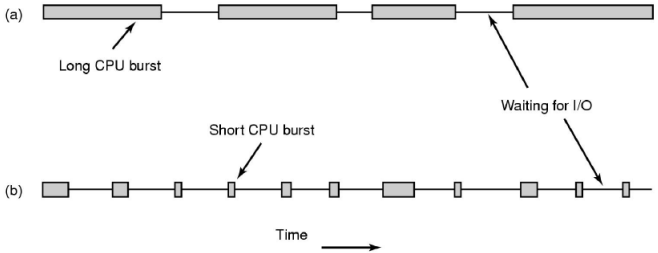
Engenharia de Computação
ECM225 – Sistemas Operacionais
Agendamento de processos

Slides da disciplina ECM225 – Sistemas Operacionais
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia
Prof. Marco Antonio Furlan de Souza
marco.furlan@maua.br

- **Para que agendar processos**?
  - Porque **sistemas operacionais** precisam **executar** diversos **processos**. O **agendador escolhe** o **próximo processo** a ser executado, **de acordo com** alguma **política**.
- **Agendador**
  - É o **componente** do **sistema operacional** (também um processo!) que se encarrega de **escolher o próximo processo** a ser executado.
- **Algoritmo de agendamento**
  - É a **algoritmo** implementado no agendador que **implementa a política de agendamento**. **Depende** das **necessidades** do **sistema operacional** (lote, interativo ou tempo real).

Comportamento dos processos

- Praticamente todos os processos alternam picos de computação com requisições de E/S:



(a) **Processo limitado por CPU** (*CPU bound*)

(b) **Processo limitado por E/S** (*I/O bound*)

- Como as operações de E/S são muito mais lentas que na CPU, o agendador deve sempre que possível escolher um processo limitado por E/S para manter os dispositivos em operação.

- **Para todos os sistemas**
  - **Equidade** (*fairness*): **dar** a todos os **processo**s uma **fatia justa de tempo da CPU**;
  - **Aplicação de política**: garantir que a **política definida** está de fato sendo **aplicada**;
  - **Equilíbrio**: **manter** todas as **partes do sistema ocupadas**.
- **Sistemas em lote** (*batch*)
  - **Taxa de execução**: **maximizar** a execução de **jobs por hora**;
  - **Tempo de virada** (*turnaround time*): **minimizar o tempo** entre a **submissão de um job e** sua **terminação**;
  - **Utilização de CPU**: manter a **CPU ocupada** o tempo todo.

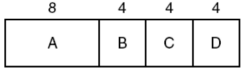
- **Sistemas interativos**
  - **Tempo de resposta**: **responder** às requisições **rapidamente**;
  - **Proporcionalidade**: **atender** às **expectativas** de execução dos **usuários**.
- **Sistemas em tempo real**
  - **Atender a prazos de computação**: **evitar perdas de dados** por não conseguir atendar às restrições de tempo;
  - **Previsibilidade**: **evitar degradação da qualidade de serviço** em alguns tipos de computação (exemplo: processamento de multimídia).

Algoritmos de agendamento

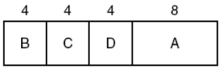
- **Sistemas em lote**
  - Primeiro a chegar, Primeiro a ser servido;
  - Tarefa mais curta primeiro;
  - Tempo restante mais curto em seguida.
- **Sistemas interativos**
  - Agendamento circular (*Round-Robin*);
  - Agendamento por prioridades;
  - Múltiplas filas;
  - Processo mais curto em seguida;
  - Agendamento garantido;
  - Agendamento por loteria;
  - Agendamento por fatia justa.
- **Agendamento em sistemas de tempo real**

- **Primeiro a chegar, Primeiro a ser servido**
  - Os **processos** são **atribuídos** à **CPU** na **ordem** que eles a **requisitam**;
  - Um **fila única de processos** é utilizada;
  - Quando um **processo está em execução** na **CPU**, os **demais aguardam** na fila;
  - Se um **processo em execução** é **interrompido** por **operações** de **E/S**, o **próximo** da **fila** é escolhido para **executar** e quando o **processo interrompido** se torna **ativo** novamente, é **transferido** para o **fim da fila**;
  - Quando um **processo termina sua execução na CPU**, o **processo** que está no **início** da **fila** é **selecionado** para **executar** na CPU;
  - **Vantagem**: algoritmo de **simples** entendimento e implementação.
  - **Desvantagem**: o **processos limitados** por **E/S** são **prejudicados** em **desempenho** em relação aos processos limitados por CPU.

- **Tarefa mais curta primeiro**
  - O **agendador** escolhe o **job mais curto** (em **tempo**) de uma fila de processos a executar;
  - **Exemplo**. Ordem original:



    - Neste caso, tem-se os seguintes **tempos de virada**: A (8), B (8+4=12), C (8+4+4=16) e D (8+4+4 +4 = 20). A **média** de **tempo de virada** é (8+12+16+20)/4 = 14.
  - Ordenado pela tarefa mais curta:



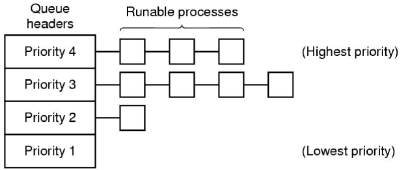
    - Neste caso, tem-se os seguintes tempos de virada: B (4), C (4+4=8), D (4+4+4=12) e A (4+4+4+8=20) A média de tempo de virada é (4+8+12+20)/4 = 11. Portanto, **neste caso é o melhor**. Este **algoritmo é ótimo** quando **todos os processos** estão **disponíveis simultaneamente**.

- **Tempo restante mais curto em seguida**
  - É uma **versão preemptiva** do algoritmo *Tarefa mais curta primeiro*;
  - Se um **novo processo está disponível** e possui um **tempo execução menor do que** o **tempo restante de execução de um processo** que está **atualmente em execução**, o **processo em execução** é **suspenso** e **este novo processo** é **escolhido** para **executar**;
  - Assim **novos jobs** tem a **chance de ser rapidamente executados**.

Agendamento circular (*Round-Robin*)

- A **cada processo** é **atribuído** um **intervalo** de **tempo** – *quantum* – no qual o **processo pode executar** na CPU;
- Se um **processo** está **executando** na CPU e **chega ao final do seu quantum**, o **agendador** se **antecipa** e **move** este **processo** para o **final** da fila e então **seleciona** um processo do **início da fila** e o coloca na CPU para **execução**;
- Se um **processo** em **execução na CPU termina ou bloqueia** (E/S) **antes** do *quantum* **esgotar**, o **agendador** o **removerá ou**, no segundo caso, o **moverá para o final da fila** e então **selecionará** um processo no **início da fila** para **executar** na CPU;
- Sobre a **escolha do *quantum***: **muito pequeno**, causa **comutação excessiva** em processos e **reduz a eficiência da CPU**; **muito grande**, causa **baixa resposta para requisições** interativas **curtas**. Valores típicos: 20-50ms.

- **Agendamento por prioridades**
  - Nesta abordagem, **adiciona-se uma prioridade** a cada **processo** e o **processo com maior prioridade** é levado à **execução na CPU** pelo agendador (na realidade podem haver vários processos com mesma prioridade);
  - Pra **prevenir** que um **processo de alta prioridade** execute **indefinitivamente**, o **agendador pode reduzir sua prioridade**, causando a **comutação para outro processo**, se sua prioridade for maior;
  - **Prioridades** podem ser **estáticas** ou **dinâmicas**;
  - **Pode-se aplicar Round-Robin** com **processos de mesma prioridade**.



- **Múltiplas filas**
  - Utiliza **múltiplas filas** de **processos** que representam "**categorias**" de **processos**, **organizados** de acordo com *quanta* que o **agendador atribui**;
  - Por **exemplo**, na **primeira fila** estão **processos** que **utilizarão** no **máximo** 1 *quantum*; na **segunda**, 2 *quanta*; na **terceira**, 4 *quanta*; e assim por diante;
  - Um **processo que necessite** de 100 *quanta* para **executar** começaria na **primeira fila**, **após comutação** pelo **agendador**, **depois** de **gastar** 1 *quantum*, **executaria** na **segunda fila** até consumir 2 *quanta* e, **assim por diante**;
  - **Então** o **número de filas** que este **processo entra**, **resulta na quantidade** de **tempo necessário** para sua **execução**: 1 + 2 + 4 + 8 + 16 + 32 + 37 (o processo precisa de 100 quanta, então não precisa consumir 64 na sétima fila);
  - Corresponde, **portanto à 7 comutações** – é um **método que tende a reduzir** o número de **comutações**.

- **Processo mais curto em seguida**
  - **Similar** ao algoritmo *Tarefa mais curta primeiro*;
  - No entanto, em **sistemas interativos**, utilizam-se **técnicas para ajustar** o **tempo de execução dos processos** para **melhorar o agendamento**;
  - Desse modo, faz-se uma **estimativa baseada no passado** do **tempo de execução** dos processos;
  - **Supor** que o **tempo medido** de um **processo** é T_0 e que a**pós sua próxima execução resultou** em T_1. **Pode-se definir** uma **estimativa** para uma **próxima execução** como $aT_0 + (1-a)T_1,\ (0 \leq a \leq 1)$;
  - No mais, **escolhe-se para execução o processo** que possuir **menor estimativa de tempo**.

- **Agendamento garantido**
  - **Promessa**: em um s**istema** com n **processos em execução**, **cada processo** deveria receber $1/n$ do **tempo da CPU**;
  - Para **manter esta promessa**, o **sistema** deve **acompanhar quanto** da **CPU** cada **processo utilizou desde** sua **criação**;
  - **Computa-se então a quantidade de CPU** que cada **processo tem direito** – o **tempo desde** sua **criação dividido** por n;
  - Como a **quantidade de tempo de CPU** que cada **processo** já **consumiu é conhecido**, pode-se **calcular a relação** do **tempo atual de CPU consumida** pelo **tempo de CPU** de **direito ao processo**.

- **Agendamento garantido**
  - Uma **relação de 0.5** significa **que ele somente teve metade do que deveria ter** e uma **relação de 2.0** significa que um processo **teve duas vezes mais tempo** do que ele teria direito;
  - O **algoritmo escolhe para executar** o **processo** que possui o **valor mais baixo desta relação**.

- **Agendamento por loteria**
  - É uma **forma similar**, porém mais simples, de *Agendamento garantido*;
  - A ideia é **distribuir aos processos** “tickets de loteria” que **representam diversos recursos** do sistema, tal como o **tempo de CPU**;
  - Sempre que uma **decisão de agendamento** tem que ser feita, um **ticket de loteria** é **escolhido aleatoriamente** e o **processo** que tem este **ticket recebe o recurso**;
  - Por exemplo, **aplicado ao agendamento de CPU**, o **sistema poderia realizar um sorteio** 50 vezes por segundo e **sortear um ganhador com 20ms** de **tempo de CPU** como prêmio.

- **Agendamento por fatia justa**
  - Trata-se de uma técnica que tenta **balancear o agendamento existente** (por exemplo, *Agendamento Circular*) **de acordo** com **alguma promessa** ou de algum direito de execução;
  - Por exemplo, **considerar um sistema** com **dois usuários**, **cada** um **com a promessa** de **utilizar 50%** da **CPU**;
  - O **usuário 1** tem **quatro processos**, A, B, C, e D, e o **usuário 2** tem somente **um processo**, E;
  - Um **agendamento justo** com **Round-Robin poderia ser**: $A\ E\ B\ E\ C\ E\ D\ E\ A\ E\ B\ E\ C\ E\ D\ E$ ...;
  - **Por outro lado**, **se o usuário 1** tem direito a **duas vezes mais CPU** que o **usuário 2**, pode-se ter **este agendamento**: $A\ B\ E\ C\ D\ E\ A\ B\ E\ C\ D\ E$ ...;
  - Existem muitos outros esquemas que se pode utilizar.

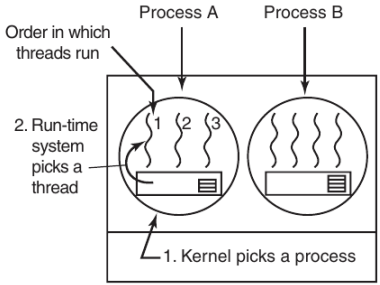
- É aquele em que o **tempo** tem um **papel essencial**;
- São geralmente **categorizados** como **tempo real crítico**, significando que há **prazos absolutos que devem ser cumpridos** e **tempo real não crítico**, significando que **descumprir um prazo** ocasional é **indesejável**, mas tolerável;
- O comportamento em **tempo real** é **conseguido dividindo o programa** em uma **série de processos**, todos **previsíveis** e **conhecidos antecipadamente**;
- Esses **processos** geralmente têm **vida curta** e podem ser **concluídos** em menos de um segundo;
- Quando um **evento externo** é detectado, cabe ao **agendador programar** os **processos** de modo que todos os **prazos sejam atendidos**.

- Os **eventos** podem se **periódicos** e **aperiódicos**. Por exemplo, se há m **eventos periódicos** e o **evento** i **ocorre** com **período** P_i e **exige** C_i **segundos** da **CPU** para **lidar com cada evento**, então a carga só pode ser tratada se $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$;
- Neste caso, **o termo** $\frac{C_i}{P_i}$ representa um **fator de utilização** da tarefa e a soma delas deve ser menor que 100% do uso de um processador;
- Um **sistema de tempo real** que **atende a esse critério** é dito **agendável**;
- **Exemplo**: em um **sistema de tempo real** não crítico com **três eventos periódicos**, com períodos de 100, 200 e 500ms, se esses eventos exigem 50, 30 e 100ms de tempo da CPU, respectivamente, o sistema é agendável, pois $0.5 + 0.15 + 0.2 < 1$;
- Algoritmos de **escalonamento de tempo real** podem ser **estáticos** ou **dinâmicos**.

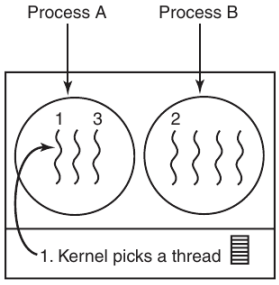
Algoritmos de threads

- O agendamento de threads **depende do tipo de thread** que se está considerando – **threads de usuário** ou **threads de kernel**;
- Nas **threads de usuário**, o **kernel desconhece essas threads** e **agenda os processos como já visto** – pode **utilizar qualquer um dos métodos apresentados** – **depende** de uma **biblioteca de threads**;
- Nas **threads de kernel**, o **kernel seleciona uma thread** de um dos **processos para executar**, **independentemente** se o **processo desta thread** está em **execução ou não**. Neste caso **o agendamento das threads** é **independente** do **agendamento dos processos**.

- **Comparação entre agendamento de threads de usuário e threads de kernel**



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



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

Referências bibliográficas

[1] SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. 8. ed. [s.l.] John Wiley & Sons, Inc., 2009.

[2] TANENBAUM, A. S.; BOS, H. **Modern operating systems**. 4. ed. [s.l.] Pearson, 2015.

[3] TANENBAUM, A. S. **Structured Computer Organization**. 5. ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2006.

[4] **Operating System Concepts**. Disponível em: <https://www.cs.rutgers.edu/~pxk/416/notes/>. Acesso em: 11 mar. 2021.

[5] **Operating Systems: Three Easy Pieces**. Disponível em: <https://pages.cs.wisc.edu/~remzi/OSTEP/>. Acesso em: 11 mar. 2021.