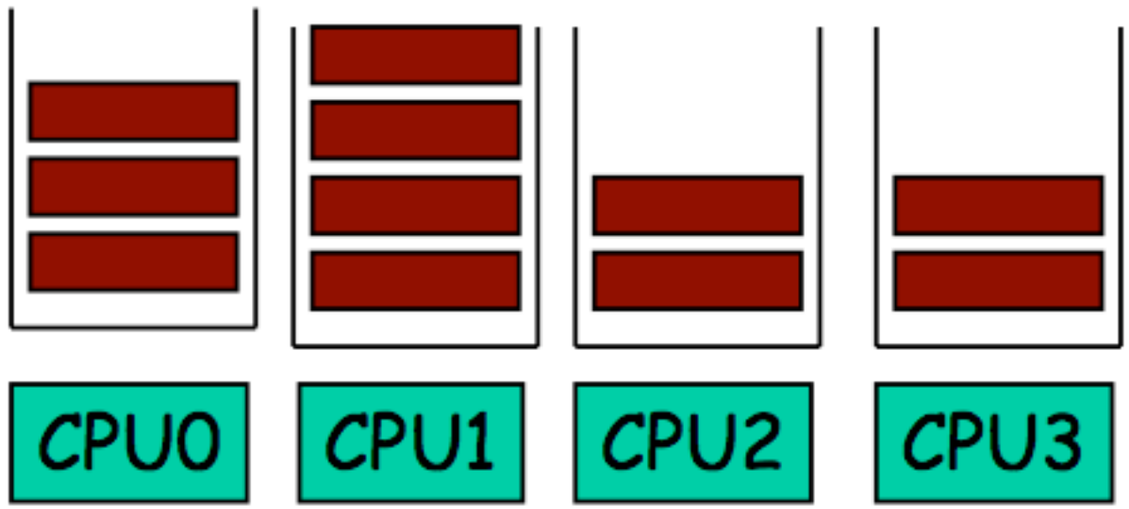
- One ready queue shared across all CPUs



- Advantages
  - Good CPU utilization
  - Fair to all processes
- Disadvantages
  - Not scalable (contention for global queue lock)
  - Poor cache locality
- Linux 2.4 uses global queue

Fila de processos por CPU

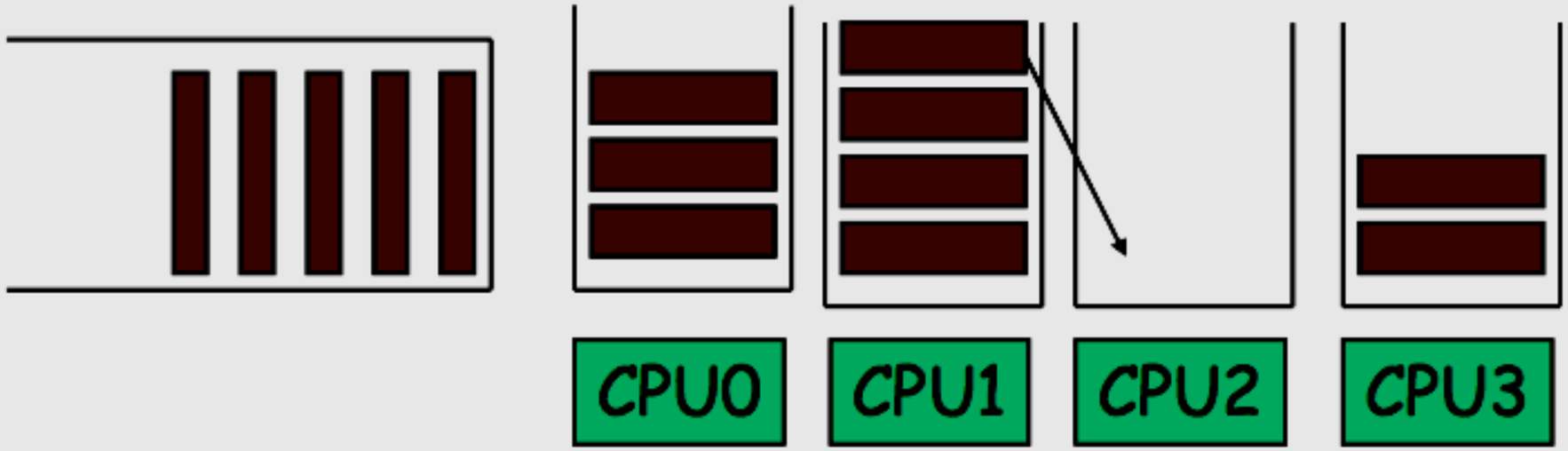
- Static partition of processes to CPUs



- Advantages
  - Easy to implement
  - Scalable (no contention on ready queue)
  - Better cache locality
- Disadvantages
  - Load-imbalance (some CPUs have more processes)
    - Unfair to processes and lower CPU utilization

Abordagem híbrida

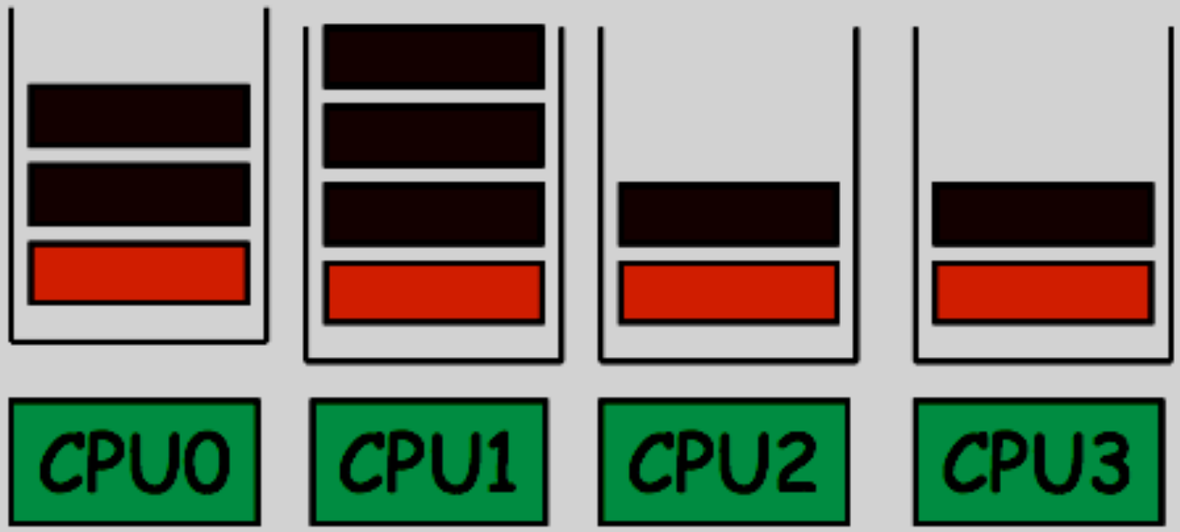
- Use both global and per-CPU queues
- Balance jobs across queues



- Processor Affinity
  - Add process to a CPU's queue if recently run on the CPU
    - Cache state may still present
- Linux 2.6 uses a very similar approach

Coescalonamento

- Multiple processes need coordination
- Should be scheduled simultaneously



- Scheduler on each CPU does not act independently
- Coscheduling (gang scheduling): run a set of processes simultaneously
- Global context-switch across all CPUs

Escalonamento em tempo real

- Real-time processes have timing constraints
  - Expressed as deadlines or rate requirements
  - E.g. gaming, video/music player, autopilot…

- Hard real-time systems – required to complete a critical task within a guaranteed amount of time
- Soft real-time computing – requires that critical processes receive priority over less fortunate ones

- Linux supports soft real-time

Escalonamento em Linux

- Avoid starvation
- Boost interactivity
  - Fast response to user despite high load
  - Achieved by inferring interactive processes and dynamically increasing their priorities
- Scale well with number of processes
  - O(1) scheduling overhead
- SMP goals
  - Scale well with number of processors
  - Load balance: no CPU should be idle if there is work
  - CPU affinity: no random bouncing of processes

Visão geral

- Multilevel Queue Scheduler
  - Each queue associated with a priority
  - A process's priority may be adjusted dynamically
- Two classes of processes
  - Real-time processes: always schedule highest priority processes
    - FCFS (SCHED_FIFO) or RR (SCHED_RR) for processes with same priority
  - Normal processes: priority with aging
    - RR for processes with same priority (SCHED_NORMAL)
    - Aging is implemented efficiently

Particionamento de prioridades

- Total 140 priorities [0, 140)
  - Smaller integer = higher priority
  - Real-time: [0,100)
  - Normal: [100, 140)

- MAX_PRIO and MAX_RT_PRIO
  - include/linux/sched.h

Tarefa

- Linux
  - A partir do kernel 2.6.23, aboliu-se o sistema de múltiplas filas de escalonamento
    - Como funciona então o escalonamento nos últimos kernels de Linux ?
      - Qual a diferença em relação à tecnica multilevel ?
- Poste sua resposta no fórum