

Chapter 5: Escalonamento de CPU



Chapter 5: CPU Scheduling

- ❑ Conceitos Basicos
- ❑ Criterios para Escalonamento
- ❑ Algoritmos de Escalonamento
- ❑ Escalonamento de multiplos-processadores
- ❑ Escalonamento de Tempo Real
- ❑ Escalonamento de Thread
- ❑ Exemplos de SOs
- ❑ Escalonamento de Thread Java
- ❑ Avaliacao de Algoritmo

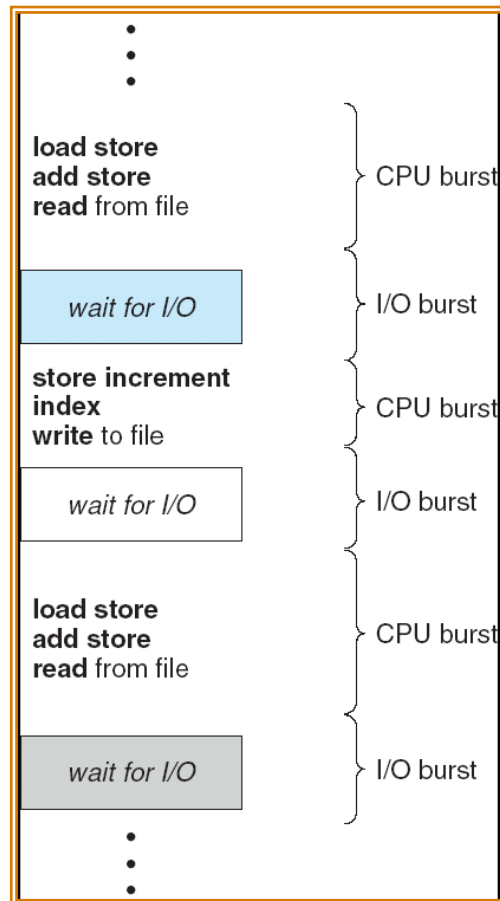


Conceitos Basicos

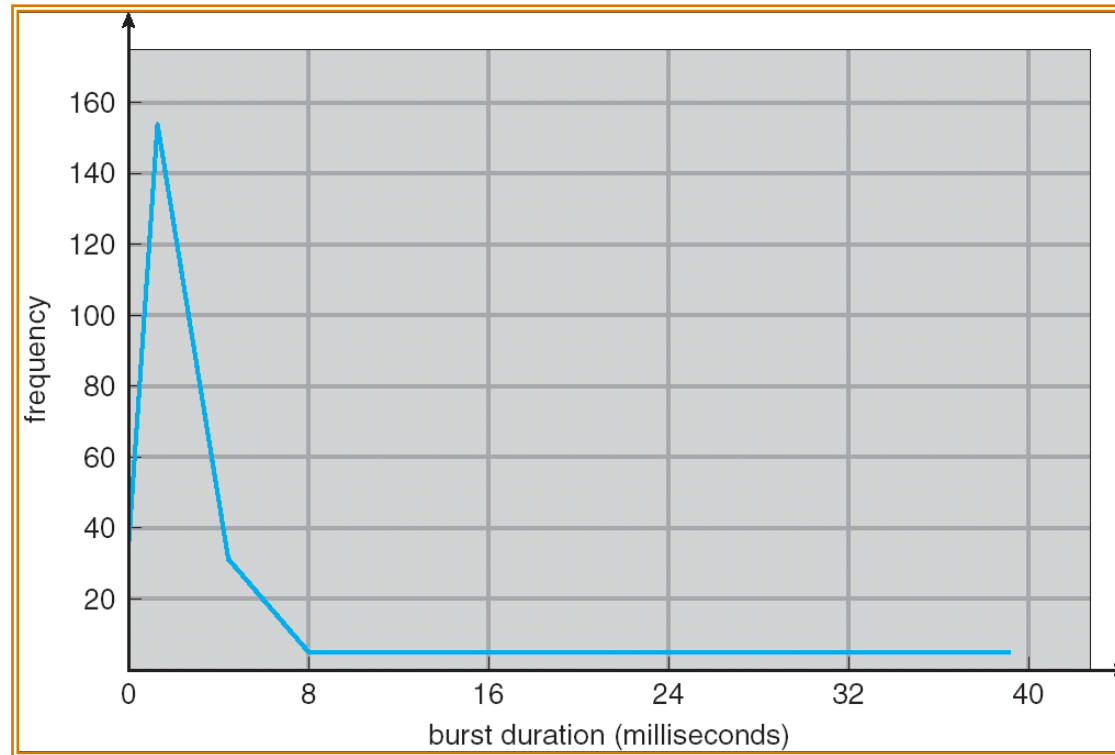
- ❑ Utilizacao maxima de CPU obtida com a multiprogramacao
- ❑ Ciclo de CPU–I/O Burst – Execucao de processo consiste de um ciclo de execucao de CPU e espera por E/S
- ❑ Distribuicao de CPU burst



Alternando sequencia de CPU e I/O Bursts



Histograma de tempos de CPU-burst



Escalonador de CPU

- ❑ Selecciona processos na memoria que estao prontos para executar
- ❑ Escalonador é chamado quando:
 1. Muda de estado running para waiting
 2. Muda de estado running para ready
 3. Muda de waiting para ready
 4. Termina um processo
- ❑ Escalonadores na condicoes 1 e 4 sao nao-preemptivos
- ❑ Condicoes 2 e 3 sao preemptivas



Expedidor (Dispatcher)

- ❑ Este modulo fornece o controle da CPU par o processo selecionado (escalonador de curto-prazo), isto envolve:
 - Mudar de contexto
 - Mudar para modo usuario
 - Ir para localizacao correta do programa do usuario para reinicia-lo
- ❑ *Latencia do Expedidor* – tempo levado para o expedidor parar um processo e comecar outro



Criterio para Escalonamento

- ❑ Utilizacao de CPU – manter a CPU mais ocupada possivel
- ❑ Throughput – numero de processos executado por unidade de tempo
- ❑ Tempo de Turnaround– quantidade de tempo para executar um processo particular
- ❑ Tempo de espera (Waiting time) – quantidade de tempo de espera de um processo na ready queue
- ❑ Tempo de Resposta (response time) – quantidade de tempo levada a partir da submissao da requisicao até a primeira resposta produzida (para ambiente time-sharing)



Criterio de Otimizacao

- ❑ Maxima utilizacao da CPU
- ❑ Maximo throughput
- ❑ Minimo tempo de turnaround
- ❑ Minimo tempo de espera
- ❑ Minimo tempo de resposta



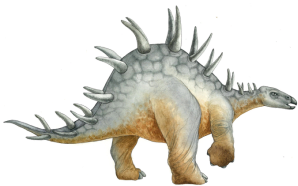
First-Come, First-Served (FCFS) Scheduling

<u>Processo</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- Suponha que projetos cheguem na ordem: P_1 , P_2 , P_3
O grafico de Gantt para o escalonamento é:



- Waiting time para $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Média de waiting time: $(0 + 24 + 27)/3 = 17$

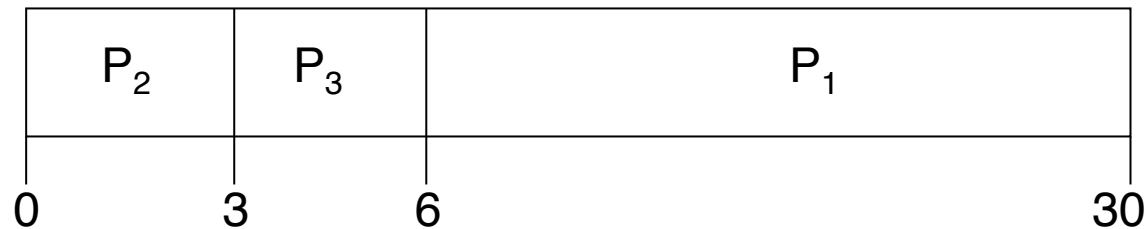


FCFS Scheduling (Cont.)

Suponha que processos cheguem nesta ordem

$$P_2, P_3, P_1$$

- O grafico de Gantt para o escalonamento é::



- Waiting time para $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Média waiting time: $(6 + 0 + 3)/3 = 3$
- Melhor que o caso anterior
- *Efeito comboio: processos pequenos aguardam processos grandes*



Shortest-Job-First (SJF) Scheduling

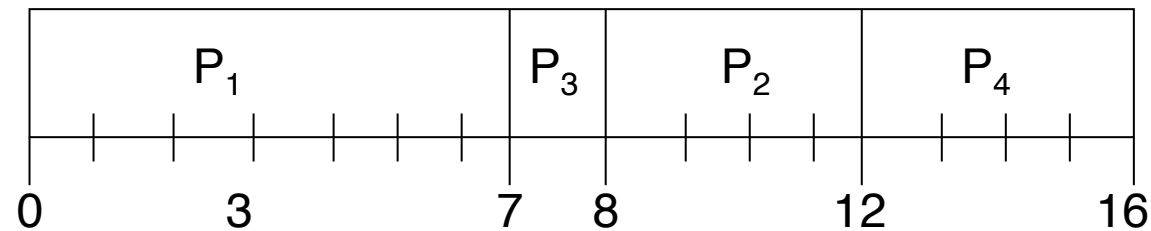
- ❑ Associa cada processo o tempo do proximo CPU burst. Escolhe o processo com menor tempo
- ❑ Dois esquemas:
 - Não preemptivo – uma vez o processo na CPU nao pode substitui-lo enquanto nao completar seu CPU burst
 - preemptivo – se um novo processo chega com CPU burst menor do que o processo em execucao, troca o processo. Este esquema é conhecido como Shortest-Remaining-Time-First (SRTF)
- ❑ SJF é ótimo – resulta no media minima de um conjunto de processos



Exemplo de SJF nao preemptivo

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



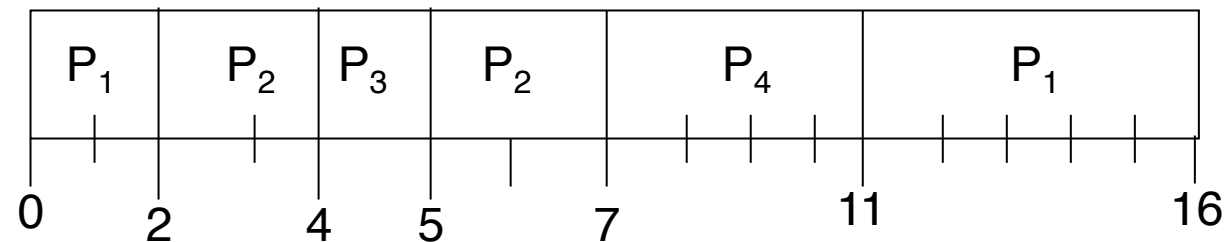
- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$



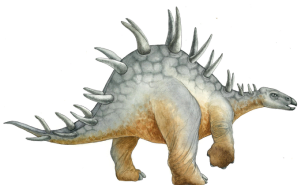
Example SJF Preemptivo

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

■ SJF (preemptive)



■ Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$



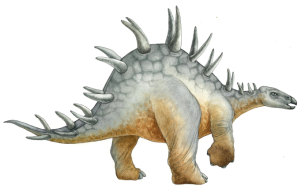
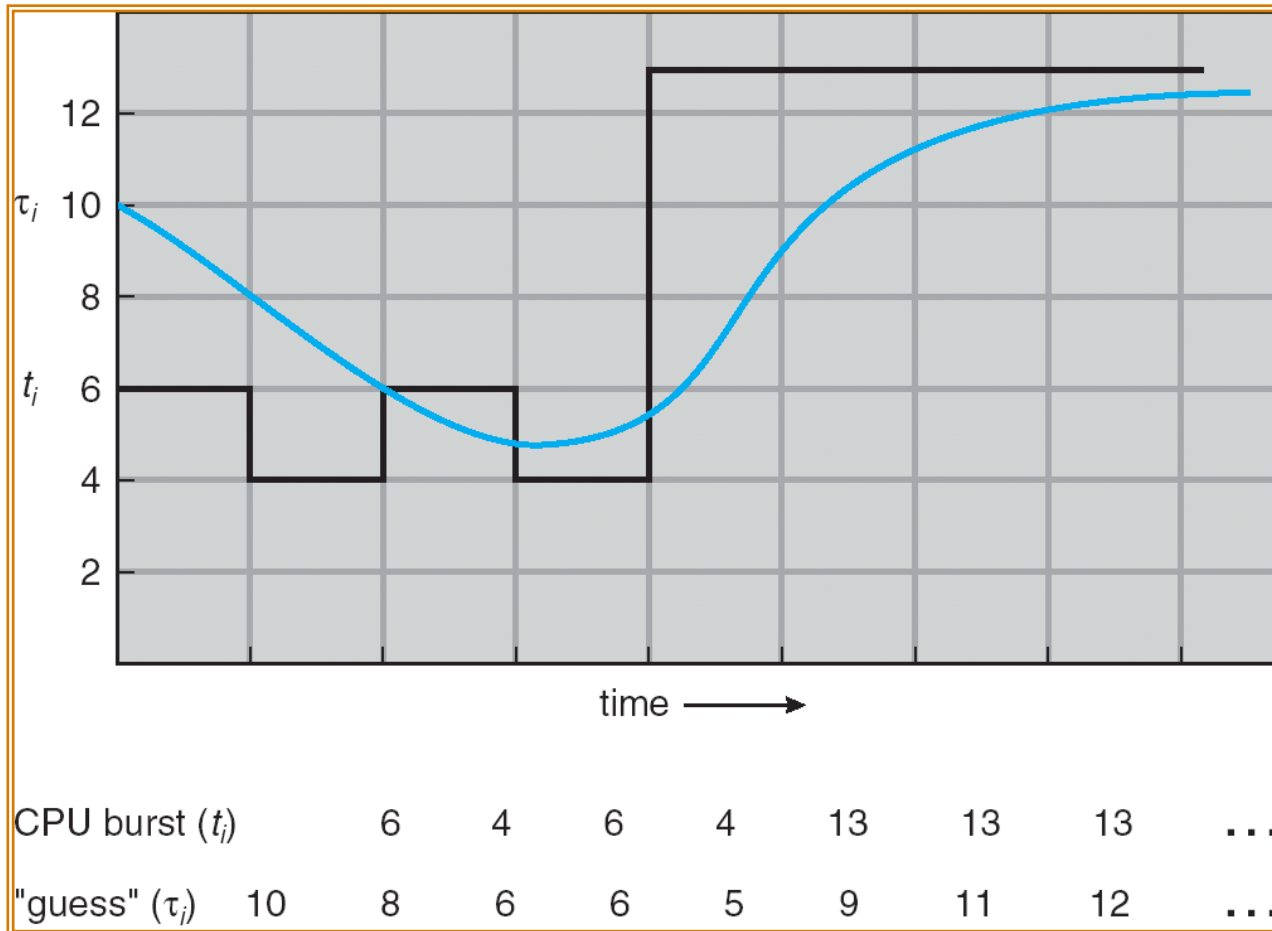
Determinando o tamanho do proximo CPU Burst

- Apenas estimado
- Pode ser definido usando o tamanho dos CPU burts anteriores

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n.$$



Predicao do tamanho do proximo CPU Burst

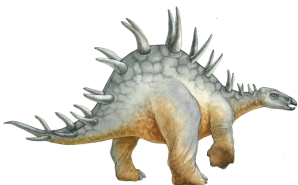


Exemplos de calculo de media exponencial

- $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Historico recente nao conta
- $\alpha = 1$
 - $\tau_{n+1} = \alpha t_n$
 - Somente o ultimo CPU burst conta
- Se expandirmos a formula, teremos:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Visto que ambos α e $(1 - \alpha)$ são menores ou iguais a 1, cada termo sucessivo é menor do que o predecessor



Priority Scheduling

- ❑ Um numero de prioridade é associado a cada processo
- ❑ A CPU é alocada com o processo de mais alta prioridade (menor inteiro \equiv mais alta prioridade)
 - Preemptivo
 - Nao preemptivo
- ❑ SJF é um priority scheduling onde a prioridade é o proximo CPU burst previsto
- ❑ Problema \equiv Starvation – processos de baixa prioridade podem nunca executar
- ❑ Solução \equiv Aging – de acordo com o progresso do tempo incrementar a prioridade do processo



Round Robin (RR)

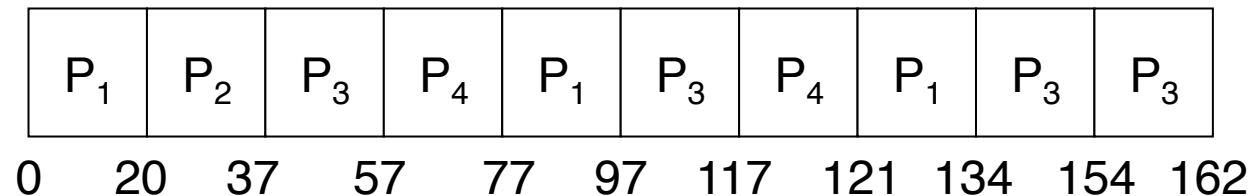
- ❑ Cada processo recebe uma fracção de tempo da CPU (*time quantum*), usualmente de 10-100 milissegundos.
- ❑ Depois deste processo, o processo é retirado e colocado no final da ready queue.
- ❑ Cada processo recebe $1/n$ tempo da CPU (n processos) em pedaços de *time quantum* q
- ❑ Nenhum processo espera mais do que $(n-1)q$ unidades de tempo
- ❑ Performance
 - q grande \Rightarrow FIFO
 - q pequeno \Rightarrow q deve ser maior do que o tempo de troca de contexto, caso contrário o overhead é muito grande



Exemplo de RR com Time Quantum = 20

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	17
P_3	68
P_4	24

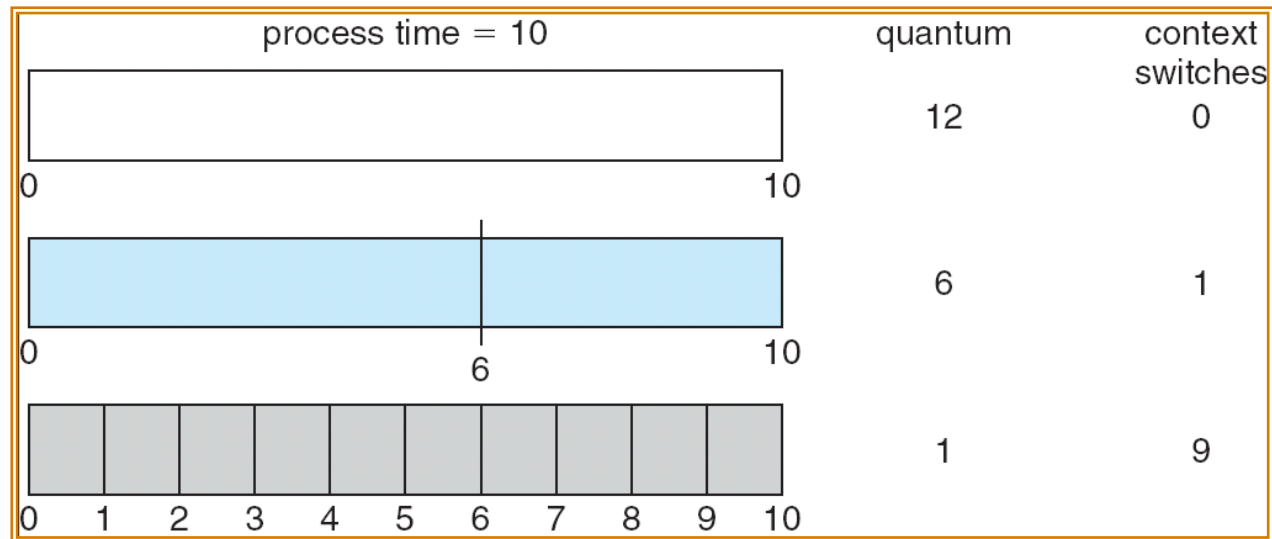
□ The Gantt chart is:



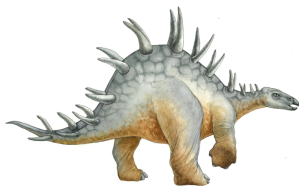
□ Media mais alta de turnaround do que SJF, mas melhor tempo de resposta



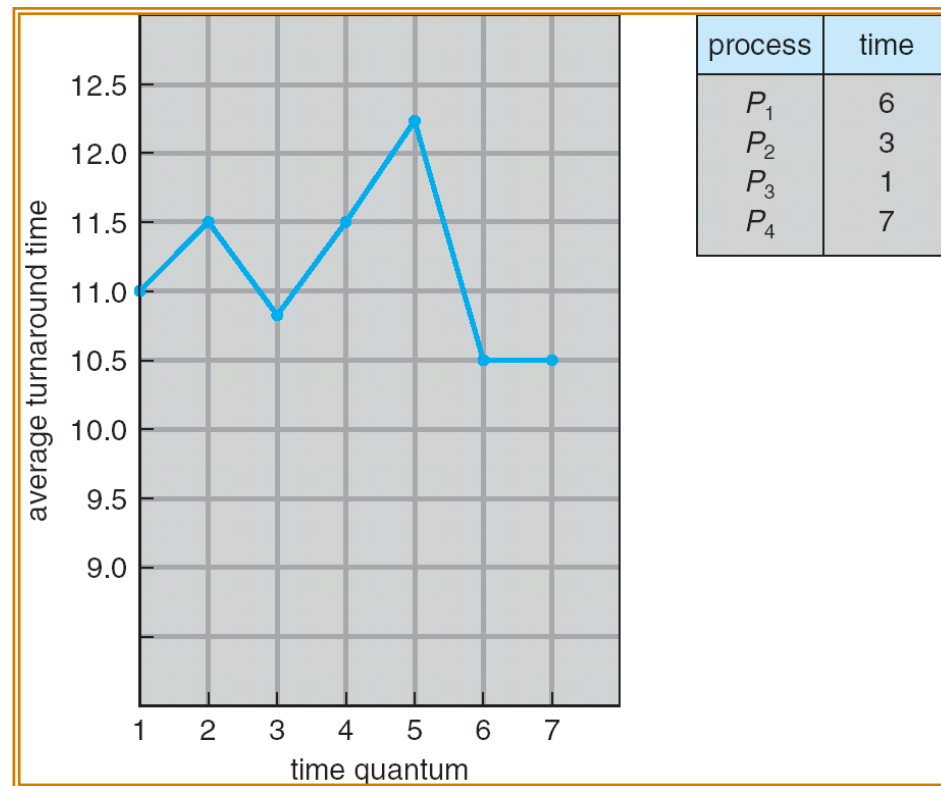
Time Quantum e tempo para troca de contexto



Exercicio (Turnaround x Time Quantum)



Tempo de Turnaround Time varia com oTime Quantum

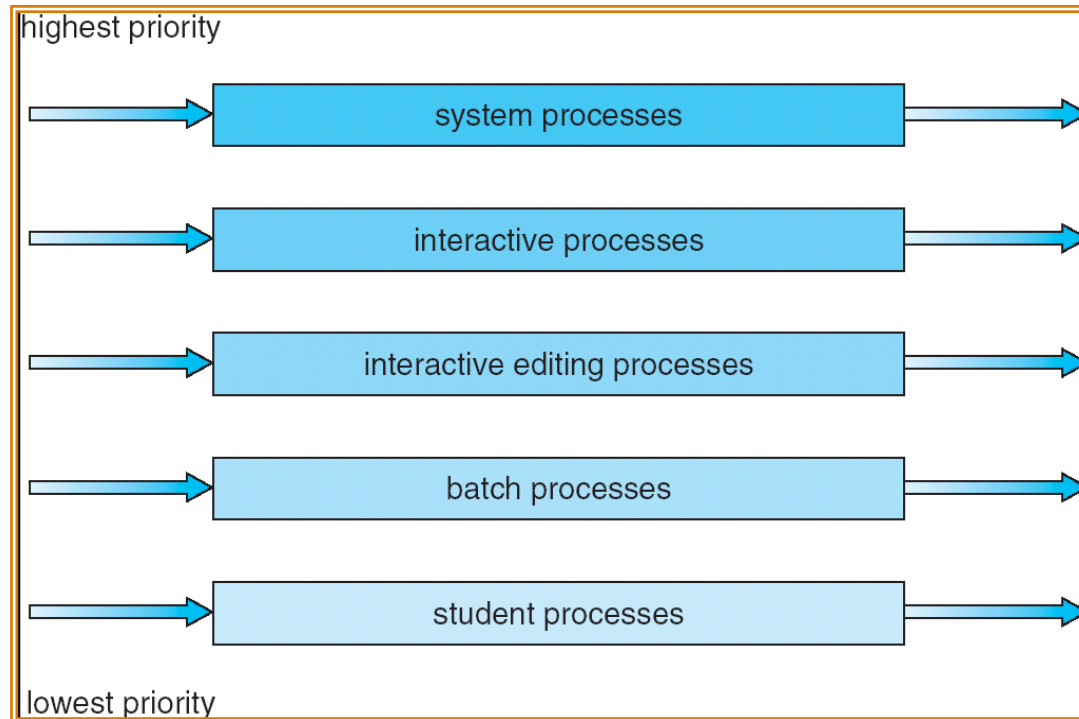


Diversos níveis de filas

- ❑ Ready queue é particionada em filas separadas:
foreground (interativa)
background (batch)
- ❑ Cada Fila em seu algoritmo de escalonamento
 - foreground – RR
 - background – FCFS
- ❑ Escalonamento pode ser feito entre filas
 - Escalonamento com prioridade fixa; (i.e., serve a todos do foreground e depois do background). Possibilidades de starvation.
 - Time slice – cada fila obtém um tempo de CPU; i.e., 80% para foreground em RR, 20% para background em FCFS



Multilevel Queue Scheduling



Multilevel Feedback Queue

- ❑ Um processo pode mover entre filas; mecanismo de aging pode ser implementado desta forma
- ❑ Separar processos de acordo com características de CPU burst
- ❑ Escalonamento Multilevel-feedback-queue definido pelos seguintes parametros:
 - Numero de filas
 - Algoritmo de escalonamento para cada fila
 - Método usado para determinar quando fazer upgrade do processo
 - Método para definir quando rebaixar processo
 - Método para determinar em qual fila o processo entra quando processo iniciar

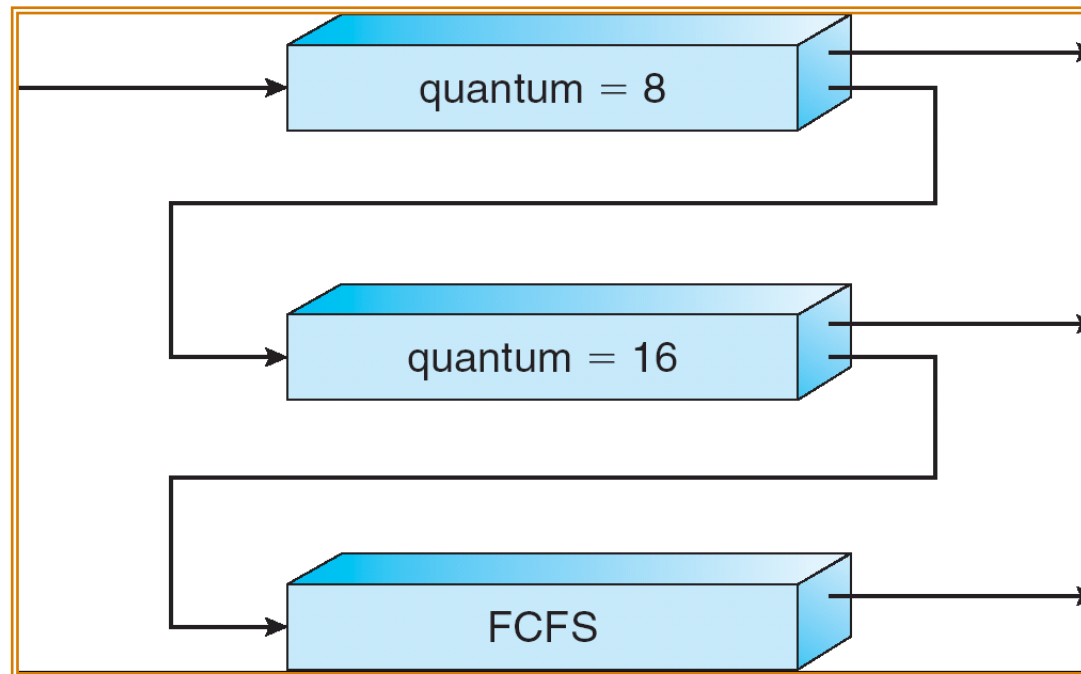


Exemplo de Multilevel Feedback Queue

- Three queues:
 - Q_0 – RR with time quantum 8 milliseconds
 - Q_1 – RR time quantum 16 milliseconds
 - Q_2 – FCFS
- Scheduling
 - A new job enters queue Q_0 which is served RR. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
 - At Q_1 job is again served RR and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .



Filas Multilevel Feedback



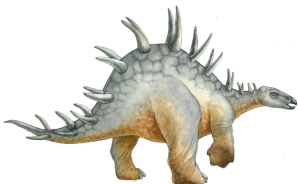
Escalonamento com varios processadores

- ❑ *Compartilhamento de Carga*
- ❑ *Escalonamento mais complexo*
- ❑ *Processadores homogeneos*
- ❑ *Multiprocessamento Assimetrico – somente um processador acessa estrutura de dados do SO, evitando a necessidade de compartilhamento de dados*
- ❑ *Multiprocessamento Simetrico – cada processador pode ser escalonado separadamente(uma unica fila compartilhada ou varias filas) - SOs modernos adotam esta estrategia*



Questoes relacionados com multiplos processadores

- ❑ Afinidade com processador:
 - Afinidade Flexivel
 - Afinidade Rigida
- ❑ Balanceamento de Carga
 - Manter a carga de trabalho distribuida igualmente entre todos processadores de um sistema SMP
 - Somente necessaria em ambientes onde cada processador tem sua propria fila de processos a executar
 - Duas abordagens:
 - ❑ Pushing Migration
 - ❑ Pull Migration



Escalonamento de Thread

- ❑ Multithreading simetrico (SMT):
 - Criar multiplos processadores logicos no mesmo processador fisico
 - Provida pelo hardware e nao pelo software
 - SOs podem aproveitar tais vantagens caso tenham ciencia desta implementaçã
- ❑ Escalonamento Local (PCS:process-contention scope) – Como a biblioteca de thread decide qual thread colocar em qual LWP (lightweight process)
- ❑ Escalonamento Global (SCS: system-contention scope) – Como o kernel decide qual thread do kernel rodar em seguida



API de Escalonamento Pthread

```
int main(int argc, char *argv[])
{
    int i, scope;
    pthread_t tid[NUM_THREADS];
    pthread_attr_t attr;

    /* get the default attributes */
    pthread_attr_init(&attr);

    /* first inquire on the current scope */
    if (pthread_attr_getscope(&attr, &scope) != 0)
        fprintf(stderr, "Unable to get scheduling scope\n");
    else {
        if (scope == PTHREAD_SCOPE_PROCESS)
            printf("PTHREAD_SCOPE_PROCESS");
        else if (scope == PTHREAD_SCOPE_SYSTEM)
            printf("PTHREAD_SCOPE_SYSTEM");
        else
            fprintf(stderr, "Illegal scope value.\n");
    }

    /* set the scheduling algorithm to PCS or SCS */
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

    /* create the threads */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_create(&tid[i], &attr, runner, NULL);

    /* now join on each thread */
    for (i = 0; i < NUM_THREADS; i++)
```



Exemplos de SO

- ❑ Escalonamento Solaris
- ❑ Escalonamento Windows XP
- ❑ Escalonamento Linux



Escalonamento Solaris

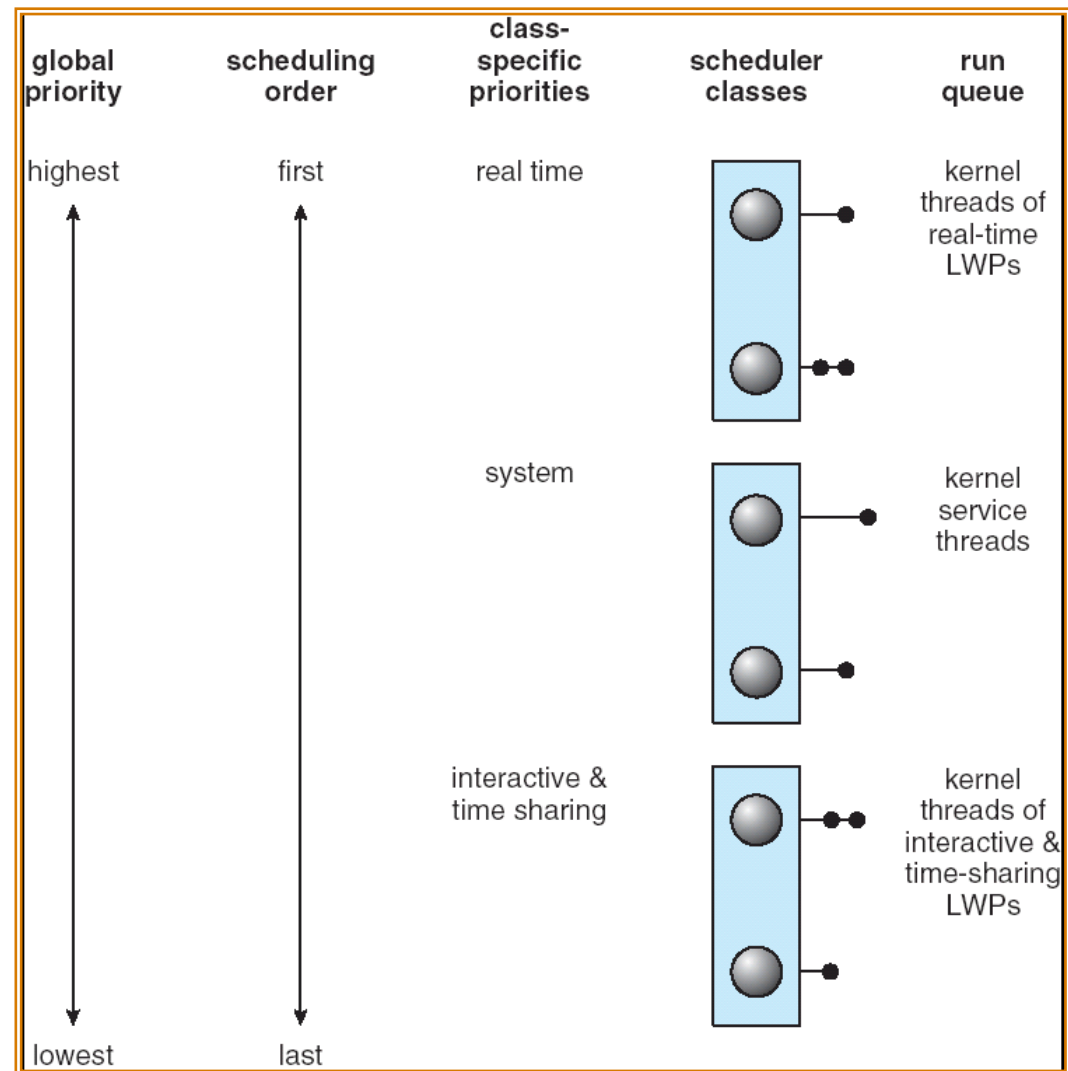


Tabela de Expedição

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59



Prioridades Windows XP

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1



Escalonamento Linux

- ❑ Dois Algoritmos: tempo compartilhado e tempo real
- ❑ Tempo Compartilhado
 - Prioridade baseada em credits
 - Credito subtraído quando uma interrupção ocorre
 - Quando credito = 0, outro processo é escolhido
 - Quando todos os processos tem credito = 0, novos credits são dados
 - ❑ Baseado em fatores que incluem prioridade e histórico
- ❑ Tempo Real
 - Tempo real flexível
 - Compatível com Posix.1b – duas classes
 - ❑ FCFS e RR
 - ❑ Processo de maior prioridade executa primeiro

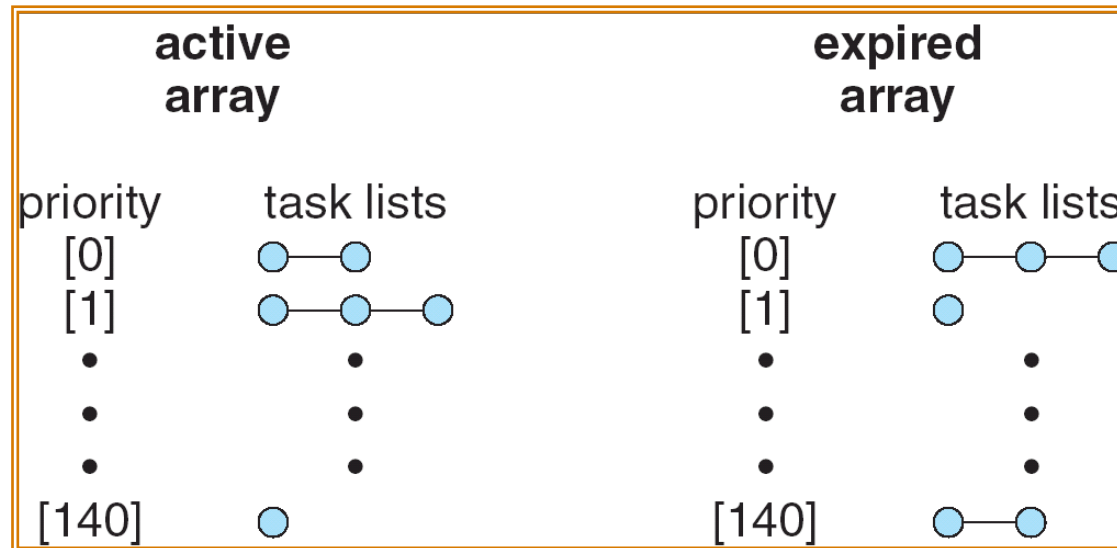


Relacionamento entre Prioridades e tamanho de time quantum

numeric priority	relative priority		time quantum
0	highest	real-time tasks	200 ms
•			
•			
•			
99			
100		other tasks	
•			
•			
•			
140	lowest		10 ms



Lista de tarefas indexadas de acordo com prioridades



Escalonamento Java

❑ Política de escalonamento definida fracamente. A thread executa até:

1. Seu time quantum expirar
2. Ser bloqueado para E/S
3. Terminar o metodo run()

Alguns sistemas podem suportar preempção

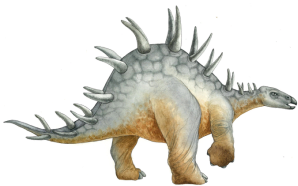


Escalonamento Java

□ Prioridades – valores de 1-10

<u>Priority</u>	<u>Comment</u>
<code>Thread.MIN_PRIORITY</code>	The minimum thread priority
<code>Thread.MAX_PRIORITY</code>	The maximum thread priority
<code>Thread.NORM_PRIORITY</code>	The default thread priority

- `MIN_PRIORITY` é 1
- `NORM_PRIORITY` é 5
- `MAX_PRIORITY` é 10



Escalonamento Java

Alterando a prioridade usando setPriority()

```
public class HighThread implements Runnable
{
    public void run() {
        Thread.currentThread().setPriority(Thread.NORM_PRIORITY + 3);
        // remainder of run() method
        . . .
    }
}
```



Escalonamento Java

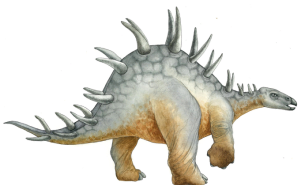
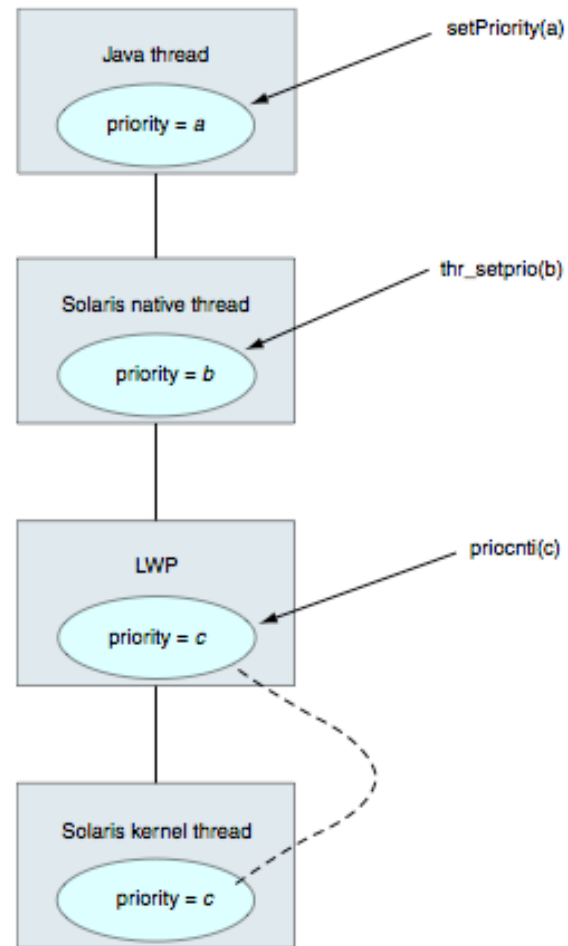
Relacionamento entre prioridades de Java e Win32

Java priority	Win32 priority
1 (MIN_PRIORITY)	LOWEST
2	LOWEST
3	BELOW_NORMAL
4	BELOW_NORMAL
5 (NORM_PRIORITY)	NORMAL
6	ABOVE_NORMAL
7	ABOVE_NORMAL
8	HIGHEST
9	HIGHEST
10 (MAX_PRIORITY)	TIME_CRITICAL



Escalonamento Java

Escalonamento de Java thread scheduling no Solaris



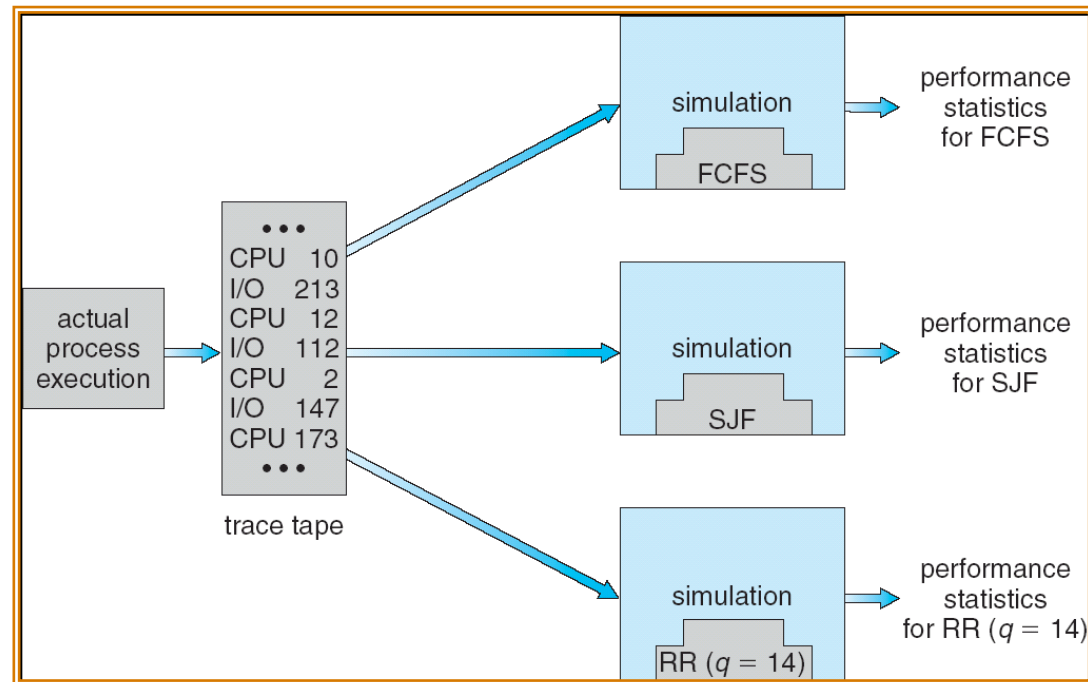
Algoritmo de Avaliação

- ❑ Modelagem Determinística – assume uma carga de trabalho pre-determinada e define a performances de cada algoritmo para aquela carga de trabalho
- ❑ Modelos de Filas:
 - Little's Formula:
 - ❑ $Tamfila = procChegam \times TempoEspera$
- ❑ Simulação:
 - Criar um modelo do sistema
 - Massa de teste
 - ❑ Monitorar o sistema real e gerar massa de teste
- ❑ Implementação
 - Avaliar o sistema real (condicoes reais)
 - Alto custo

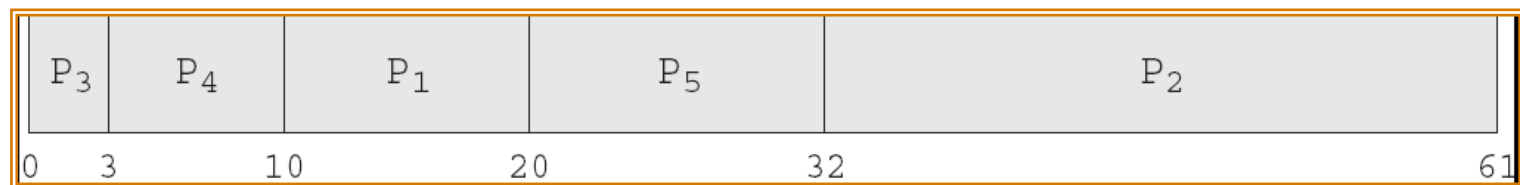
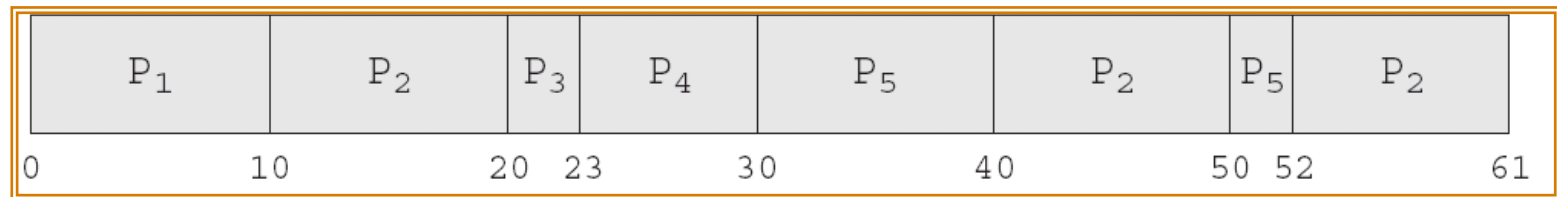
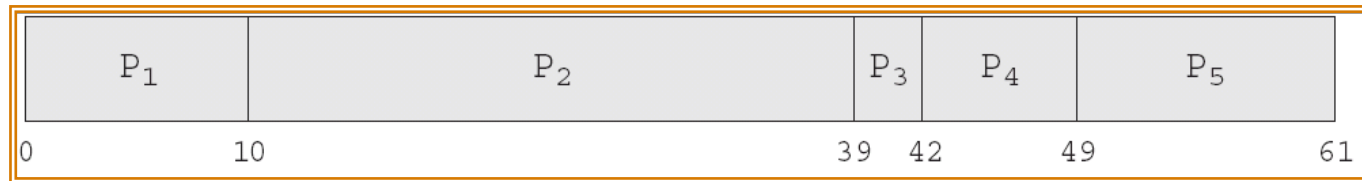


Algoritmo de Avaliação

Avaliação do escalonamento da CPU por simulação



Algoritmo de Avaliação



Fim do Capitulo 5

