

Escalonamento de Processos

SSC-140 - Sistemas Operacionais I

Prof. Denis M. L. Martins

Instituto de Ciências Matemáticas e de Computação

- 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.

Material de Referência desta Aula

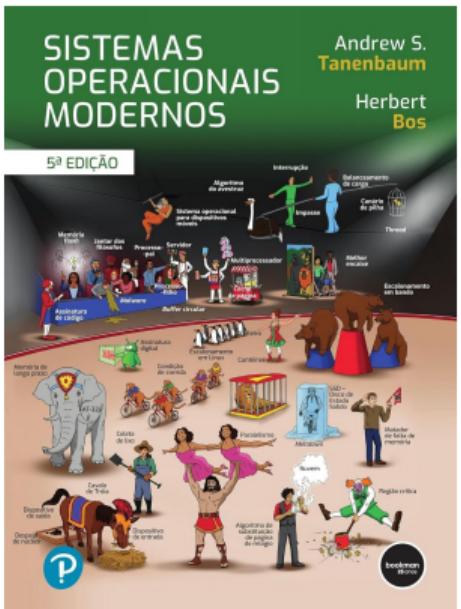


Figura 1: Seções 2.4, 10.3.4 e 11.4.3

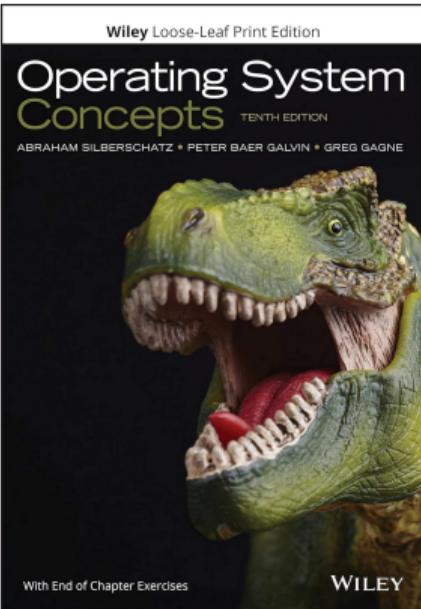


Figura 2: Capítulo 5 e Seções 20.5 e 21.3.4

Relembrando Conceitos

- Processo ≈ programa em execução.
 - ▶ **Programa:** entidade **passiva** guardada no disco (**arquivo executável**).
 - ▶ **Processo:** carregado em memória.
- **Chamadas de sistema** para criação (fork) e término (exit ou kill) de processos
- SO mantém uma **filas de processos:**
 - ▶ Prontos (*ready queue*)
 - ▶ Em espera (*wait queue*)

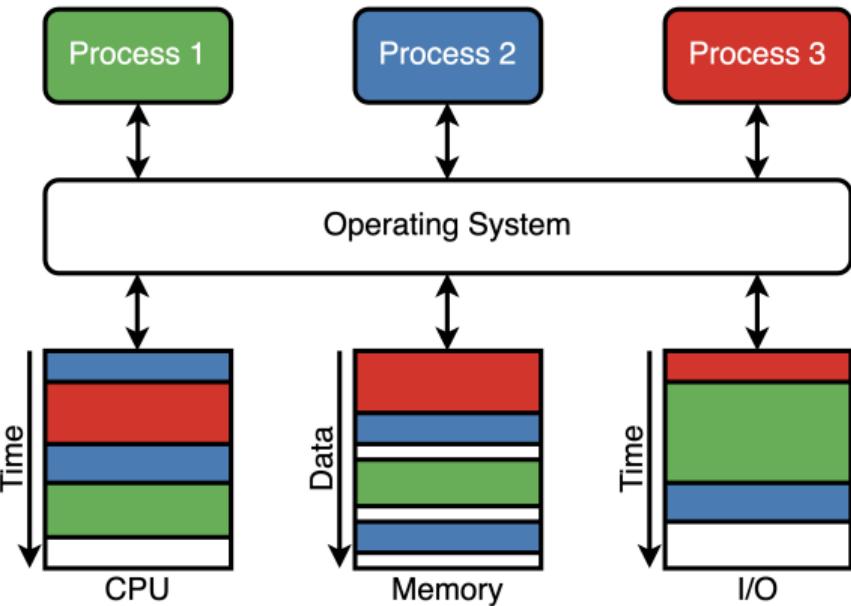


Figura 3: Processos no SO. Créditos: OS OER.

Utilização de CPU

- **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.
- Em ambiente **multiprogramado**: Múltiplos processos **competindo** pela CPU ao mesmo tempo.
 - ▶ CPU muda de um processo para outro rapidamente (pseudoparalelismo).
 - ▶ Objetivo: Maximizar a utilização da CPU.

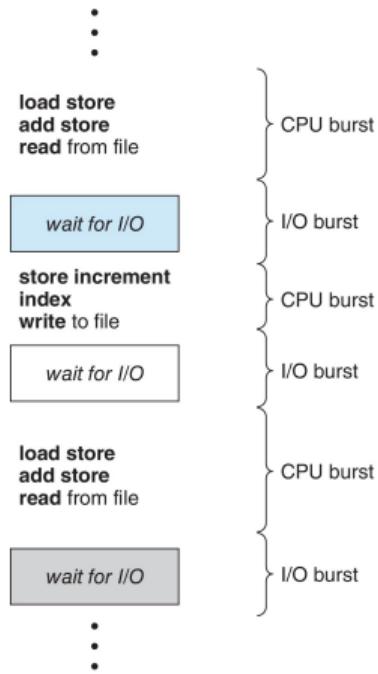


Figura 4: Surtos de CPU e de I/O. Créditos: Silberschatz, Galvin and Gagne, 2018.

Escalonamento de CPU

- **Escalonador** é o componente do SO que decide qual processo será executado na CPU.
- Utiliza um **algoritmo de escalonamento**.
- **Alocar** a CPU para um processo: executar instruções do processo
- Chaveamento de processos na CPU é custoso: **mudança de contexto**.

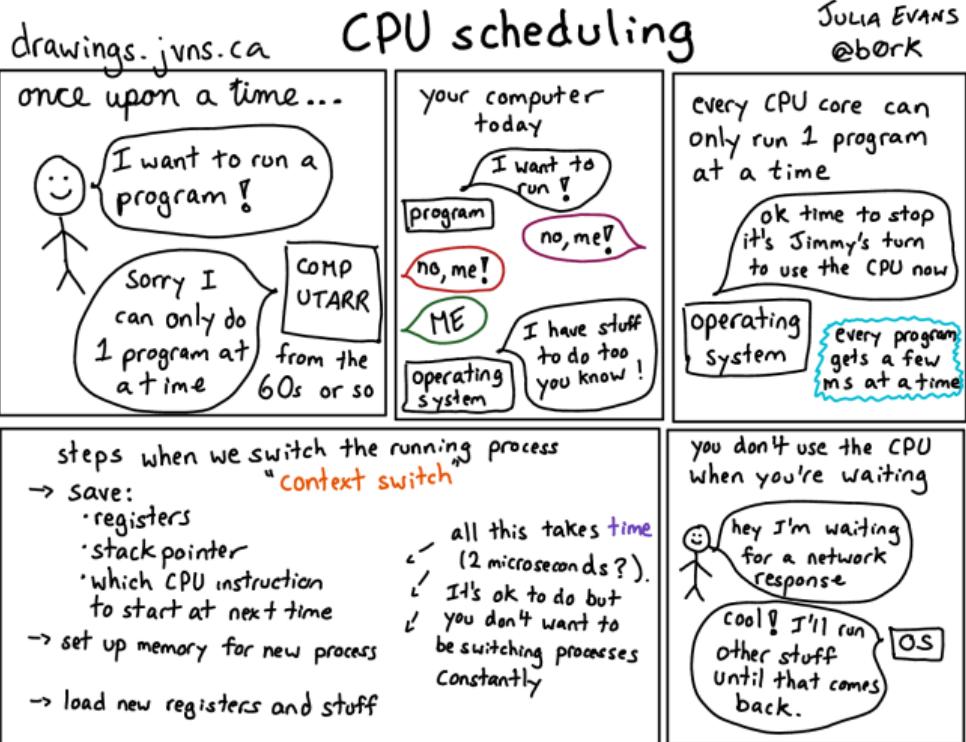


Figura 5: Escalonamento de CPU. Créditos: Julia Evans.

Mudança de Contexto

- Interromper o processo atual
- Mudar de modo usuário para modo núcleo (acesso a conjunto de instruções privilegiadas)
- Salvar o contexto da CPU (conteúdo dos registradores) em memória principal
- O contexto de um processo é armazenado no Bloco de Controle de Processo (*Process Control Block - PCB*).
- Carregar o estado do novo processo.
- Potencialmente refazer cache (tema de aula futura).
- O tempo de mudança de contexto é considerado **sobrecarga**.

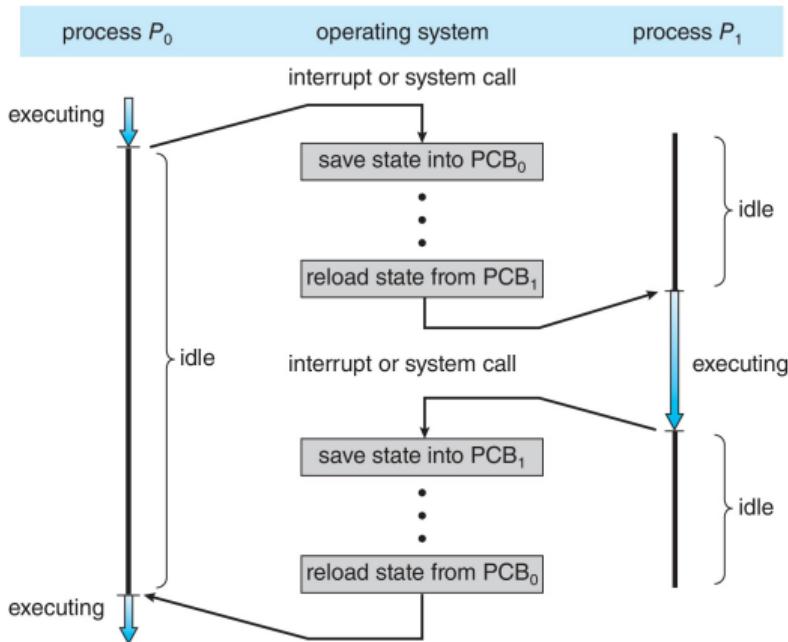


Figura 6: Alternância da CPU de um processo para outro. Créditos: Silberschatz, Galvin and Gagne, 2018.

- **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) → 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: ↑ vazão, ↓ tempo de resposta.

- **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) → 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: ↑ vazão, ↓ tempo de resposta.

Escalonador (*Scheduler*)

Planejamento: Seleciona qual processo vai usar a CPU.

Despachante (*Dispatcher*)

Alocação: Oferece controle da CPU para o processo. Implementa a troca de contextos em baixo nível.

Latência do Despachante

Tempo que o Despachante para um(a) processo (re)comece outro(a).

- Escalonamento em **Lote**: Projetado para tarefas que não exigem interação do usuário durante sua execução.
 - ▶ Tarefas são agrupadas em "lotes" e executadas sequencialmente.
 - ▶ Foco na maximização do throughput (número de tarefas concluídas por unidade de tempo).
- Escalonamento **Interativo**: Projetado para sistemas que precisam responder rapidamente às interações do usuário.
 - ▶ Prioriza o tempo de resposta.
- Escalonamento em **Tempo Real**: Projetado para sistemas onde as tarefas devem ser concluídas dentro de prazos rigorosos e predefinidos.
 - ▶ A falha em cumprir um prazo pode ter consequências graves (ex: perda de dados, danos físicos).
 - ▶ Soft x Hard.

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)

CPU alocada por ordem de chegada

Considere o cenário abaixo:

Processo	Tempo de Duração
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	
0	24	27	30

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	
0	3	6	30

Tempo de espera: $P_1 = 6; P_2 = 0; P_3 = 3$

Tempo médio de espera: $(6+0+3)/3 = 3$

First-Come, First-Served (FCFS)

CPU alocada por ordem de chegada

Considere o cenário abaixo:

Processo	Tempo de Duração
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	
0	24	27	30

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	
0	3	6	30

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)

CPU alocada ao processo que tem menor tempo de duração

Considere o cenário abaixo:

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

Exemplo SJF. Ordem de chegada: P_1, P_2, P_3, P_4

No escalonamento SJF, teríamos:

	P_4		P_1		P_3			P_2		
0	3	9					16			24

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

Shortest Remaining Time First (SRTF)

Preemptivo: CPU alocada ao processo que tem menor tempo de duração restante

Considere o cenário abaixo:

Processo	Tempo de Chegada	Tempo de Duração
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$	$ $
0	1	5	10	17	26

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

Round Robin (RR)

Preemptivo: processo executa por intervalo de tempo (*quantum*) q

Considere o cenário abaixo:

Processo	Tempo de Duração
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	
0	4	7	10	14	18	22	26	30

P_1 sai da CPU no tempo 4 e retorna no tempo 10. Então, o seu tempo espera é $10 - 4 = 6$

P_2 espera por 4 e P_3 por 7

Tempo médio de espera: $(6+4+7)/3 = 5.66$

- Uma prioridade (número inteiro) associada a cada processo.
- A CPU é alocada ao processo com maior prioridade (menor número = maior prioridade).
- Um processo de alta prioridade pode interromper a execução de um processo de baixa prioridade.
- Processos de mesma prioridade são escalonados, por exemplo, via RR
- **Problema: Starvation (Inanição)** – processos de baixa prioridade podem nunca executar se houver um fluxo contínuo de processos de alta prioridade.
- **Solução: Aging (Envelhecimento)** – aumentar gradativamente a prioridade de um processo ao longo do tempo, para garantir que mesmo os processos de baixa prioridade eventualmente recebam sua vez de execução.

Escalonamento em Linux

- Padrão desde a versão 2.6.23 do Kernel Linux.
- Baseado em classes de escalonamento, de acordo com as demandas de processamento.
- Multiple Feedback Queues
 - ▶ SCHED_DEADLINE: tarefas de tempo real com prazos.
 - ▶ SCHED_FIFO
 - ▶ SCHED_RR
 - ▶ SCHED_NORMAL: CFS
 - ▶ SCHED_BATCH: tarefas CPU-bound de baixa prioridade.
 - ▶ SCHED_IDLE: só recebe CPU se não houver tarefa ativa.
- Estrutura de dados: Árvore Rubro-Negras (red-black trees) baseada em vruntime, com valores menores na esquerda da árvore. **Demo:** CFS visualizer

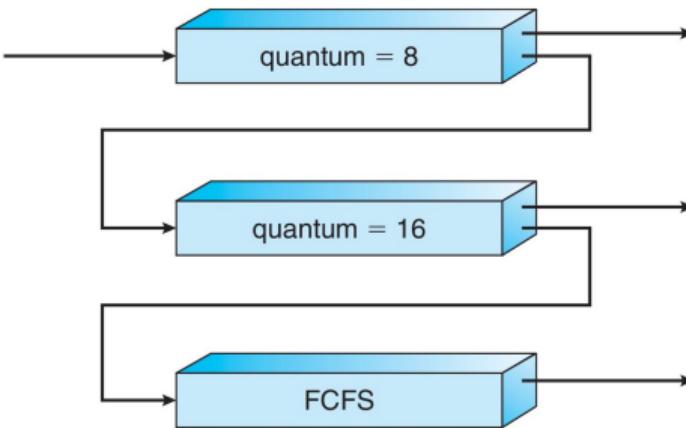


Figura 7: Multiple Feedback Queues.
Créditos: Silberschatz, Galvin and Gagne,
2018.

- Escalonamento preemptivo baseado em prioridade.
- O despachador (*dispatcher*) é o módulo do núcleo responsável pelo escalonamento escalonador.
- Thread executa até que (1) seja bloqueado, (2) utilize sua fatia de tempo ou (3) sofra preempção por um thread de maior prioridade.
- Implementa um esquema de prioridade de 32 níveis para threads.
- 6 classes de prioridade para processos.
 - ▶ IDLE PRIORITY CLASS
 - ▶ BELOW NORMAL PRIORITY CLASS
 - ▶ NORMAL PRIORITY CLASS
 - ▶ ABOVE NORMAL PRIORITY CLASS
 - ▶ HIGH PRIORITY CLASS
 - ▶ REALTIME PRIORITY CLASS
- Pode ser modificada pela chamada de sistema SetPriorityClass ()

Fechamento e Perspectivas

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.

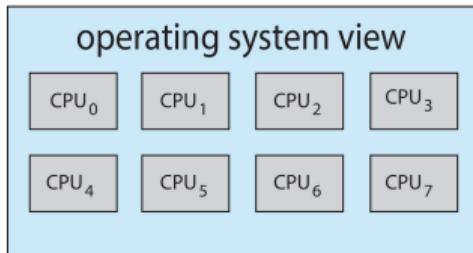
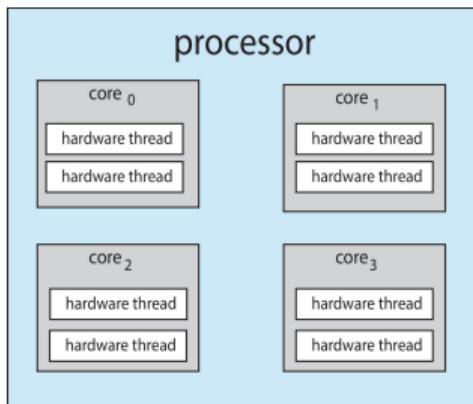


Figura 8: Processador multicore. Créditos:
Silberschatz, Galvin and Gagne, 2018.

- Escalonador precisa tomar decisões mais complicadas:
 - ▶ qual processo/thread executar?
 - ▶ em qual CPU?
 - ▶ como manter equilíbrio?
 - ▶ como manter cache?
- Como escalar processos em hardware diversos? GPUs, CPUs, FPGAs...
- **Balanceamento de carga:** Evitar que algumas CPUs fiquem sobrecarregadas enquanto outras ficam ociosas.
- **Escalonamento por Afinidade:** Restringir um processo ou thread para executar em um subconjunto de CPUs.
- Tema de aula futura.



(a) A sample model and an illustration of the effect of the schedule construct unroll

```

model = ensemble(...)
for t_depth1 = 0 to 2 step 1:
    T = getTree(model, t_depth1)
    for batch = 0 to BATCH_SIZE step 1:
        treePred = walkDecisionTree(T,
                                      input[batch]), unrollDepth = 1
        reduce(result[batch], treePred) <'+', 0.0>
reorder(tree, batch)
// Split tree loop so trees with
// equal depth are processed together
split(tree, t_depth1, t_depth2, 2)
// Unroll the tree walks
unrollWalk(t_depth1, 1)
unrollWalk(t_depth2, 2)
for t_depth2 = 2 to 4 step 1:
    T = getTree(model, t_depth2)
    for batch = 0 to BATCH_SIZE step 1:
        treePred = walkDecisionTree(T,
                                      input[batch]), unrollDepth = 2>
        reduce(result[batch], treePred) <'+', 0.0>
    
```

(b) A serial CPU schedule

```

// Split the trees into two-sets
tile(tree, t0, t1, 2)
reorder(batch, t1, t0)
split(t0, t0_depth1, t0_depth2, 2)
unrollWalk(t0_depth1, 1)
unrollWalk(t0_depth2, 2)
// Configure the GPU kernel dimensions
gpuDimension(batch, grid.x)
gpuDimension(t1, block.x)
par.for batch = 0 to 2 step 1 <grid.x>:
    par.for t1 = 0 to 2 step 1 <block.x>:
        for t0_depth1 = 0 to 2 step 2:
            T = getTree(model, t0_depth1 + t1)
            treePred = walkDecisionTree(T,
                                         input[batch]), unrollDepth = 1
            reduce(result[batch], treePred) <'+', 0.0>
        for t0_depth2 = 2 to 4 step 2:
            T = getTree(model, t0_depth2 + t1)
            treePred = walkDecisionTree(T,
                                         input[batch]), unrollDepth = 2>
            reduce(result[batch], treePred) <'+', 0.0>
    
```

(c) A GPU schedule

Figure 1. Figure 1a shows the model for the motivating example. Figure 1b and 1c show some possible schedules to compile the motivating example and the generated SILVANFORGE IR. The arrows connect entities in the schedule to related entities in the IR. Indices that go over trees are shown in orange and those that go over input rows are shown in green.

Figura 9: SilvanForge: A Schedule-Guided Retargetable Compiler for Decision Tree Inference.
Disponível em <https://dl.acm.org/doi/abs/10.1145/3694715.3695958>

● 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 TANENBAUM¹ e no capítulo 3 do livro de Hailperin².
- ▶ Implementar e simular algoritmos de escalonamento.

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

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

Suponha que um algoritmo de escalonamento favoreça os processos que utilizaram a menor quantidade de tempo de processador no passado recente. Por que este algoritmo favorecerá processos com uso intensivo de E/S (I/O-bound) e ainda assim não prejudicará permanentemente processos com uso intensivo de CPU (CPU-bound)?

Dúvidas e Discussão

Prof. Denis M. L. Martins

Muito Obrigado!