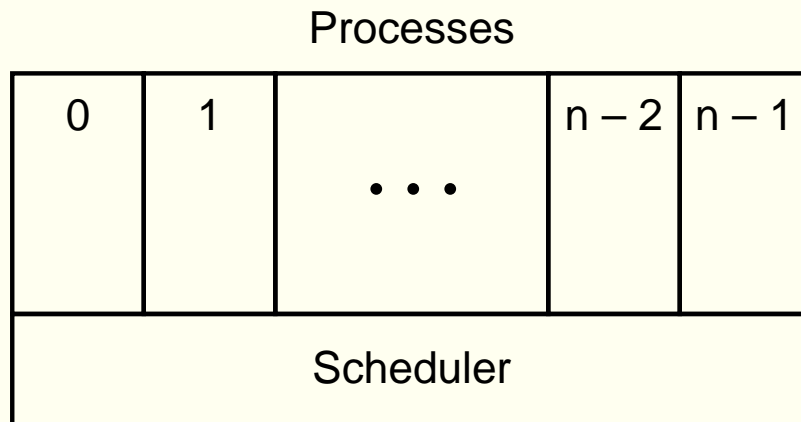


**MC514**  
**Sistemas Operacionais:**  
**Teoria e Prática**  
1s2006

**Escalonamento de Processos e Threads**

# Escalonador

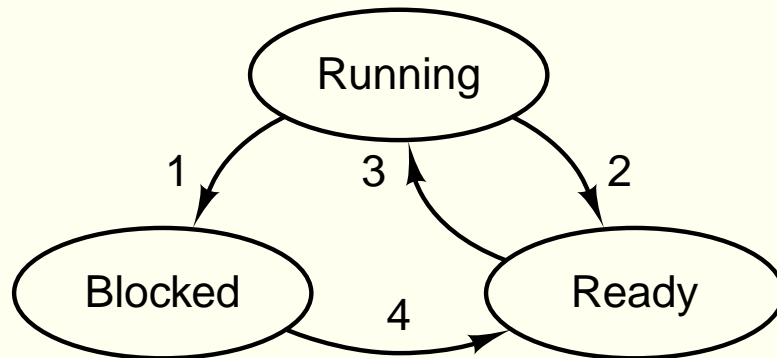


A função do escalonador é escolher qual deve ser o próximo processo a ser executado.

## Quando escalonar

- Quando um processo é criado
- Quando um processo termina
- Quando um processo faz uma operação de I/O
- Interrupção de relógio (sistemas preemptivos)

# Estado dos processos



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Por que algumas arestas estão faltando?

# Campos gerenciados por processo

- Gerência de processos: registradores, contador de programa, *program status word*, estado, prioridades, identificador de processos, sinais
- Gerência de memória: apontadores para os segmentos de dados, texto e pilha.
- Gerência de arquivos: diretório raiz e corrente, descritores de arquivos, identificadores de usuário e grupo

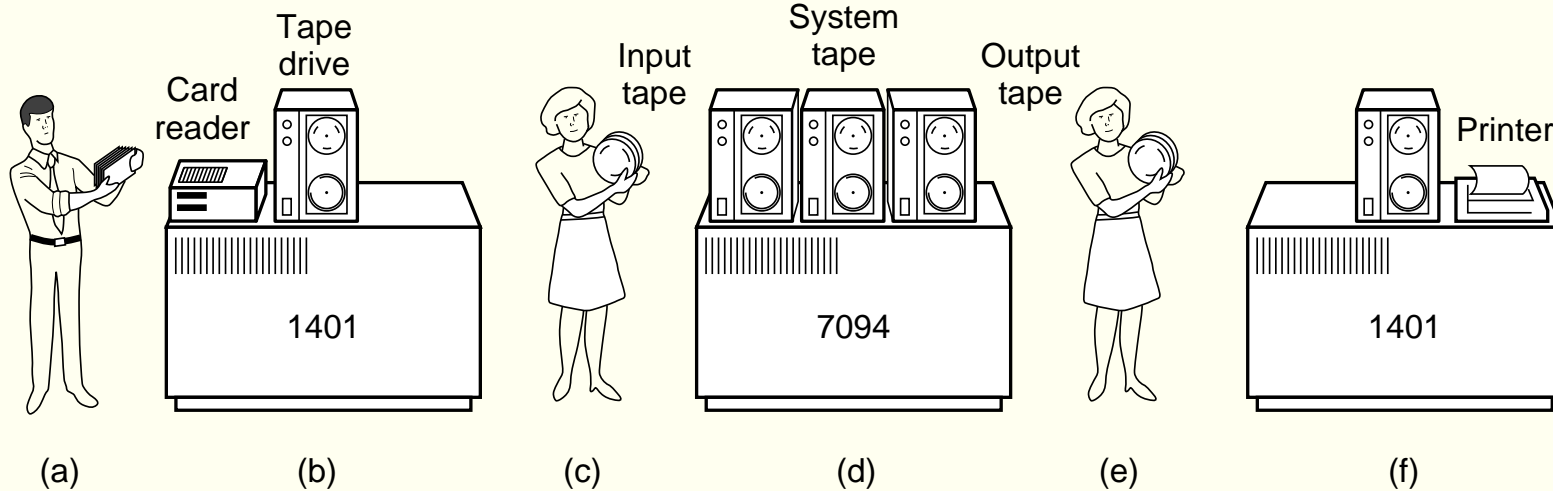
# Mudança de contexto

1. Hardware empilha contador de programa e carrega novo contador de programa do vetor de interrupções.
2. Procedimentos em linguagem assembly salvam os registradores e configuram uma nova pilha.
3. Tratador de interrupção em linguagem C é invocado
4. Escalonador decide qual processo executar
5. Procedimento C retorna para o código de máquina
6. Procedimento em linguagem assembly passa a executar o novo processo

# Objetivos dos algoritmos de escalonamento

- Justiça
  - Cada processo deve receber a sua parte da CPU
- *Policy enforcement*
  - Respeito às políticas estabelecidas
- Equilíbrio
  - Todas as partes do sistema devem estar operando

# Escalonamento em sistemas batch





# Escalonamento em sistemas batch

- *Throughput*
  - o número de jobs por hora deve ser maximizado
- *Turnaround time*
  - o tempo entre a submissão e o término de um job deve ser minimizado
- Utilização da CPU
  - A CPU deve ficar ocupada o tempo todo

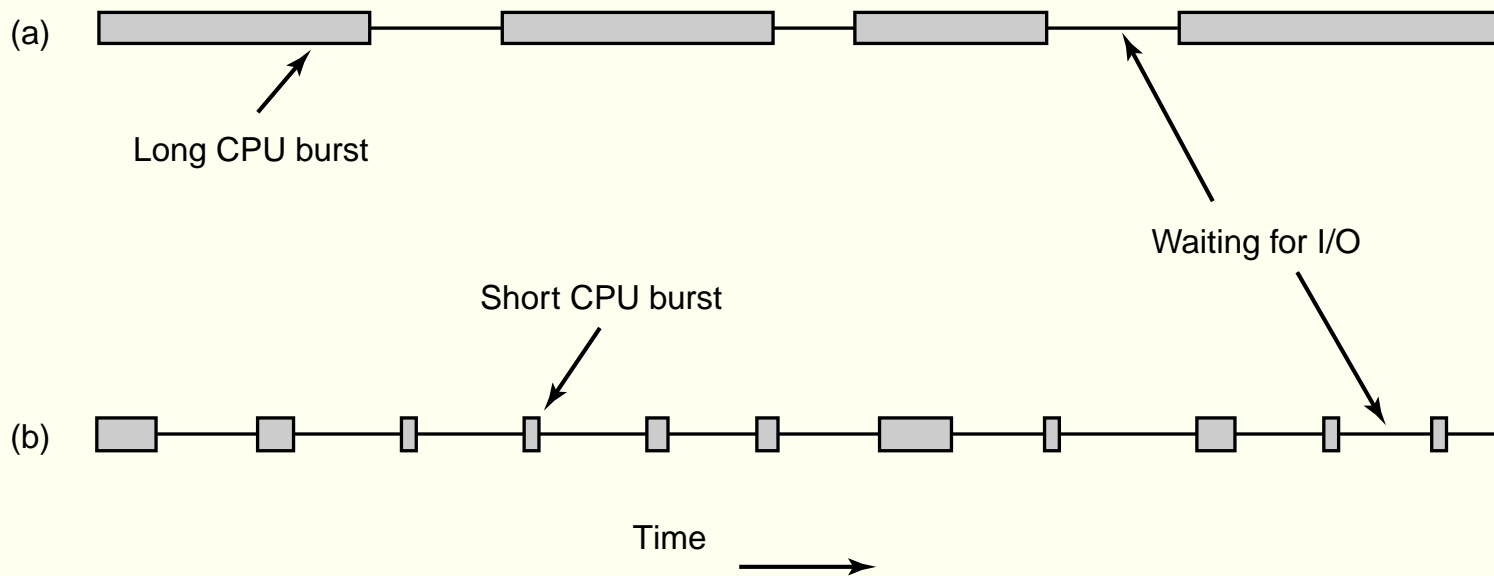
# Escalonamento em sistemas batch

## First-Come First-Served

- Processos obtêm a CPU na ordem de requisição
- Não preemptivo
- Aproveitamento ruim da CPU

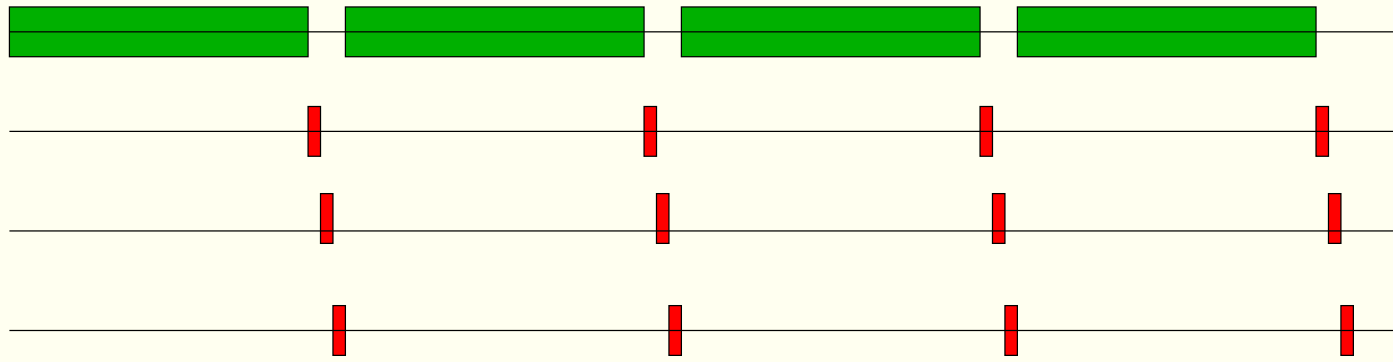
# Aproveitamento da CPU

## CPU-bound



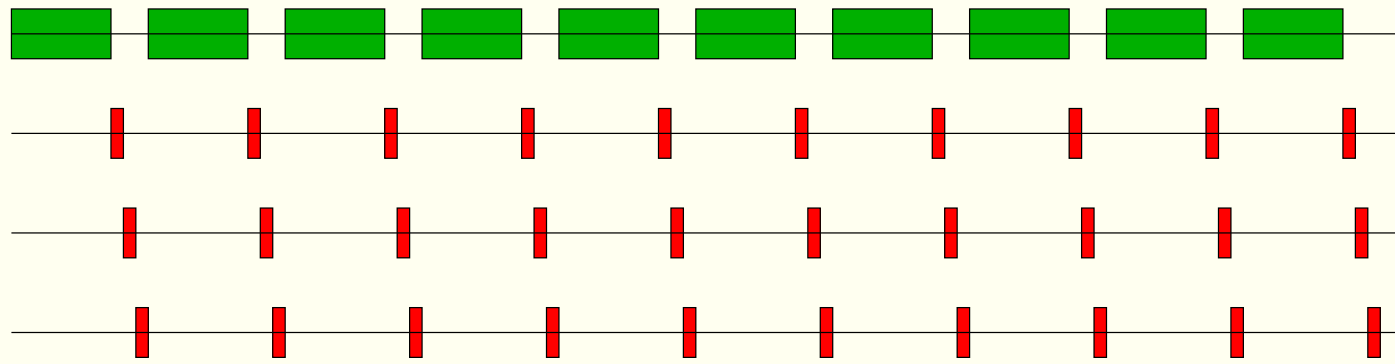
## IO-bound

# Aproveitamento da CPU



Processos I/O-bound conseguem poucos ciclos

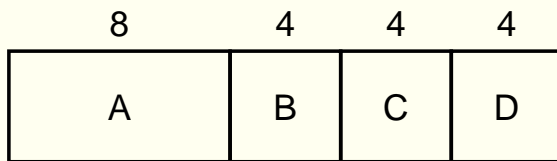
# Aproveitamento da CPU



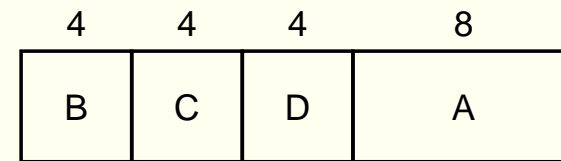
Processos I/O-bound conseguem mais ciclos

# Escalonamento em sistemas batch

## Shortest Job First



(a)



(b)

- Vazão (throughput) excelente
- Turnaround time

$$(a) (8 + 12 + 16 + 20)/4 = 14$$

$$(b) (4 + 8 + 12 + 20)/4 = 11$$

# Escalonamento em sistemas batch

## Shortest Job First

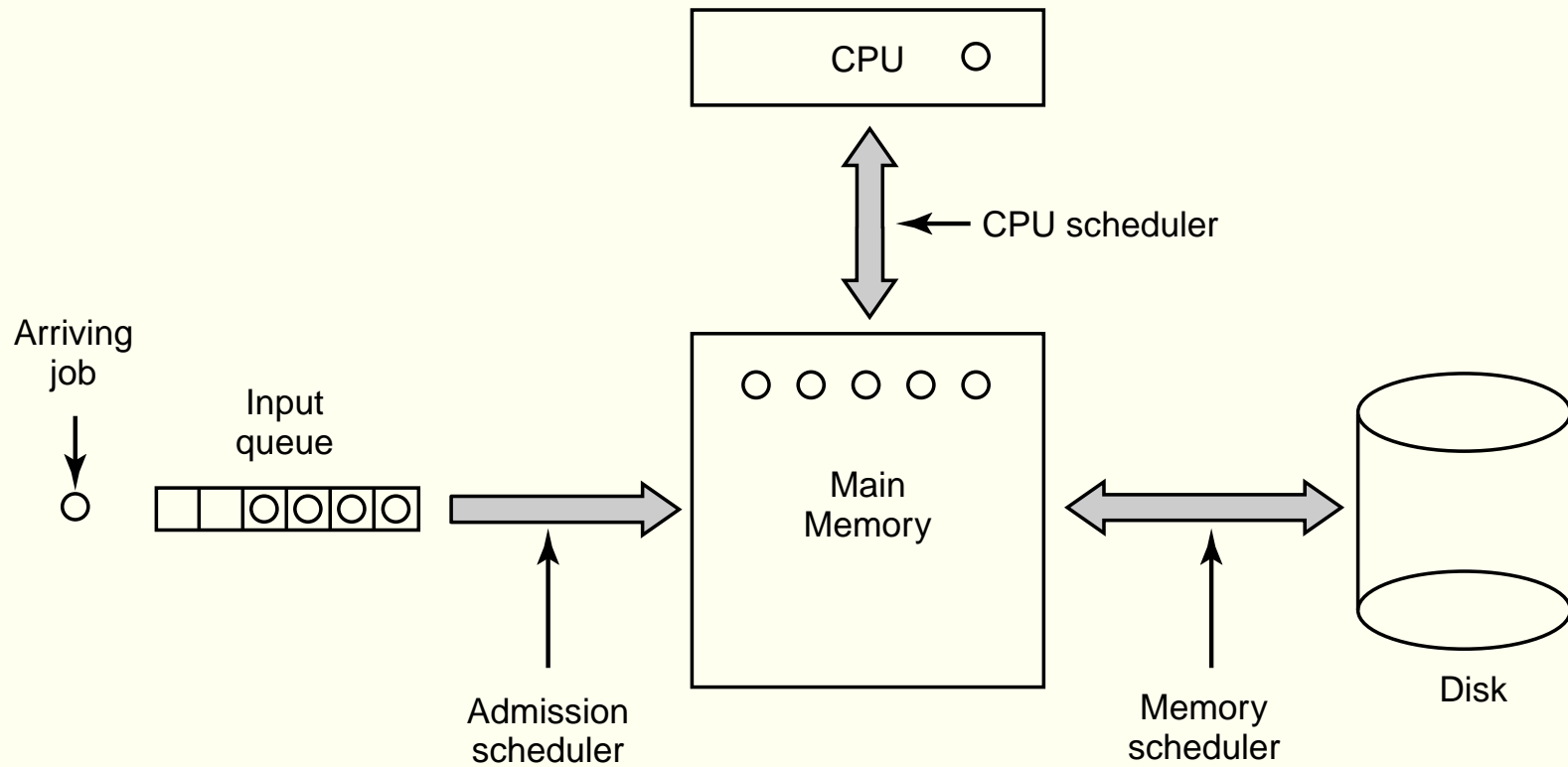
- Todos jobs precisam ser conhecidos previamente
  - Processos no tempo 0: 8 10
  - Processos no tempo 3: 4 4 8 10
- Se jobs curtos chegarem continuamente, os jobs longos nunca serão escalonados
  - Processos no tempo 100: 4 4 4 4 4 4 4 4 4 8 10

# Shortest Remaining Time Next

- Versão preemptiva do algoritmo anterior
- Tenta ficar livre logo dos processos :-)
- Bom throughput
- Bom serviço para jobs curtos novos



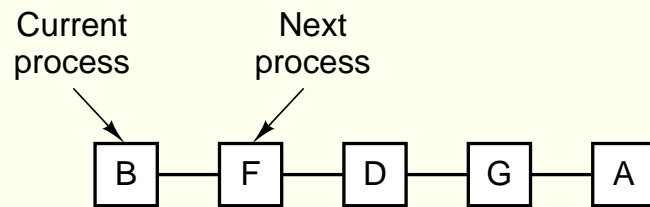
# Escalonamento em três níveis



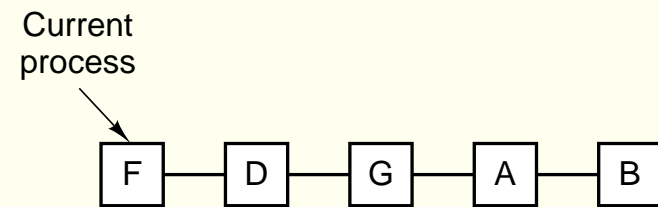
# Escalonamento em sistemas interativos

- Tempo de resposta
  - O usuário quer respostas rápidas
- Proporcionalidade
  - É necessário respeitar as expectativas de tempo

# Round-Robin



(a)

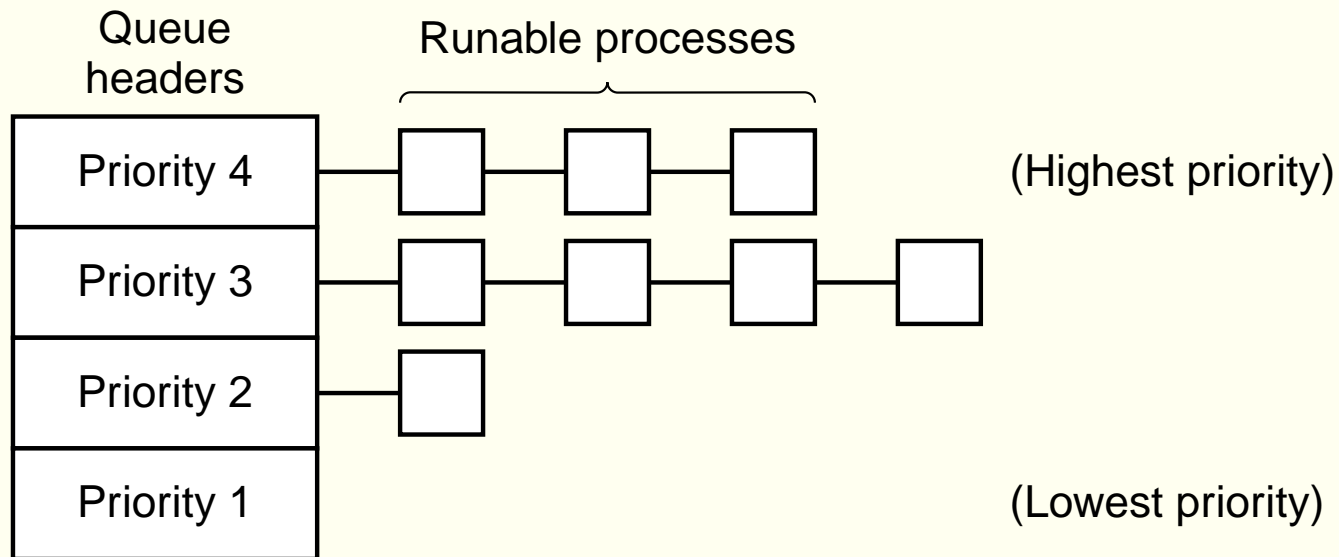


(b)

- Preemptivo
- Time quantum

Como saber o valor ideal?

# Prioridades para escalonamento



- Prioridades estáticas ou dinâmicas
- Comando `nice`

# CTSS

## Compatible Time Sharing System

- É mais eficiente rodar programas CPU-bound raramente por períodos longos do que frequentemente por períodos curtos
- Como determinar a classe de um processo?

Classe 0 (1 quantum)	→	P1	P2	P5	P7
Classe 1 (2 quanta)	→	P0	P3		
Classe 2 (4 quanta)	→	P4			
Classe 3 (8 quanta)	→	P6			

# Shortest Process Next

- Baseado no algoritmo shortest job first
- Comandos  $\equiv$  jobs
- Estimativas de tempo (*aging*)
  - $T_0$
  - $T_0/2 + T_1/2$
  - $T_0/4 + T_1/4 + T_2/2$
  - $T_0/8 + T_1/8 + T_2/4 + T_3/2$

# Justiça em sistemas interativos

- Escalonamento garantido
  - O SO faz promessas e deve mantê-las (e.g.  $1/n$  CPU)
- Loteria
  - Baseado na distribuição de tickets
  - Fácil dar pesos distintos aos processos
- *Fair-share*
  - Cada usuário receberá uma parte adequada do poder de processamento da CPU

# Escalonamento em sistemas de tempo real

- Respeitar deadlines
- Previsibilidade
- Hard real time e soft real time
- Tratamento dos eventos periódicos  $\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$

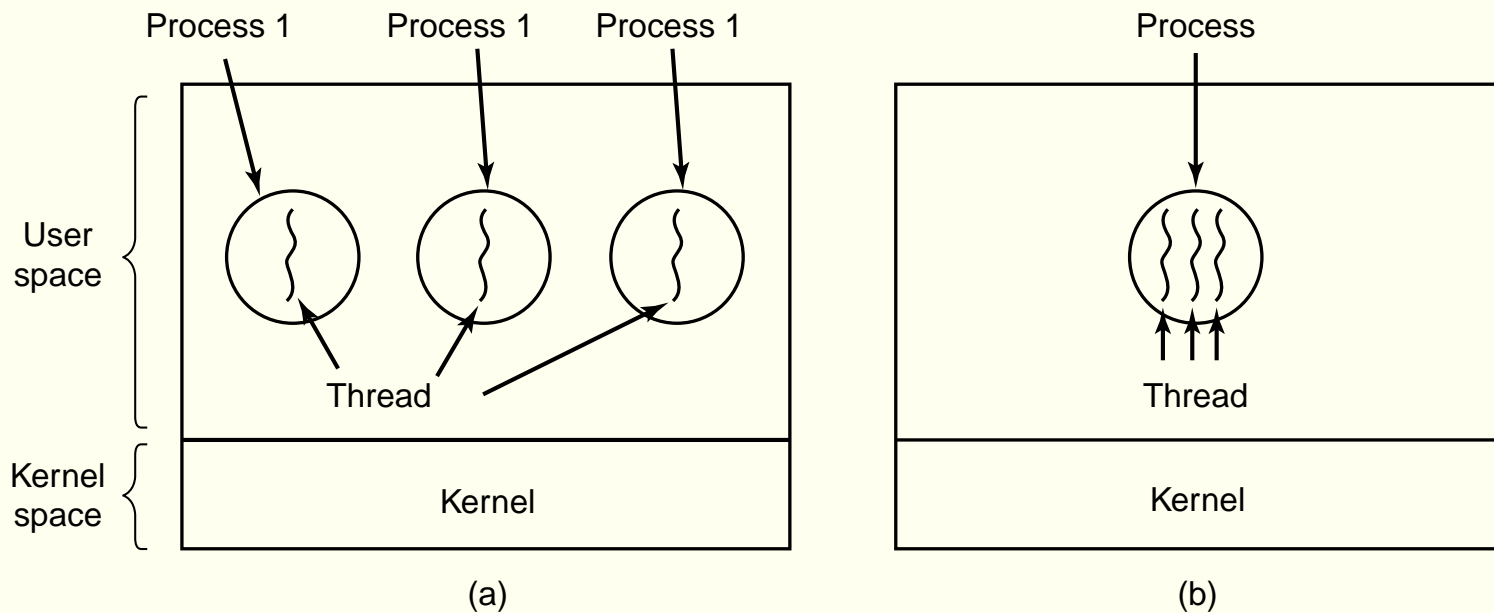


# Política versus Mecanismo

- Considere o seguinte cenário:
  - um processo pai e vários processos filhos
  - os processos filhos atendem requisições
  - o processo pai sabe quais são as mais prioritárias
- O escalonador provê o **mecanismo** de escalonamento
- O processo pai sabe a **política** a ser adotada

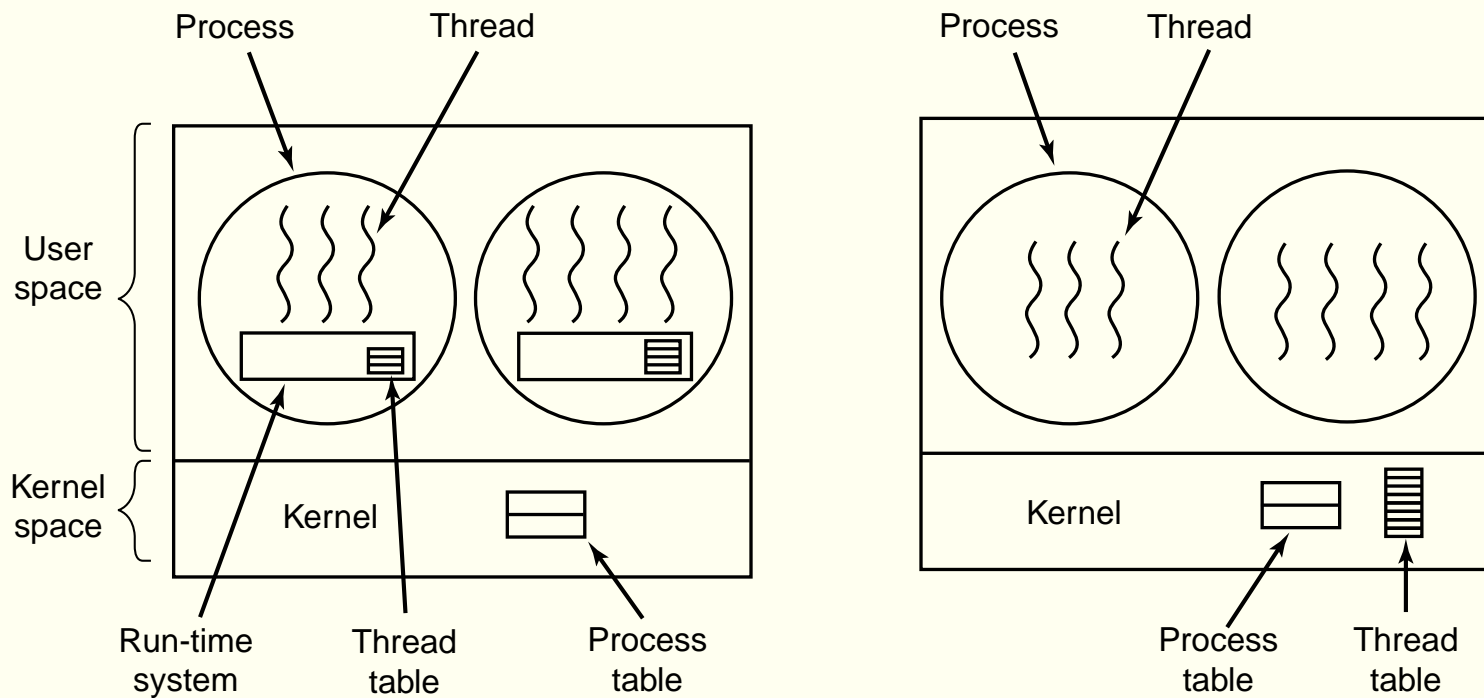
# Escalonamento de threads

- Escalonamento em dois níveis: processos e threads



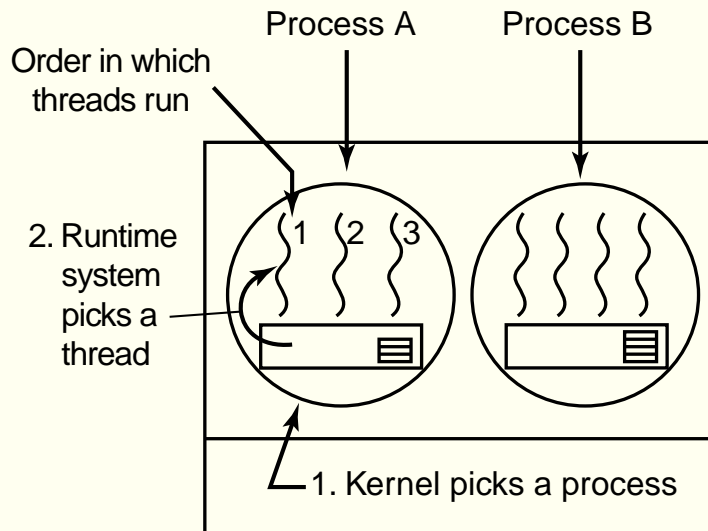
# Escalonamento de threads

## Threads de usuário e de kernel



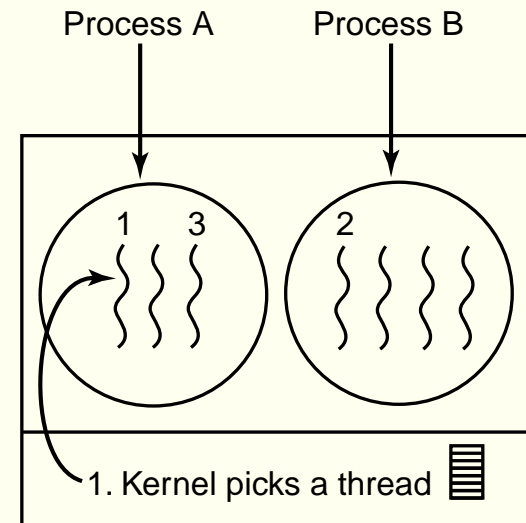
# Escalonamento de threads

## Threads de usuário e de kernel



Possible:      A1, A2, A3, A1, A2, A3  
Not possible: A1, B1, A2, B2, A3, B3

(a)



Possible:      A1, A2, A3, A1, A2, A3  
Also possible: A1, B1, A2, B2, A3, B3

(b)

# Escalonamento de threads

## Implementação híbrida

