Capítulo 5: Escalonamento de CPU

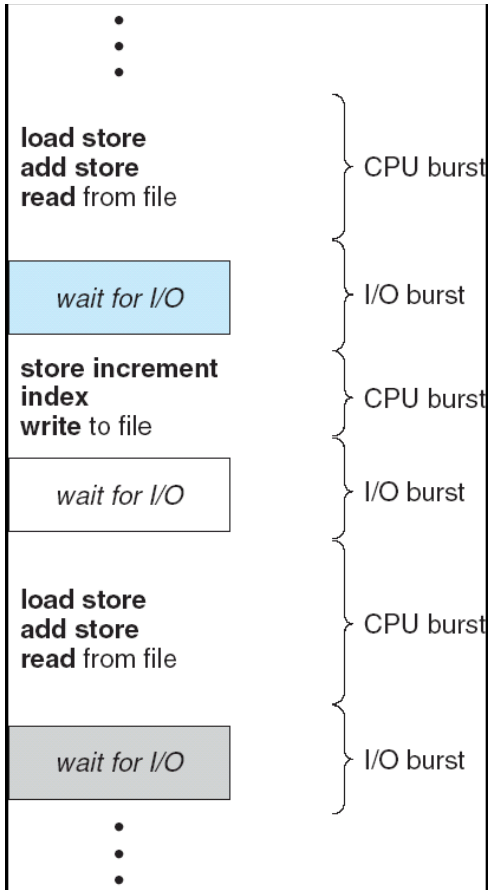
Capítulo 5: Escalonamento de CPU

- Conceitos básicos
- Critérios de escalonamento
- Algoritmos de escalonamento
- Escalonamento em múltiplos processadores
- Escalonamento em tempo real
- Escalonamento de threads
- Exemplos de sistemas operacionais
- Escalonamento de thread em Java
- Avaliação de algoritmo

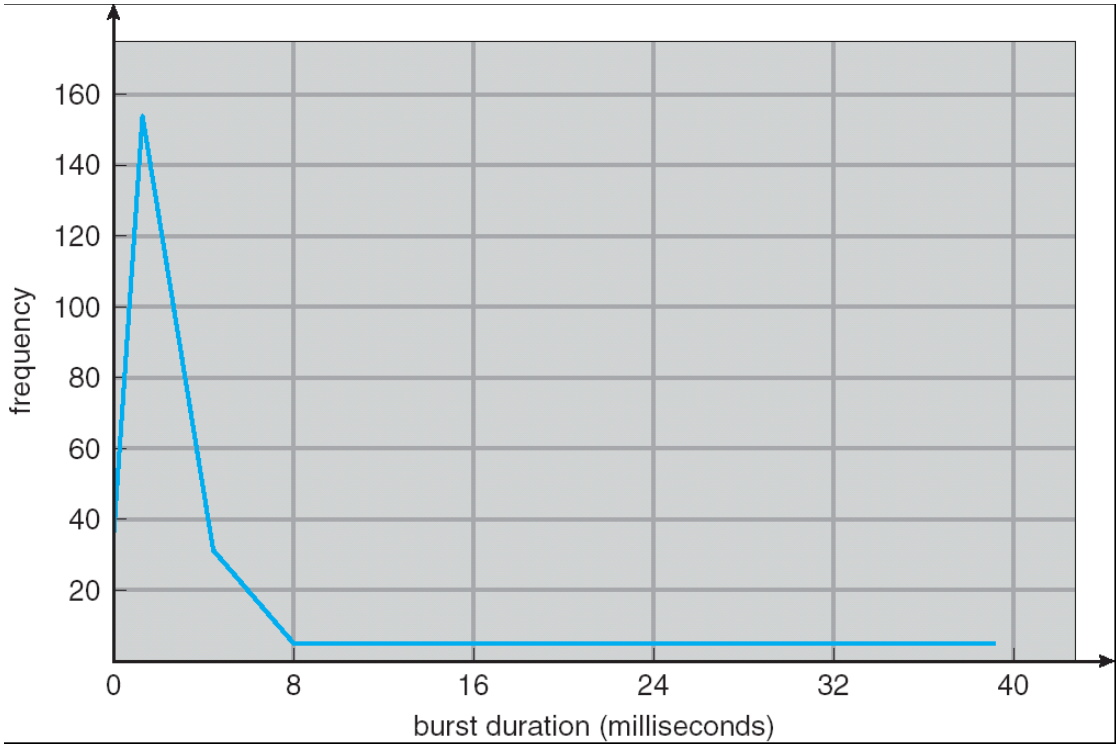
Conceitos básicos

- Utilização máxima de CPU obtida com multiprogramação
- *burst* de CPU e *burst* de E/S – Execução do processo consiste em *ciclos* de execução de CPU e espera de E/S
- Distribuição de burst de CPU (imensa maioria são bem curtos)

Seqüência alternada de bursts de CPU e E/S



Histograma de tempos de burst de CPU



Escalonador de CPU

- Seleciona dentre os processos na memória que estejam prontos para executar, e aloca a CPU a um deles
- Decisões de escalonamento de CPU podem ocorrer quando um processo:
  1. Passa do estado executando para esperando
  2. Passa do estado executando para pronto
  3. Passa de esperando para pronto
  4. Termina
- Escalonamento sob 1 e 4 é *não preemptivo*
- Todo o restante é *preemptivo* (“pede para a criança sair da cama elástica para dar a vez para outra”)

Despachante

- O módulo despachante dá o controle da CPU ao processo selecionado pelo escalonamento de curto prazo; isso envolve:
  - troca de contexto
  - troca para o modo usuário
  - salto para o local apropriado no programa do usuário para reiniciar esse programa
- *Latência de despacho* – tempo para que o despachante termine um processo e inicie outro em execução

Critérios de escalonamento

- Utilização de CPU– mantenha a CPU o mais ocupada possível
- Vazão – número de processos que completam sua execução por unidade de tempo
- Tempo de retorno – tempo gasto para executar um processo em particular (espera + E/S + CPU)
- Tempo de espera – tempo em que um processo esteve esperando na fila de prontos
- Tempo de resposta – tempo desde quando uma solicitação foi submetida até a primeira reposta ser produzida, não a saída (para ambiente de tempo compartilhado)

Critérios de otimização

- Utilização máxima de CPU
- Vazão máxima
- Tempo de retorno mínimo
- Tempo de espera mínimo
- Tempo de resposta mínimo

Escalonamento First-Come, First-Served (FCFS)

Processo	Tempo de burst
P_1	24
P_2	3
P_3	3

- Suponha que os processos cheguem nesta ordem: P_1 , P_2 , P_3

  O Gráfico de Gantt para o escalonamento FCFS é:

- Tempo de espera para P_1 = 0; P_2 = 24; P_3 = 27
- Tempo de espera médio: (0 + 24 + 27)/3 = 17

Escalonamento FCFS (cont.)

Suponha que os processos cheguem nesta ordem:

P_2 , P_3 , P_1

- O Gráfico de Gantt será:

- Tempo de espera para $P_1 = 6$; $P_2 = 0$, $P_3 = 3$
- Tempo de espera médio: (6 + 0 + 3)/3 = 3
- Muito melhor que o caso anterior
- *Efeito comboio:* processo curto atrás de processo longo

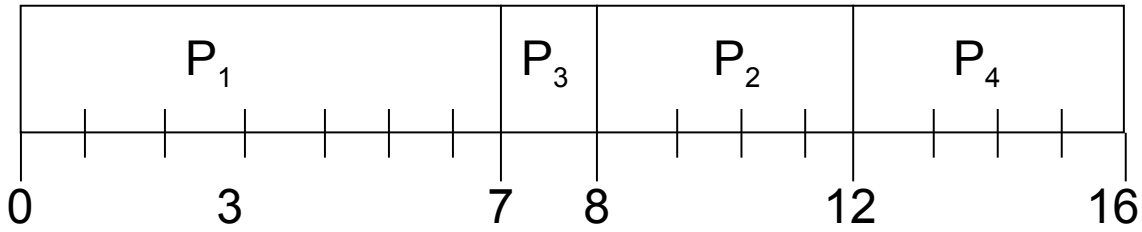
Escalonamento Shortest-Job-First (SJF)

- Associe a cada processo a extensão de seu próximo burst de CPU. Use essas extensões para escalonar o processo com o menor tempo (evita o *efeito comboio*)
- Dois esquemas:
  - não preemptivo – uma vez que a CPU é dada ao processo, ele não pode ser apropriado até que termine seu burst de CPU
  - preemptivo – se um novo processo chega com tamanho de burst de CPU menor que o tempo restante do processo atualmente em execução, apropria. Esse esquema é conhecido como Shortest-Remaining-Time-First (SRTF)
- SJF é **ideal** – gera o menor tempo de espera médio para determinado conjunto de processos

Exemplo de SJF não preemptivo

Processo	Tempo chegada	Tempo de burst
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

- SJF (não preemptivo)

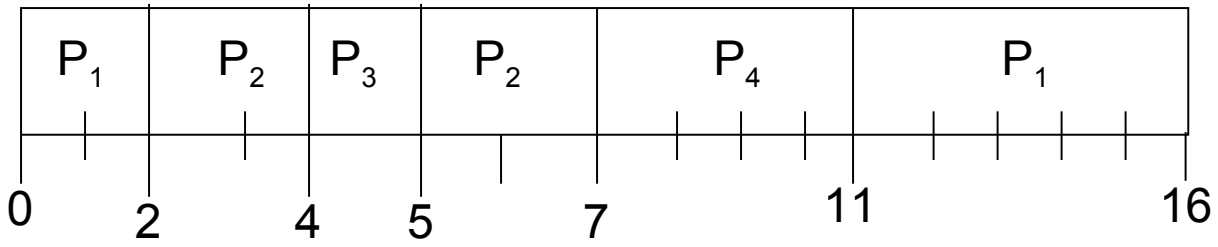


- Tempo médio de espera = (0 + 6 + 3 + 7)/4 = 4

Exemplo de SJF preemptivo (SRTF)

Processo	Tempo chegada	Tempo de burst
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

- SJF preemptivo (SRTF):

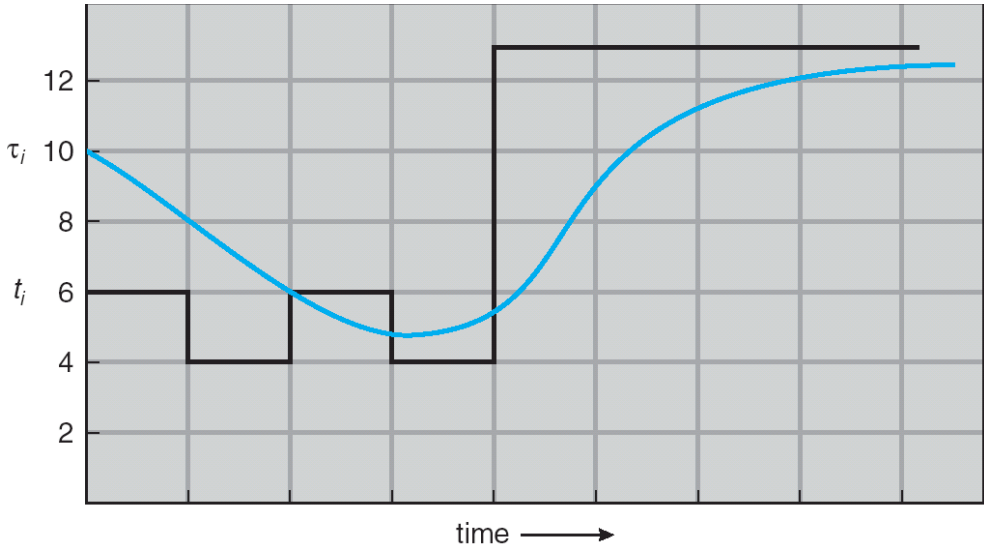


- Tempo médio de espera = (9 + 1 + 0 +2)/4 = 3

Determinando a extensão do próximo burst de CPU

- Só pode estimar a extensão
- Pode ser feito usando a extensão dos bursts de CPU anteriores, usando a média exponencial

Previsão da extensão do próximo burst de CPU



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Escalonamento por prioridade

- Um número de prioridade (inteiro) é associado a cada processo
- A CPU é alocada ao processo com a maior prioridade (menor inteiro ! maior prioridade)
  - Preemptivo
  - não preemptivo
- SJF é um escalonamento por prioridade onde a prioridade é o próximo tempo de burst de CPU previsto
- Problema ! Estagnação – processos com baixa prioridade podem nunca ser executados
- Solução ! Envelhecimento – à medida que o tempo passa, aumenta a prioridade do processo

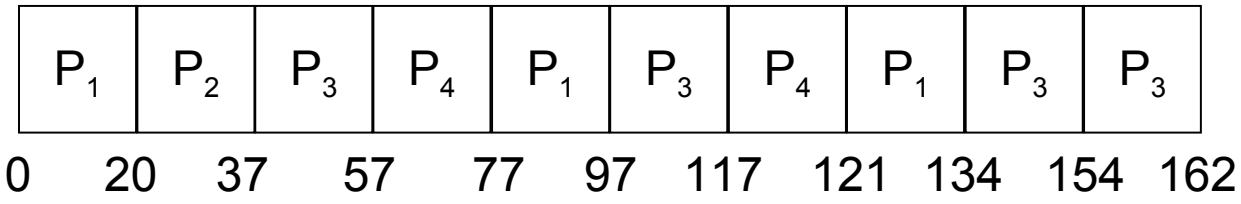
Round Robin (RR)

- Cada processo recebe uma pequena unidade de tempo de CPU (*quantum de tempo*), normalmente 10-100 milissegundos. Depois que esse tempo tiver passado, o processo é apropriado e acrescentado ao final da fila de pronto.
- Se houver *n* processos na fila de pronto e o quantum de tempo for *q*, então cada processo recebe 1/*n* do tempo de CPU em pedaços de no máximo *q* unidades de tempo de uma só vez. Nenhum processo espera mais do que (*n* – 1)*q* unidades de tempo.
- Desempenho
  - *q* grande ! FIFO
  - *q* pequeno ! *q* deve ser grande com relação à troca de contexto, ou então o overhead é muito alto

Exemplo de RR com quantum de tempo = 20

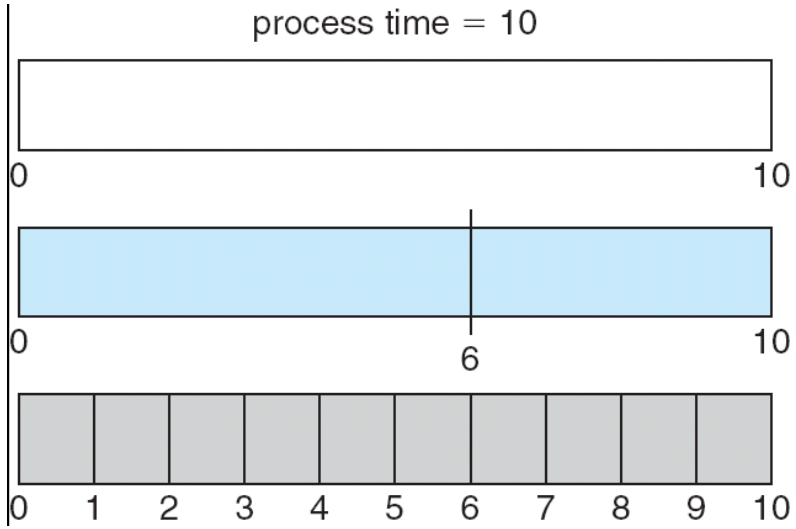
Processo	Tempo de burst
P_1	53
P_2	17
P_3	68
P_4	24

- O gráfico de Gantt é:



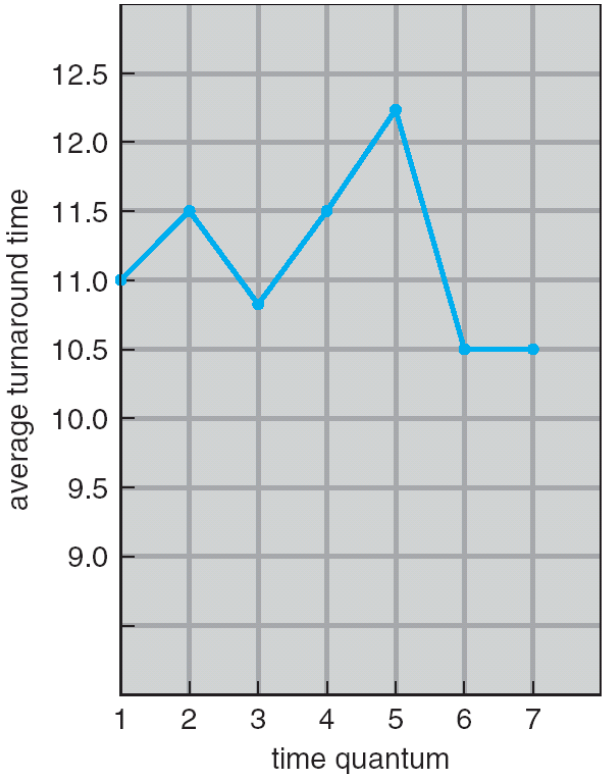
- Normalmente, maior tempo de retorno médio que SJF, porém com *resposta* melhor

Quantum de tempo e tempo de troca de contexto



quantum	context switches
12	0
6	1
1	9

Tempo de retorno varia com o quantum de tempo

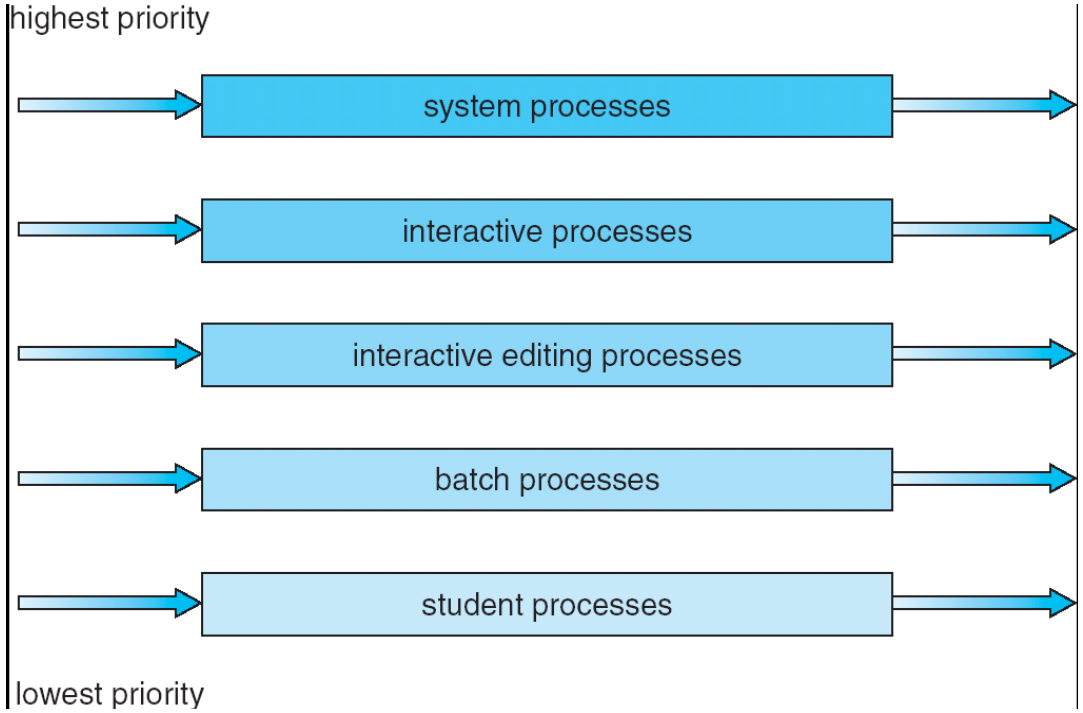


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Fila multinível

- Fila de pronto está particionada em duas filas separadas: primeiro plano (interativo) e segundo plano (batch)
- Cada fila tem seu próprio algoritmo de escalonamento
  - primeiro plano – RR
  - segundo plano – FCFS
- O escalonamento precisa ser feito entre as filas
  - Escalonamento com prioridade fixa; (ou seja, serve tudo em primeiro plano, depois em segundo plano). Possibilidade de estagnação.
  - Fatia de tempo – cada fila recebe uma certa quantidade de tempo de CPU, que ela pode escalonar entre seus processos; isto é, 80% para primeiro plano no RR
  - 20% para segundo plano no FCFS

Escalonamento de fila multinível



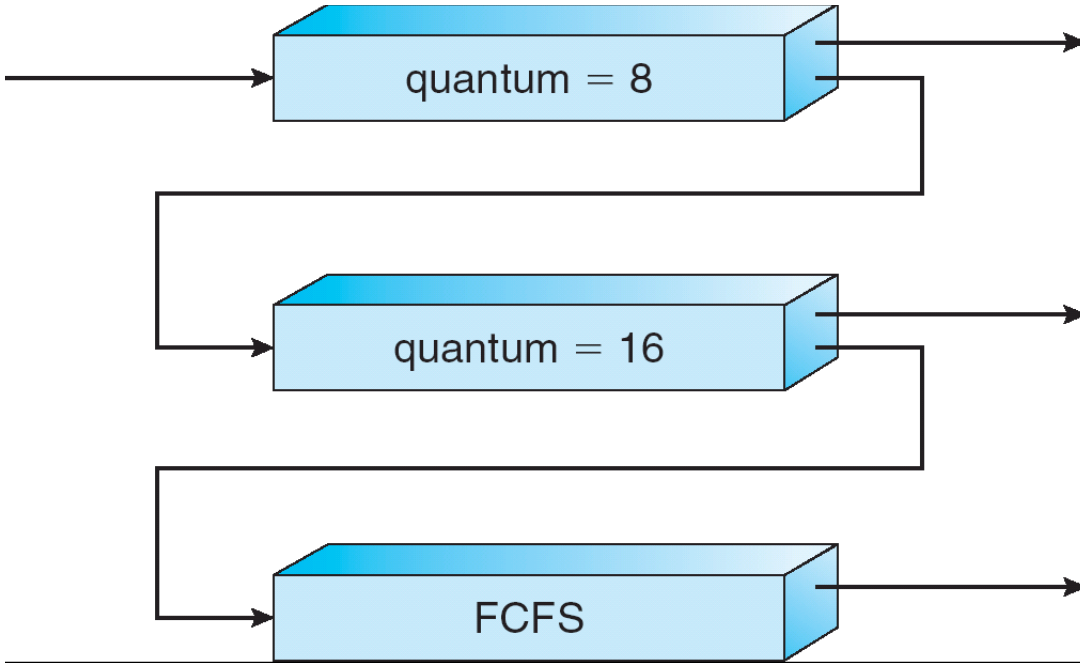
Fila de feedback multinível

- Um processo pode mover entre as diversas filas; o envelhecimento pode ser implementado dessa forma
- Escalonador de fila de feedback multinível definido pelos seguintes parâmetros:
  - número de filas
  - algoritmos de escalonamento para cada fila
  - método usado para determinar quando fazer o upgrade de um processo
  - método usado para determinar quando rebaixar um processo
  - método usado para determinar em qual fila um processo entrará quando esse processo precisar de serviço

Exemplo de fila de feedback multinível

- Três filas:
  - Q_0 – RR com quantum de tempo de 8 milissegundos
  - Q_1 – RR com quantum de tempo de 16 milissegundos
  - Q_2 – FCFS
- Escalonamento
  - Um novo job entra na fila Q_0 que é servida RR. Quando ganha a CPU, o job recebe 8 milissegundos. Se não terminar em 8 milissegundos, o job é movido para a fila Q_1.
  - Em Q_1 o job é novamente servido RR e recebe 16 milissegundos adicionais. Se não completar, ele é apropriado e movido para a fila Q_2.

Filas de feedback multinível



Escalonamento de múltiplos processadores

- Escalonamento de CPU mais complexa quando múltiplas CPUs estão disponíveis
- *Processadores homogêneos* dentro de um multiprocessador
- *Compartilhamento de carga*
- *Multiprocessamento assimétrico* – somente um processador acessa as estruturas de dados do sistema, aliviando a necessidade de compartilhamento de dados

Escalonamento de tempo real

- Sistemas de *tempo real rígido* – exigidos para completar uma tarefa crítica dentro de um período de tempo garantido
- Computação em *tempo real flexível* – exige que processos críticos recebam prioridade em relação aos menos favorecidos

Escalonamento de thread

- Escalonamento local – a biblioteca de threads decide qual thread colocar em um LWP disponível
- Escalonamento global – o kernel decide qual thread executar 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++)
       pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */

   pthread_exit(0);
}
```

Escalonamento em Java

- Política de escalonamento livremente definida. Uma thread executa até:
  1. Seu quantum de tempo expirar
  2. Ela for bloqueada para E/S
  3. Ela sair do seu método run()

Alguns sistemas *podem* dar suporte a preempção

Escalonamento em Java

- Prioridades – valores variam de 1-10

Priority	Comment
`Thread.MIN_PRIORITY`	The minimum thread priority
`Thread.MAX_PRIORITY`	The maximum thread priority
`Thread.NORM_PRIORITY`	The default thread priority

- MIN_PRIORITY is 1
- NORM_PRIORITY is 5
- MAX_PRIORITY is 10

Escalonamento em Java

Mudando de prioridade com setPriority()

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

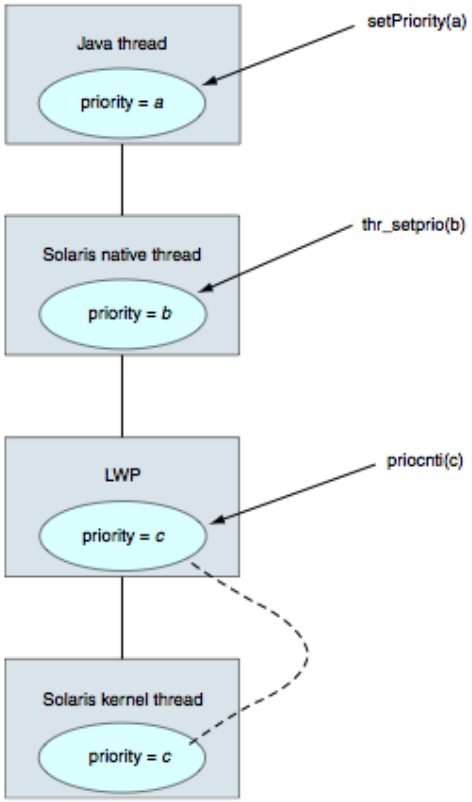
Escalonamento em Java

Relacionamento entre prioridades 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 em Java

Escalonamento de thread Java no Solaris

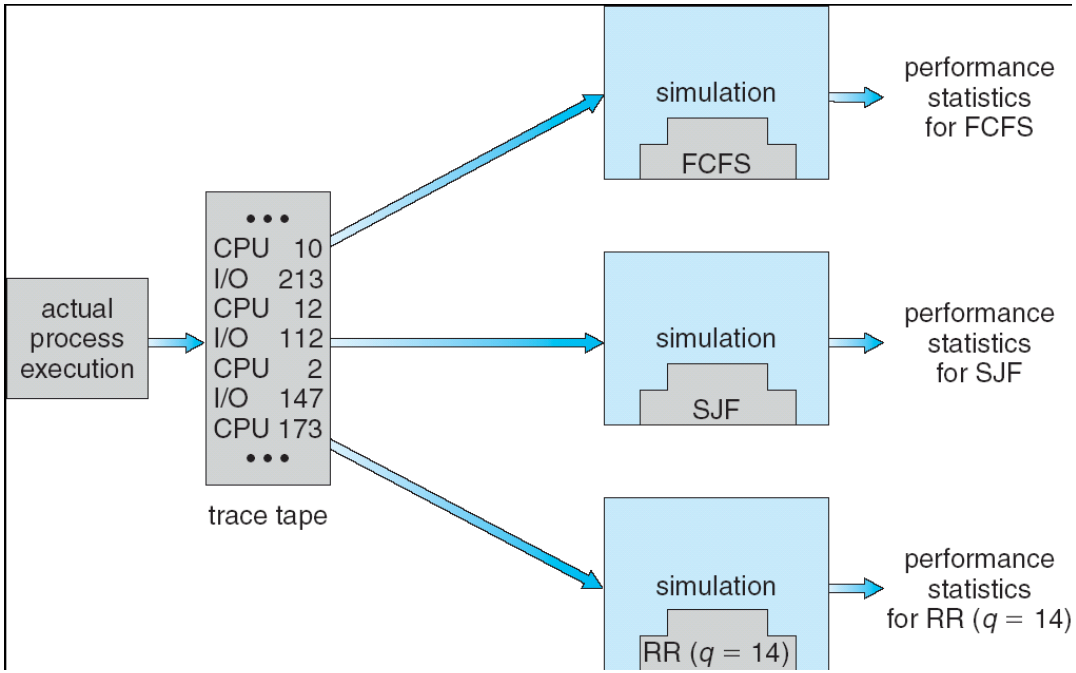


Avaliação de algoritmo

- Modelagem determinística – apanha carga de trabalho predeterminada em particular e define o desempenho de cada algoritmo para essa carga de trabalho
- Modelos de enfileiramento
- Implementação

Avaliação de algoritmo

Avaliação de escalonadores de CPU por simulação



Final do Capítulo 5