PUC CAMPINAS

Revisão SO: Parte 2

Threads e Escalonamento

Implementação de Núcleo de Sistema Operacional

Prof. Dr. Denis M. L. Martins

Engenharia de Computação: 5° Semestre

Introdução

Objetivos de Aprendizado

- Definir o que é uma thread e como ela difere de um processo.
- Explicar as vantagens do uso de threads em Sistemas Operacionais.
- Explicar o conceito de escalonamento em Sistemas Operacionais.
- Explicar os principais algoritmos de escalonamento.

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.

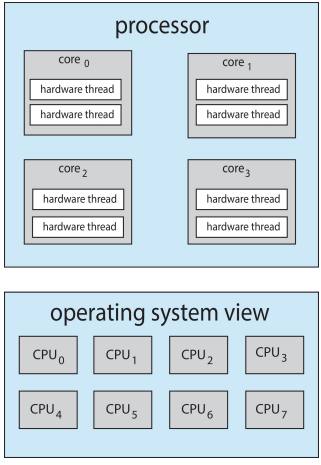


Figura 1: Processador multicore. Créditos: Silberschatz, Galvin and Gagne, 2018.

Multithread

Threads

- **Thread** = unidade de escalonamento do SO. Sequência independente de computações.
- Thread são mais leves e mais fáceis de criar e destruir do que processos.
- Permitem um novo nível de multitarefa:
  - Um mesmo programa (processo) pode realizar várias tarefas concorrentemente.
  - Processador de texto (e.g. MS Word): processa texto do teclado, verifica ortografia, salva continuamente o documento $\rightarrow$ **3 threads num mesmo processo**.
- Alto grau de *independência*: pense em funções diferentes no código.
- Concorrência e divisão de tempo de execução: em uma CPU de único núcleo, 3 threads pareceriam rodar em 3 núcleos fictícios com 1/3 da velocidade da CPU real.
- Núcleos individuais não estão se tornando significativamente mais rápidos (relembre a Lei de Moore). $\rightarrow$ É necessário utilizar programação paralela para aproveitar o hardware atual.

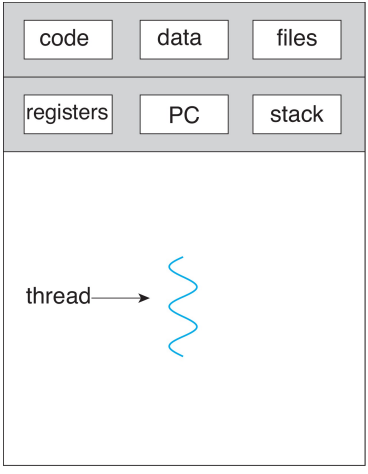
Threads

- **Thread** = unidade de escalonamento do SO. Sequência independente de computações.
- Thread são mais leves e mais fáceis de criar e destruir do que processos.
- Permitem um novo nível de multitarefa:
  - Um mesmo programa (processo) pode realizar várias tarefas concorrentemente.
  - Processador de texto (e.g. MS Word): processa texto do teclado, verifica ortografia, salva continuamente o documento $\rightarrow$ **3 threads num mesmo processo**.
- Alto grau de *independência*: pense em funções diferentes no código.
- Concorrência e divisão de tempo de execução: em uma CPU de único núcleo, 3 threads pareceriam rodar em 3 núcleos fictícios com 1/3 da velocidade da CPU real.
- Núcleos individuais não estão se tornando significativamente mais rápidos (relembre a Lei de Moore). $\rightarrow$ É necessário utilizar programação paralela para aproveitar o hardware atual.

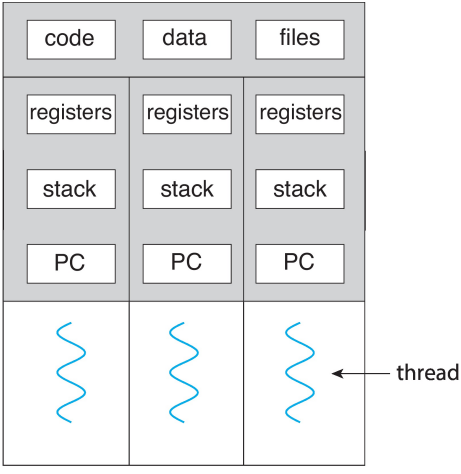
Threads

- **Thread** = unidade de escalonamento do SO. Sequência independente de computações.
- Thread são mais leves e mais fáceis de criar e destruir do que processos.
- Permitem um novo nível de multitarefa:
  - Um mesmo programa (processo) pode realizar várias tarefas concorrentemente.
  - Processador de texto (e.g. MS Word): processa texto do teclado, verifica ortografia, salva continuamente o documento $\rightarrow$ **3 threads num mesmo processo**.
- Alto grau de *independência*: pense em funções diferentes no código.
- Concorrência e divisão de tempo de execução: em uma CPU de único núcleo, 3 threads pareceriam rodar em 3 núcleos fictícios com 1/3 da velocidade da CPU real.
- Núcleos individuais não estão se tornando significativamente mais rápidos (relembre a Lei de Moore). $\rightarrow$ É necessário utilizar programação paralela para aproveitar o hardware atual.

Threads e Processos



single-threaded process



multithreaded process

Razões para usar Threads

- **Utilização de Recursos:**
  - A programação multithreaded mantém o hardware ocupado, evitando desperdício de recursos.
  - Quando uma thread está bloqueada (ex: esperando I/O ou eventos externos), outras podem continuar executando.
  - Distribuir tarefas entre múltiplas threads e núcleos de CPU aumenta a eficiência e acelera a execução.
- **Melhoria na Responsividade:**
  - Threads dedicadas a eventos externos garantem respostas rápidas.
  - Essencial para sistemas em tempo real, como jogos e servidores.

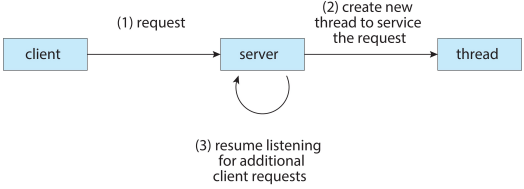


Figura 2: Aplicação de threads em um servidor web. Créditos: Silberschatz, Galvin and Gagne, 2018.

Execução de Threads

- Relembre a diferença entre **Concorrência** e **Paralelismo** → paradigmas de execução.
- Cada thread tem sua própria *stack*.
- Uma thread pode ler, escrever ou até mesmo apagar a *stack* de um outro thread.
- A execução de múltiplas threads torna-se mais complexa quando elas precisam interagir entre si (problema de sincronização), e.g., thread escreve dados na memória e outra os lê.

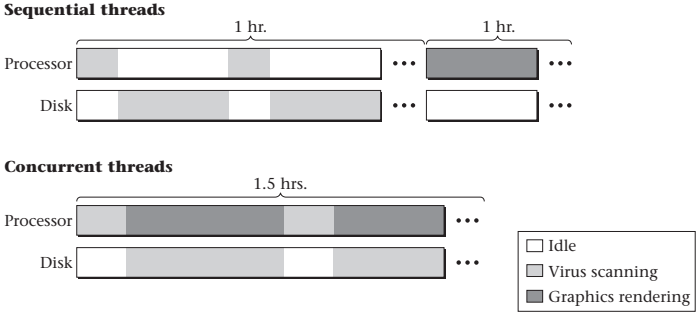


Figura 3: Execução Intercalada de Threads para otimizar a utilização de recursos. Créditos: Max Hailperin.

Mudança de Contexto

- SO escalona threads para execução (usando, e.g., técnicas de time-slicing $\rightarrow$ aula futura).
- Semelhante ao que ocorre com processos, escalonamento de threads involve **mudança de contexto**.
  - Remover a thread atual da CPU: salvar o estado da computação $\rightarrow$ Thread Control Block (TCB), semelhante ao PCB (aula anterior).
  - Despachar a próxima thread para a CPU: recuperar o estado da computação.
- Mais leve que escalonar processos.
  - Todas as threads compartilham os mesmos registradores da CPU.
  - Cada thread tem o seu *Program Counter* (PC) e um *Stack Pointer* (SP).

Mudança de Contexto

- SO escalona threads para execução (usando, e.g., técnicas de time-slicing $\rightarrow$ aula futura).
- Semelhante ao que ocorre com processos, escalonamento de threads involve **mudança de contexto**.
  - Remover a thread atual da CPU: salvar o estado da computação $\rightarrow$ Thread Control Block (TCB), semelhante ao PCB (aula anterior).
  - Despachar a próxima thread para a CPU: recuperar o estado da computação.
- Mais leve que escalonar processos.
  - Todas as threads compartilham os mesmos registradores da CPU.
  - Cada thread tem o seu *Program Counter* (PC) e um *Stack Pointer* (SP).

Mudança de Contexto

- SO escalona threads para execução (usando, e.g., técnicas de time-slicing $\rightarrow$ aula futura).
- Semelhante ao que ocorre com processos, escalonamento de threads involve **mudança de contexto**.
  - Remover a thread atual da CPU: salvar o estado da computação $\rightarrow$ Thread Control Block (TCB), semelhante ao PCB (aula anterior).
  - Despachar a próxima thread para a CPU: recuperar o estado da computação.
- Mais leve que escalonar processos.
  - Todas as threads compartilham os mesmos registradores da CPU.
  - Cada thread tem o seu *Program Counter* (PC) e um *Stack Pointer* (SP).

Mudança de Contexto

- SO escalona threads para execução (usando, e.g., técnicas de time-slicing $\rightarrow$ aula futura).
- Semelhante ao que ocorre com processos, escalonamento de threads involve **mudança de contexto**.
  - Remover a thread atual da CPU: salvar o estado da computação $\rightarrow$ Thread Control Block (TCB), semelhante ao PCB (aula anterior).
  - Despachar a próxima thread para a CPU: recuperar o estado da computação.
- Mais leve que escalonar processos.
  - Todas as threads compartilham os mesmos registradores da CPU.
  - Cada thread tem o seu *Program Counter* (PC) e um *Stack Pointer* (SP).

Biblioteca de Threads

- Objetivo: prover ao programador uma API para criação e gerenciamento de threads.
- Biblioteca pode ser implementada inteiramente em espaço do usuário ou pode estar a nível de kernel.
- Exemplo de biblioteca: PThreads em C.
  - Segue o padrão da API POSIX (IEEE 1003.1c). Comum em UNIX (Linux e macOS)
  - API especifica o comportamento da biblioteca de threads, mas a implementação é de decisão do desenvolvedor da biblioteca.

Função da API	Descrição
`pthread_create`	Cria uma nova thread e a inicia na função especificada.
`pthread_exit`	Termina a execução de uma thread e libera seus recursos
`pthread_join`	Aguarda a conclusão de uma thread específica.
`pthread_attr_inti`	Cria e inicializa a estrutura de atributos da thread.

Exemplo de threads em PThreads

```
#include <pthread.h>
#include <stdio.h>

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   /* set the default attributes of the thread */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid, &attr, runner, argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}
```

Exemplo de threads em PThreads (cont.)

```
/* The thread will execute in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Exemplo de threads em PThreads (cont.)

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Escalonamento

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.

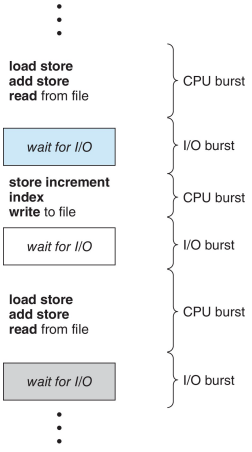


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

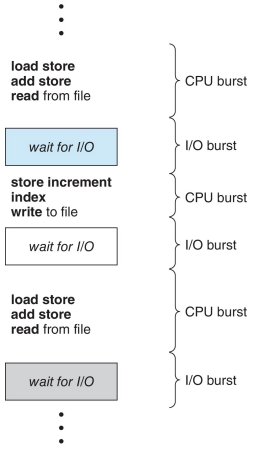


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

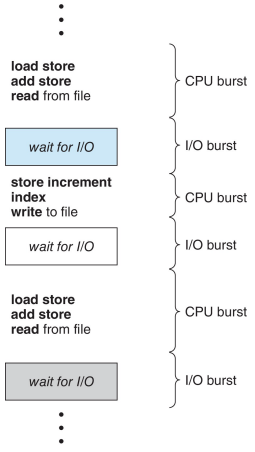


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

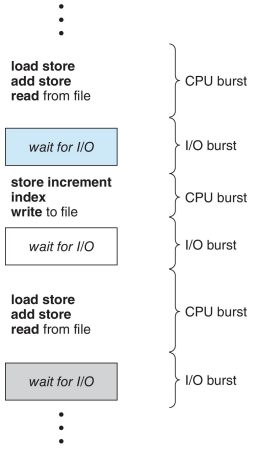


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

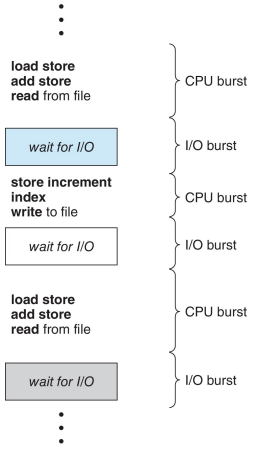


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

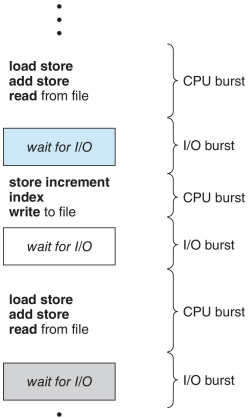


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

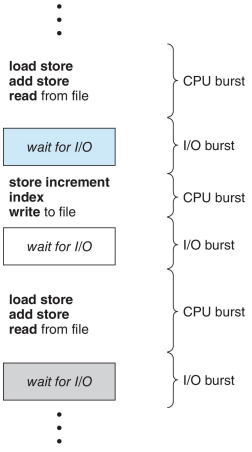


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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.

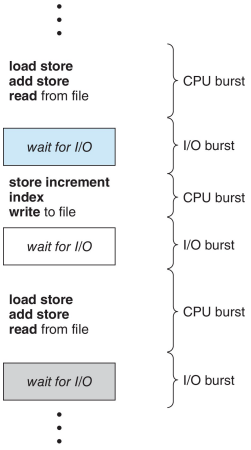


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

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)

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.

Escalonamento First-Come, First-Served (FCFS)

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 First-Come, First-Served (FCFS)

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 First-Come, First-Served (FCFS)

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 Shortest Job First (SJF)

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:

P_4	P_1	P_3	P_2

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

Escalonamento Round Robin (RR)

Considere o cenário abaixo:

Processo	Tempo de Chegada	Tempo de Serviço
P_1	0	24
P_2	1	3
P_3	2	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: $^{(6+4+7)}/_{3} = 5.66\ ms$.

Fechamento e Perspectivas

Fechamento e Perspectivas

- **Threads**
  - Uma thread é a menor unidade de execução dentro de um processo.
  - Diferentes threads de um mesmo processo compartilham memória e recursos.
- **Escalonamento**
  - Tempo compartilhado: escalonador escolhe qual processo ou thread usa a CPU em um determinado momento e por quanto tempo.
  - Diferentes algoritmos de escalonamento impactam o desempenho do sistema de maneiras distintas.
  - Round Robin melhora a responsividade, enquanto SJF minimiza o tempo médio de espera.
- **Próximos Passos**
  - Ler seção 2.4 (Escalonamento) do livro *TANENBAUM, A.; Sistemas Operacionais Modernos. 4a ed. Pearson Brasil, 2015* para revisar os conceitos.
  - Explorar sincronização: condições de corrida, semáforos, mutexes e deadlocks.

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins

denis.mayr@puc-campinas.edu.br