SO: Escalonamento

Projetos de Sistemas Operacionais

Prof. Dr. Denis M. L. Martins

Engenharia de Computação: 5º Semestre

Introdução

Objetivos de Aprendizado

- Explicar o conceito de escalonamento e sua importância em sistemas operacionais.
- Compreender os mecanismos de escalonamento e seus objetivos.
- Comparar diferentes algoritmos de escalonamento e seus impactos no desempenho do SO.

Disclaimer

Parte do material apresentado a seguir foi adaptado de *IT Systems – Open Educational Resource*, disponível em https://oer.gitlab.io/oer-courses/it-systems/, produzido por Jens Lechtenböger, e distribuído sob a licença CC BY-SA 4.0.

Multicore e Multithread

Chip-multithreading (CMT)

- Cada core possui múltiplas hardware threads (núcleos **lógicos** dentro de cada núcleo **físico**, com seus próprios registradores).
- Intel chama isso de *hyperthreading*.
- Em um sistema quad-core com 2 hardware threads por core, o SO "percebe" 8 processadores lógicos.
- Threads concorrentes ainda compartilham recursos internos do núcleo.

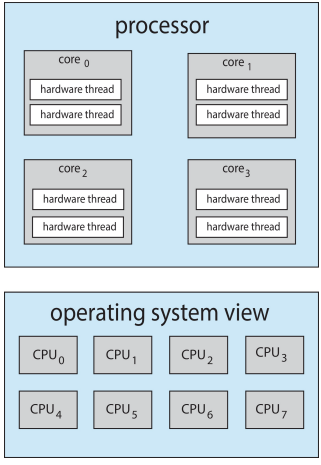


Figura 1: Processador multicore. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento

Escalonamento de CPU

- **Função principal**: Permitir multitarefa.
- Utilização máxima da CPU usando multitarefa.
  - Múltiplos processos e threads competindo pela CPU ao mesmo tempo.
  - Escalonamento para selecionar quais processos/threads serão executados na CPU.

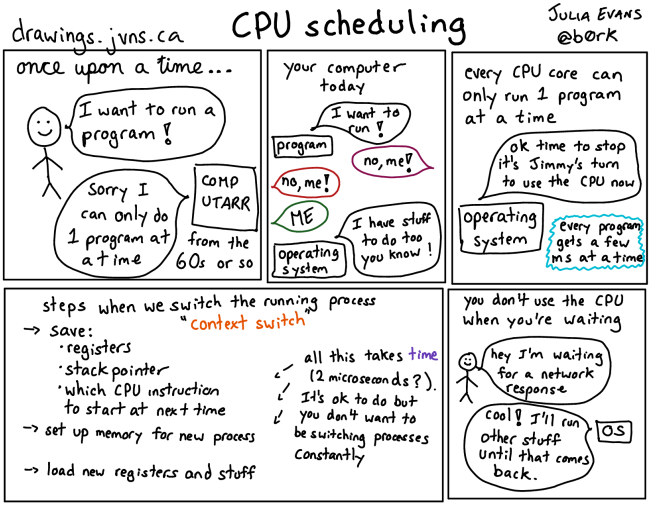


Figura 2: Escalonamento de CPU. Créditos: Julia Evans.

Escalonamento de CPU

- **Função principal**: Permitir multitarefa.
- Utilização máxima da CPU usando multitarefa.
  - Múltiplos processos e threads competindo pela CPU ao mesmo tempo.
  - Escalonamento para selecionar quais processos/threads serão executados na CPU.

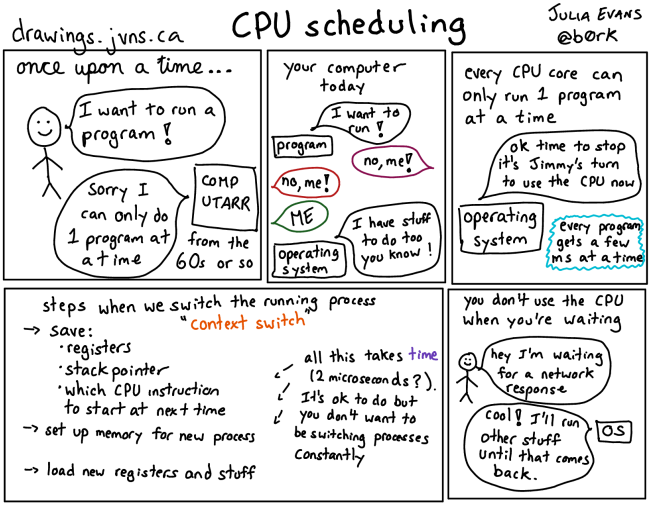


Figura 2: Escalonamento de CPU. Créditos: Julia Evans.

Escalonamento de CPU

- **Função principal**: Permitir multitarefa.
- Utilização máxima da CPU usando multitarefa.
  - Múltiplos processos e threads competindo pela CPU ao mesmo tempo.
  - Escalonamento para selecionar quais processos/threads serão executados na CPU.

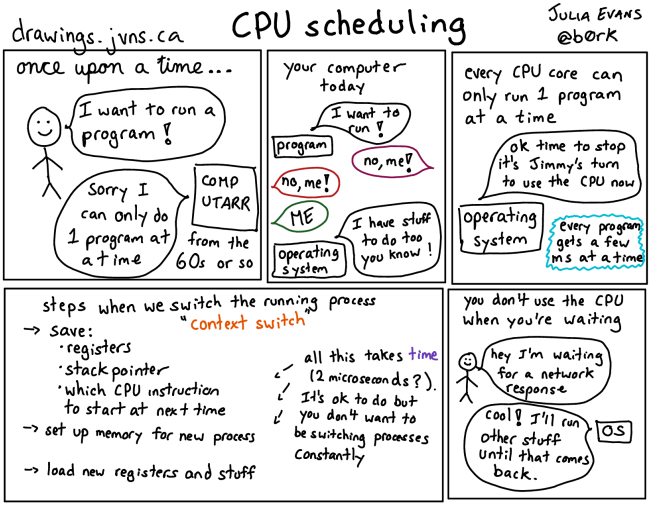


Figura 2: Escalonamento de CPU. Créditos: Julia Evans.

Escalonamento de CPU

- **Função principal**: Permitir multitarefa.
- Utilização máxima da CPU usando multitarefa.
  - Múltiplos processos e threads competindo pela CPU ao mesmo tempo.
  - Escalonamento para selecionar quais processos/threads serão executados na CPU.

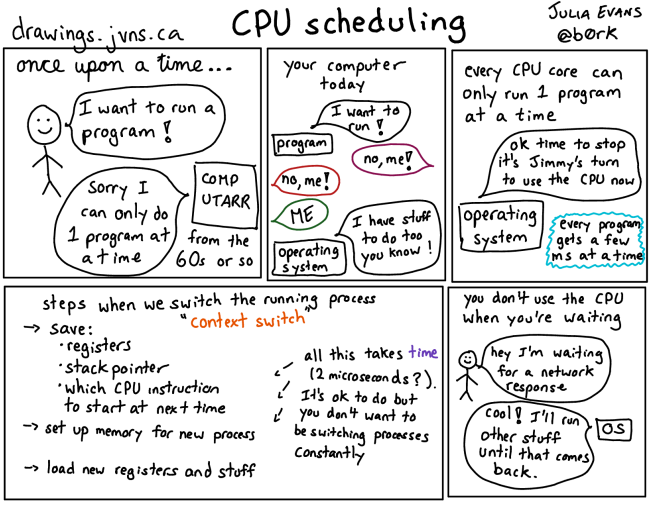


Figura 2: Escalonamento de CPU. Créditos: Julia Evans.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

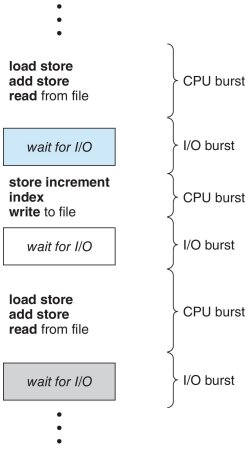


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

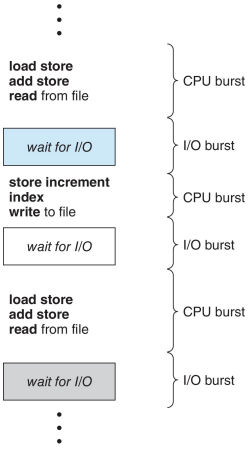


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

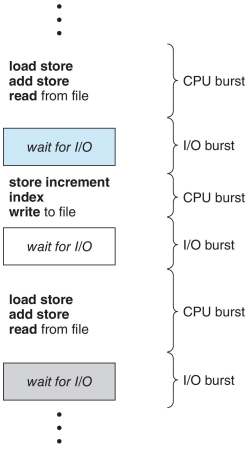


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

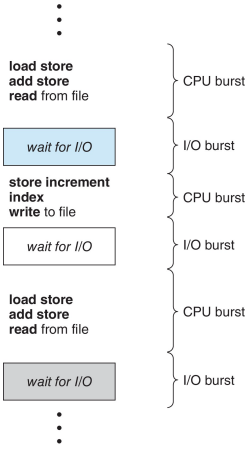


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

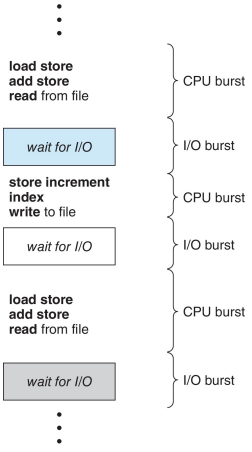


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas → processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

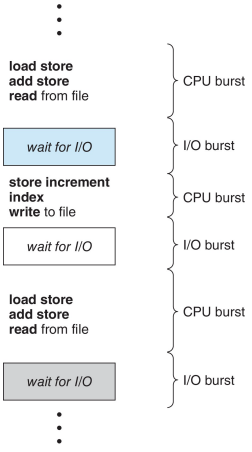


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas $\rightarrow$ processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

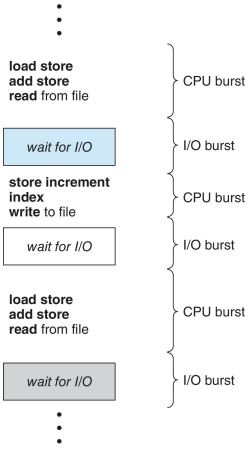


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU (cont.)

- **Alternância** de surtos de CPU com surtos de E/S (disco ou rede)
  - Do inglês, *CPU–I/O Burst Cycle*.
  - Execução de processo consiste de um ciclo de execução na CPU e espera por E/S.
  - Distribuição de surtos é a preocupação principal.
- Entrada/Saída (E/S) (*I/O*, no inglês) é muito mais lento que a CPU.
  - Processo entra no estado bloqueado (*waiting*) esperando por um dispositivo externo.
  - CPUs mais rápidas $\rightarrow$ processos tendem a ficar mais limitados pela E/S (*I/O bound*).
- **Fator principal**: comprimento do surto de CPU.

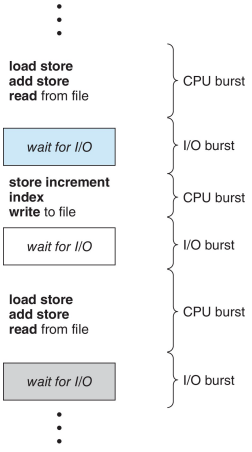


Figura 3: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Debug seu conhecimento

Quais afirmações são corretas sobre threads?

1. Programas em execução são gerenciados como threads pelo sistema operacional.
2. Processos e threads são unidades de gerenciamento do sistema operacional.
3. Cada processo pode conter uma ou mais threads.
4. A concorrência só pode ocorrer em sistemas multi-core.
5. O escalonamento pode levar à execução intercalada de múltiplas threads.

Relembre os conceitos fundamentais

Reflita sobre cada alternativa. Tente justificar sua resposta para cada uma delas.

Objetivos/Critérios

- **Utilização de CPU**: manter CPU ocupada.
- **Vazão** (*Throughput*): número de processos/threads completados por unidade de tempo.
- **Tempo de espera**: tempo que um processo está esperando na fila de *pronto* (ready) $\rightarrow$ poderia estar executando, mas não está.
- **Tempo de resposta**: tempo entre a submissão de um processo/thread até a produção da primeira resposta.
- *Em dispositivos móveis*: **consumo de energia** (bateria) pode ser um critério

Trade-offs

Em geral, objetivos são conflitantes.

Exemplo

Sempre executar tarefas curtas em detrimento das tarefas longas: $\uparrow$ vazão, $\downarrow$ tempo de resposta.

Objetivos/Critérios

- **Utilização de CPU**: manter CPU ocupada.
- **Vazão** (*Throughput*): número de processos/threads completados por unidade de tempo.
- **Tempo de espera**: tempo que um processo está esperando na fila de *pronto* (ready) $\rightarrow$ poderia estar executando, mas não está.
- **Tempo de resposta**: tempo entre a submissão de um processo/thread até a produção da primeira resposta.
- *Em dispositivos móveis*: **consumo de energia** (bateria) pode ser um critério

Trade-offs

Em geral, objetivos são conflitantes.

Exemplo

Sempre executar tarefas curtas em detrimento das tarefas longas: $\uparrow$ vazão, $\downarrow$ tempo de resposta.

Objetivos/Critérios

- **Utilização de CPU**: manter CPU ocupada.
- **Vazão** (*Throughput*): número de processos/threads completados por unidade de tempo.
- **Tempo de espera**: tempo que um processo está esperando na fila de *pronto* (ready) $\rightarrow$ poderia estar executando, mas não está.
- **Tempo de resposta**: tempo entre a submissão de um processo/thread até a produção da primeira resposta.
- *Em dispositivos móveis*: **consumo de energia** (bateria) pode ser um critério

Trade-offs

Em geral, objetivos são conflitantes.

Exemplo

Sempre executar tarefas curtas em detrimento das tarefas longas: $\uparrow$ vazão, $\downarrow$ tempo de resposta.

Eficiência do Escalonador

- Além de escolher o processo certo a ser executado, o escalonador precisa fazer uso eficiente da CPU.
- Lembre que a alternância (ou chaveamento) de processos é **cara**.
- Sequência de ações:
  1. Trocar de modo usuário para *modo de kernel*
  2. Salvar estado do processo atual
  3. Inicializar novo processo (restaurar ou carregar info. na memória)
  4. Potencialmente: refazer cache de memória. (tema de aula futura)

Principais Componentes

Escalonador (Scheduler)

Planejamento: Seleciona qual processo/thread vai usar a CPU.

Despachante (Dispatcher)

Alocação: Oferece controle da CPU para o processo/thread .

Latência do Despachante

Tempo que o Despachante pare um(a) processo/thread e (re)comece outro(a).
Mudança de contexto $\rightarrow$ salvar o estado atual no PCB/TCB e restaurar estado de outro PCB/TCB.

Principais Componentes

Escalonador (Scheduler)

Planejamento: Seleciona qual processo/thread vai usar a CPU.

Despachante (Dispatcher)

Alocação: Oferece controle da CPU para o processo/thread .

Latência do Despachante

Tempo que o Despachante pare um(a) processo/thread e (re)comece outro(a).
Mudança de contexto $\rightarrow$ salvar o estado atual no PCB/TCB e restaurar estado de outro PCB/TCB.

Principais Componentes

Escalonador (Scheduler)

Planejamento: Seleciona qual processo/thread vai usar a CPU.

Despachante (Dispatcher)

Alocação: Oferece controle da CPU para o processo/thread .

Latência do Despachante

Tempo que o Despachante pare um(a) processo/thread e (re)comece outro(a).
Mudança de contexto $\rightarrow$ salvar o estado atual no PCB/TCB e restaurar estado de outro PCB/TCB.

Funcionamento do Escalonador

- Seleciona entre os processos prontos.
- Questão Fundamental: Quando tomar decisão de escalonamento?
  - Novo processo criado.
  - Término de um processo.
  - Processo bloqueado por E/S.
  - Interrupção de E/S. (tema de aula futura)

Tipos de Escalonamento

Não-preemptivo

- Escolhe um processo e o deixa ser executado até que ele seja bloqueado ou libere a CPU voluntariamente.
- *Sem suspeção forçosa* por parte do escalonador.

Preemptivo → Mais usado por SO modernos

- Escolhe um processo e o deixa ser executado por no máximo um período de tempo predeterminado.
- Após esse período, caso o processo ainda esteja em execução, o processo é suspenso e outro processo é escolhido para executar.
- *Time-slicing*. Interrupção ocorre ao fim do período de tempo para devolver o controle da CPU para o escalonador.

Tipos de Escalonamento

Não-preemptivo

- Escolhe um processo e o deixa ser executado até que ele seja bloqueado ou libere a CPU voluntariamente.
- *Sem suspeção forçosa* por parte do escalonador.

Preemptivo → Mais usado por SO modernos

- Escolhe um processo e o deixa ser executado por no máximo um período de tempo predeterminado.
- Após esse período, caso o processo ainda esteja em execução, o processo é suspenso e outro processo é escolhido para executar.
- *Time-slicing*. Interrupção ocorre ao fim do período de tempo para devolver o controle da CPU para o escalonador.

Algoritmos de Escalonamento

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado → seleciona processo no começo da fila de prontos → quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado $\rightarrow$ seleciona processo no começo da fila de prontos $\rightarrow$ quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado → seleciona processo no começo da fila de prontos → quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado → seleciona processo no começo da fila de prontos → quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado $\rightarrow$ seleciona processo no começo da fila de prontos $\rightarrow$ quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado $\rightarrow$ seleciona processo no começo da fila de prontos $\rightarrow$ quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

First-Come, First-Served (FCFS) ou First-In, First-Out (FIFO)

- Não-preemptivo
- CPU atribuída por ordem de chegada
- Fila única de processos prontos
- Novos processos são alocados no *fim da fila*
- **Funcionamento**: processo em execução termina, desiste, ou é bloqueado $\rightarrow$ seleciona processo no começo da fila de prontos $\rightarrow$ quando o processo bloqueado volta a estar pronto, ele vai para o *fim* da fila de prontos.
- **Vantagem**: simples de programar.
- **Desvantagem**: pode causar longos tempos de espera. Não otimiza a eficiência de alocação da CPU.

Escalonamento FCFS/FIFO: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Serviço
P_1	24
P_2	3
P_3	3

Exemplo 1. Ordem de Chegada: P_1, P_2, P_3

No escalonamento FCFS, teríamos:

P_1	P_2	P_3

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

Exemplo 2. Ordem de Chegada: P_2, P_3, P_1

No escalonamento FCFS, teríamos:

P_2	P_3	P_1

Tempo de espera: $P_1 = 6; P_2 = 0; P_3 = 3$.
Tempo médio de espera: $(6+0+3)/3 = 3$.

Escalonamento FCFS/FIFO: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Serviço
P_1	24
P_2	3
P_3	3

Exemplo 1. Ordem de Chegada: P_1, P_2, P_3

No escalonamento FCFS, teríamos:

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

Exemplo 2. Ordem de Chegada: P_2, P_3, P_1

No escalonamento FCFS, teríamos:

Tempo de espera: $P_1 = 6; P_2 = 0; P_3 = 3$.
Tempo médio de espera: $(6+0+3)/3 = 3$.

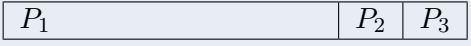
Escalonamento FCFS/FIFO: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Serviço
P_1	24
P_2	3
P_3	3

Exemplo 1. Ordem de Chegada: P_1, P_2, P_3

No escalonamento FCFS, teríamos:



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

Exemplo 2. Ordem de Chegada: P_2, P_3, P_1

No escalonamento FCFS, teríamos:

Tempo de espera: $P_1 = 6; P_2 = 0; P_3 = 3.$
Tempo médio de espera: $(6+0+3)/3 = 3.$

Shortest Job First (SJF)

- Não-preemptivo
- Tarefa mais curta primeiro
- CPU atribuída ao processo que tem menor tempo de serviço
- **Vantagem**: ótimo quando todas as tarefas estão disponíveis simultaneamente $rightarrow$ minimiza tempo médio de espera.
- **Desvantagem**: difícil de determinar o tempo de serviço (estimar, pedir informação durante criação do processo/thread)

Shortest Job First (SJF)

- Não-preemptivo
- Tarefa mais curta primeiro
- CPU atribuída ao processo que tem menor tempo de serviço
- **Vantagem**: ótimo quando todas as tarefas estão disponíveis simultaneamente $rightarrow$ minimiza tempo médio de espera.
- **Desvantagem**: difícil de determinar o tempo de serviço (estimar, pedir informação durante criação do processo/thread)

Shortest Job First (SJF)

- Não-preemptivo
- Tarefa mais curta primeiro
- CPU atribuída ao processo que tem menor tempo de serviço
- **Vantagem**: ótimo quando todas as tarefas estão disponíveis simultaneamente $rightarrow$ minimiza tempo médio de espera.
- **Desvantagem**: difícil de determinar o tempo de serviço (estimar, pedir informação durante criação do processo/thread)

Shortest Job First (SJF)

- Não-preemptivo
- Tarefa mais curta primeiro
- CPU atribuída ao processo que tem menor tempo de serviço
- **Vantagem**: ótimo quando todas as tarefas estão disponíveis simultaneamente $rightarrow$ minimiza tempo médio de espera.
- **Desvantagem**: difícil de determinar o tempo de serviço (estimar, pedir informação durante criação do processo/thread)

Shortest Job First (SJF)

- Não-preemptivo
- Tarefa mais curta primeiro
- CPU atribuída ao processo que tem menor tempo de serviço
- **Vantagem**: ótimo quando todas as tarefas estão disponíveis simultaneamente $rightarrow$ minimiza tempo médio de espera.
- **Desvantagem**: difícil de determinar o tempo de serviço (estimar, pedir informação durante criação do processo/thread)

Escalonamento SJF: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Serviço
P_1	6
P_2	8
P_3	7
P_4	3

Exemplo SJF

No escalonamento SJF, teríamos:

Tempo médio de espera: $(3+16+9+0)/4 = 7$.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Shortest Remaining Time First (SRTF)

- Tempo restante mais curto primeiro
- Versão preemptiva do SJF
- CPU atribuída ao processo que tem menor tempo de serviço restante
  - Quando um novo processo chega na fila de prontos, seu tempo total é comparado com o tempo restante do processo atual.
  - Se $t_{novo} \leq t_{atual}$, então o processo atual é suspenso e o novo processo é selecionado para execução.
- **Vantagem**: reduz tempo de espera em cenários dinâmicos (onde novos processos são criados ao longo do tempo).
- **Desvantagens**:
  - Difícil de estimar o tempo restante.
  - Pode acarretar em muitas mudanças de contexto.

Escalonamento SRTF: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Chegada	Tempo de Serviço
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Exemplo SRTF

No escalonamento SRTF, teríamos:

P_1	P_2	P_4	P_1	P_3

Tempo médio de espera: $(10-1)+(1-1)+(17-2)+(5-3)/4 = 6.5$.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Round Robin

- Chaveamento Circular
- Preemptivo: a cada processo é designado um intervalo de tempo (*quantum*) q durante o qual o processo pode executar.
  - Expirado o intervalo, o processo sofre preempção e vai para o fim da fila de prontos
  - A CPU recebe um outro processo pronto.
  - Se o processo termina ou é bloqueado, então ele sai da CPU e um novo é selecionado.
- Um timer interrompe a cada q unidades de tempo para que outro processo seja escalonado.
- Tradicionalmente, o *quantum* q é configurado entre 10-100 ms.
  - Se n processos estão na fila de prontos, então cada processo tem $1/n$ do tempo de CPU em cada rodada.
  - Nenhum processo espera mais que $(n-1)q$ unidades de tempo.
- **Vantagem**: garante distribuição justa da CPU (sem monopólio)
- **Desvantagem**: decidir o comprimento de q
  - Se q for longo $\rightarrow$ FIFO/FCFS
  - Tem que levar em consideração o *overhead* de mudanças de contexto.

Escalonamento RR: Exemplo

Considere o cenário abaixo:

Processo	Tempo de Serviço
P_1	24
P_2	3
P_3	3

Exemplo RR

No escalonamento RR com $q = 4$, teríamos:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

P_1 sai da CPU no tempo 4 e retorna no tempo 10. Então, o seu tempo espera é $10 - 4 = 6\ ms$.
P_2 espera por $4\ ms$ e P_3 por $7\ ms$.
Tempo médio de espera: $^{(6+4+7)}/_{3} = 5.66\ ms$.

Escalonamento RR: Exercício

Crie o Gantt chart mostrando o escalonamento RR com $q = 4$ no cenário abaixo.

Calcule também os tempos de espera para cada processo e o tempo médio de espera.

Processo	Tempo de Chegada	Tempo de Serviço
P_1	0	24
P_2	1	3
P_3	2	3

Prioridades e Escalonamento

Threads e Prioridades

- Threads podem ter diferentes prioridades (especifica durante sua criação).
- Prioridade é armazenada no TCB.
- SO pode oferecer formas de modificar prioridades em tempo de execução.

Filas de Threads

- Relembre os estados de uma thread: nova, em execução, em espera, pronta, concluída.
- SO gerencia estados de threads via filas (com estrutura de dados apropriada).
  - Fila(s) de execução: potencialmente por núcleo de CPU.
  - Fila(s) de espera: potencialmente por tipo de evento (bloqueio/interrupção).

Threads e Prioridades

- Threads podem ter diferentes prioridades (especifica durante sua criação).
- Prioridade é armazenada no TCB.
- SO pode oferecer formas de modificar prioridades em tempo de execução.

Filas de Threads

- Relembre os estados de uma thread: nova, em execução, em espera, pronta, concluída.
- SO gerencia estados de threads via filas (com estrutura de dados apropriada).
  - Fila(s) de execução: potencialmente por núcleo de CPU.
  - Fila(s) de espera: potencialmente por tipo de evento (bloqueio/interrupção).

Escalonamento por Prioridades

- Escalonador leva prioridades em consideração.
  - Uma thread pode, a qualquer momento, desistir de sua execução chamando o método `yield`.
  - Thread de mesma prioridade seja escalonada e despachada para a CPU.
  - Em geral, tentativa de desistência para uma thread me prioridade mais baixa são ignoradas.
- SJF pode ser visto como escalonamento por prioridades.
  - Prioridade é inversa ao tempo estimado de serviço.
- Problema: ***starvation*** (tema de aula futura)
  - Threads com baixa prioridade podem nunca executar.
  - Solução: ***aging*** a prioridade de uma thread aumenta conforme o tempo passa.

Fechamento e Perspectivas

Palavras Finais

- **Resumo**
  - Tempo compartilhado: escalonador escolhe qual processo ou thread usa a CPU em um determinado momento e por quanto tempo.
  - Kernel pode gerenciar o escalonamento por threads ou por processos.
- **Diferentes tipos de escalonadores**
  - Não-preemptivo e Preemptivo.
  - Algoritmos tradicionais: FIFO, Round Robin.
  - É essencial para aplicações modernas como servidores web, sistemas em tempo real e aplicações gráficas.
- **Próximos passos**
  - Explorar o conceito de escalonamento no capítulo 2.4 do livro de TANENBAUM[1] e no capítulo 3 do livro de Hailperin[2].
  - Implementar algoritmos de escalonamento.

[1] A. TANENBAUM. 2015. Sistemas Operacionais Modernos. 4a ed. Pearson Brasil

[2] M. Hailperin. 2019. Operating Systems and Middleware - Supporting Controlled Interaction. Revised edition 1.3.1.

Palavras Finais

- **Resumo**
  - Tempo compartilhado: escalonador escolhe qual processo ou thread usa a CPU em um determinado momento e por quanto tempo.
  - Kernel pode gerenciar o escalonamento por threads ou por processos.
- **Diferentes tipos de escalonadores**
  - Não-preemptivo e Preemptivo.
  - Algoritmos tradicionais: FIFO, Round Robin.
  - É essencial para aplicações modernas como servidores web, sistemas em tempo real e aplicações gráficas.
- **Próximos passos**
  - Explorar o conceito de escalonamento no capítulo 2.4 do livro de TANENBAUM[1] e no capítulo 3 do livro de Hailperin[2].
  - Implementar algoritmos de escalonamento.

[1] A. TANENBAUM. 2015. Sistemas Operacionais Modernos. 4a ed. Pearson Brasil

[2] M. Hailperin. 2019. Operating Systems and Middleware - Supporting Controlled Interaction. Revised edition 1.3.1.

Palavras Finais

- **Resumo**
  - Tempo compartilhado: escalonador escolhe qual processo ou thread usa a CPU em um determinado momento e por quanto tempo.
  - Kernel pode gerenciar o escalonamento por threads ou por processos.
- **Diferentes tipos de escalonadores**
  - Não-preemptivo e Preemptivo.
  - Algoritmos tradicionais: FIFO, Round Robin.
  - É essencial para aplicações modernas como servidores web, sistemas em tempo real e aplicações gráficas.
- **Próximos passos**
  - Explorar o conceito de escalonamento no capítulo 2.4 do livro de TANENBAUM[1] e no capítulo 3 do livro de Hailperin[2].
  - Implementar algoritmos de escalonamento.

[1] A. TANENBAUM. 2015. Sistemas Operacionais Modernos. 4a ed. Pearson Brasil

[2] M. Hailperin. 2019. Operating Systems and Middleware - Supporting Controlled Interaction. Revised edition 1.3.1.

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins

denis.mayr@puc-campinas.edu.br