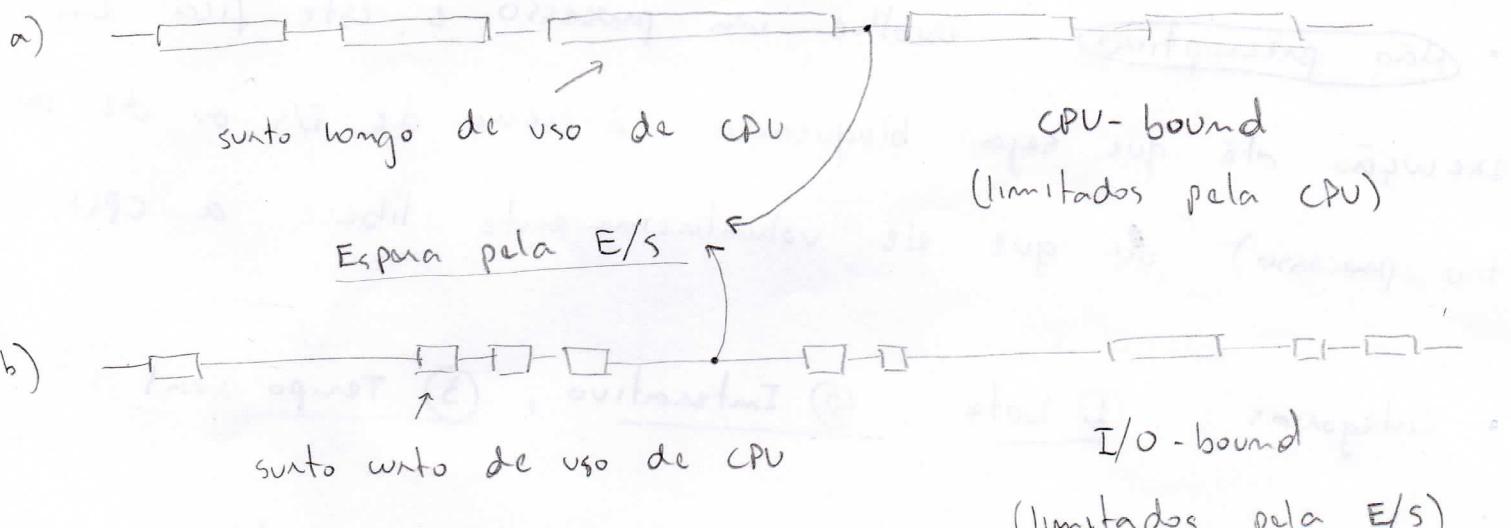


2.4 ESCALONAMENTO DE PROCESSO EM UM S.O.

O S.O. usa o escalonador, que utiliza o algoritmo de escalonamento:

→ Chavar processo é custoso, por isso o escalonador deve usar a CPU de modo eficiente.

Chavamento: modo usuário → modo núcleo



a) → Processo orientado à CPU

b) → Processo orientado à E/S

Se um hardware fornece interrupções periódicas a cada

X Hz, uma decisão de escalonamento pode ser tomada a cada interrupção, isto é, a cada k-ésima interrupção.

Os algoritmos de escalonamento podem ser divididos em duas categorias quanto ao modo com que tratam as interrupções:

- Premptivo - escolhe um processo e o deixa em execução por um tempo máximo fixado. Caso ainda esteja em execução ao final desse tempo o processo é suspenso e o escalonador escolherá outro processo para executar.
- Não preemptivo - escolhe um processo e este fica em execução até que seja bloqueado (à espera de E/S ou de outro processo) ou que de voluntariamente libere a CPU.
- Categorias: ① Lote; ② Interativo; ③ Tempo real

VAZÃO - número de tarefas por hora que o sistema termina.
Ex.: 50 tarefas por hora.

TEMPO DE RETORNO - indica quanto tempo, em média, o usuário tem que esperar pelo fim do trabalho.

parametros que interessam (sistemas em Lote)

TEMPO DE RESPOSTA (sistemas interativos) - tempo entre a emissão de um comando e a obtenção do resultado.

2.4.3) Escalonamento em sistemas interativos PREEMPTIVO

- Escalonamento por chaveamento circular (round-robin)
 - Um dos algoritmos mais antigos, simples, justos e amplamente usados.
 - A cada processo é atribuído um intervalo de tempo, chamado quantum, no qual ele é permitido executar.
 - Se, ao final do quantum, o processo ainda estiver executando, a CPU sofrerá preempção e será dada a outro processo.
 - Se o processo foi bloqueado ou terminou antes do quantum, a CPU será chaveada para outro processo.
 - Tempo razoável de um quantum: 20ms a 50ms.
- * Quanta = plural de quantum (ex.: Física Moderna)

TANENBAUM, Andrew. Sistemas Operacionais Modernos. 3. ed [S. l.]: Pearson, 2008. 653p.

TANENBAUM, Andrew; BOS, Herbert. Sistemas Operacionais Modernos. 4. ed. [S. l.]: Pearson, 2015. 758p.

PROJETO

(ordenamento) - soluções otimizadas seg. implementação.

Variáveis pedidas:

- * Tempo de "cliente" na fila Δt_1
- * Tempo de "cliente" em atendimento Δt_2
- * Tempo de "cliente" no sistema $\Delta T = \Delta t_1 + \Delta t_2$

Substituição:

cliente \rightarrow PROCESSO

atendimento \rightarrow USO DE CPU

\rightarrow Tempo médio de processo na fila.

\rightarrow Tempo médio de uso de CPU por processo.

\rightarrow Tempo médio de processo no sistema.

Escolher 3 de 5 métricas:

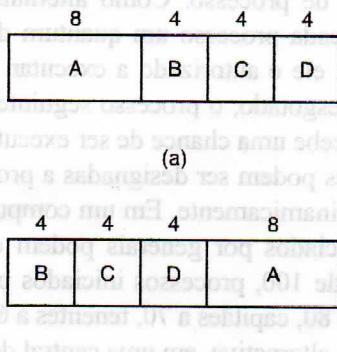
- Tempo médio de permanência no sistema
- Tempo médio de espera na fila.
- Tempo médio de atendimento.
- Média de clientes no sistema
- Média de clientes na fila.

bastante precisão quanto tempo levará para executar um lote de 1.000 solicitações, tendo em vista que um trabalho similar é realizado todos os dias. Quando há vários trabalhos igualmente importantes esperando na fila de entrada para serem iniciados, o escalonador escolhe a **tarefa mais curta primeiro (shortest job first)**. Observe a Figura 2.41. Nela vemos quatro tarefas *A*, *B*, *C* e *D* com tempos de execução de 8, 4, 4 e 4 minutos, respectivamente. Ao executá-las nessa ordem, o tempo de retorno para *A* é 8 minutos, para *B* é 12 minutos, para *C* é 16 minutos e para *D* é 20 minutos, resultando em uma média de 14 minutos.

Agora vamos considerar executar essas quatro tarefas usando o algoritmo *tarefa mais curta primeiro*, como mostrado na Figura 2.41(b). Os tempos de retorno são agora 4, 8, 12 e 20 minutos, resultando em uma média de 11 minutos. A tarefa mais curta primeiro é provavelmente uma ótima escolha. Considere o caso de quatro tarefas, com tempos de execução de *a*, *b*, *c* e *d*, respectivamente. A primeira tarefa termina no tempo *a*, a segunda no tempo *a + b*, e assim por diante. O tempo de retorno médio é $(4a + 3b + 2c + d)/4$. Fica claro que *a* contribui mais para a média do que os outros tempos, logo deve ser a tarefa mais curta, com *b* em seguida, então *c* e finalmente *d* como o mais longo, visto que ele afeta apenas seu próprio tempo de retorno. O mesmo argumento aplica-se igualmente bem a qualquer número de tarefas.

Vale a pena destacar que a *tarefa mais curta primeiro* é ótima apenas quando todas as tarefas estão disponíveis simultaneamente. Como um contraexemplo, considere cinco tarefas, *A* a *E*, com tempos de execução de 2, 4, 1, 1 e 1, respectivamente. Seus tempos de

FIGURA 2.41 Um exemplo do escalonamento tarefa mais curta primeiro. (a) Executando quatro tarefas na ordem original. (b) Executando-as na ordem tarefa mais curta primeiro.



chegada são 0, 0, 3, 3 e 3. No início, apenas *A* ou *B* podem ser escolhidos, dado que as outras três tarefas não chegaram ainda. Usando a *tarefa mais curta primeiro*, executaremos as tarefas na ordem *A*, *B*, *C*, *D*, *E* para um tempo de espera médio de 4,6. No entanto, executá-las na ordem *B*, *C*, *D*, *E*, *A* tem um tempo de espera médio de 4,4.

Tempo restante mais curto em seguida

Uma versão preemptiva da *tarefa mais curta primeiro* é o **tempo restante mais curto em seguida (shortest remaining time next)**. Com esse algoritmo, o escalonador escolhe o processo cujo tempo de execução restante é o mais curto. De novo, o tempo de execução precisa ser conhecido antecipadamente. Quando uma nova tarefa chega, seu tempo total é comparado com o tempo restante do processo atual. Se a nova tarefa precisa de menos tempo para terminar do que o processo atual, este é suspenso e a nova tarefa iniciada. Esse esquema permite que tarefas curtas novas tenham um bom desempenho.

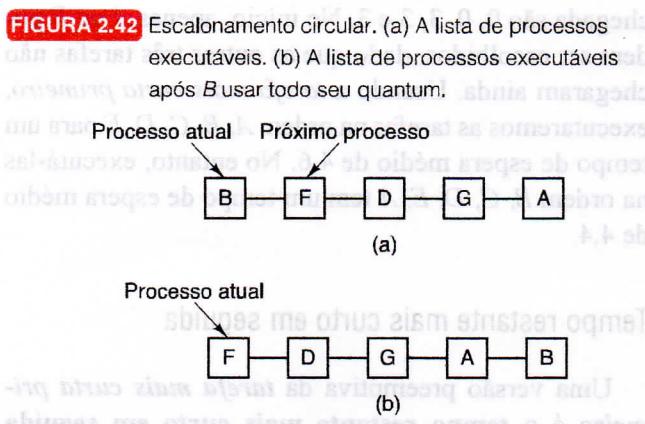
2.4.3 Escalonamento em sistemas interativos

Examinaremos agora alguns algoritmos que podem ser usados em sistemas interativos. Eles são comuns em computadores pessoais, servidores e outros tipos de sistemas também.

Escalonamento por chaveamento circular

Um dos algoritmos mais antigos, simples, justos e amplamente usados é o **circular (round-robin)**. A cada processo é designado um intervalo, chamado de seu **quantum**, durante o qual ele é deixado executar. Se o processo ainda está executando ao fim do quantum, a CPU sofrerá uma preempção e receberá outro processo. Se o processo foi bloqueado ou terminado antes de o quantum ter decorrido, o chaveamento de CPU será feito quando o processo bloquear, é claro. O escalonamento circular é fácil de implementar. Tudo o que o escalonador precisa fazer é manter uma lista de processos executáveis, como mostrado na Figura 2.42(a). Quando o processo usa todo o seu quantum, ele é colocado no fim da lista, como mostrado na Figura 2.42(b).

A única questão realmente interessante em relação ao escalonamento circular é o comprimento do quantum. Chavear de um processo para o outro exige certo



montante de tempo para fazer toda a administração — salvando e carregando registradores e mapas de memória, atualizando várias tabelas e listas, carregando e descarregando memória cache, e assim por diante. Suponha que esse **chaveamento de processo** ou **chaveamento de contexto**, como é chamado às vezes, leva 1 ms, incluindo o chaveamento dos mapas de memória, carregar e descarregar o cache etc. Também suponha que o quantum é estabelecido em 4 ms. Com esses parâmetros, após realizar 4 ms de trabalho útil, a CPU terá de gastar (isto é, desperdiçar) 1 ms no chaveamento de processo. Desse modo, 20% do tempo da CPU será jogado fora em overhead administrativo. Claramente, isso é demais.

Para melhorar a eficiência da CPU, poderíamos configurar o quantum para, digamos, 100 ms. Agora o tempo desperdiçado é de apenas 1%. Mas considere o que acontece em um sistema de servidores se 50 solicitações entram em um intervalo muito curto e com exigências de CPU com grande variação. Cinquenta processos serão colocados na lista de processos executáveis. Se a CPU estiver ociosa, o primeiro começará imediatamente, o segundo não poderá começar até 100 ms mais tarde e assim por diante. O último azarado talvez tenha de esperar 5 s antes de ter uma chance, presumindo que todos os outros usem todo o seu quantum. A maioria dos usuários achará demorada uma resposta de 5 s para um comando curto. Essa situação seria especialmente ruim se algumas das solicitações próximas do fim da fila exigissem apenas alguns milissegundos de tempo da CPU. Com um quantum curto, eles teriam recebido um serviço melhor.

Outro fator é que se o quantum for configurado por um tempo mais longo que o surto de CPU médio, a preempção não acontecerá com muita frequência. Em vez disso, a maioria dos processos desempenhará uma operação de bloqueio antes de o quantum acabar, provocando um chaveamento de processo. Eliminar a preempção

melhora o desempenho, porque os chaveamentos de processo então acontecem apenas quando são logicamente necessários, isto é, quando um processo é bloqueado e não pode continuar.

A seguinte conclusão pode ser formulada: estabelecer o quantum curto demais provoca muitos chaveamentos de processos e reduz a eficiência da CPU, mas estabelecer o longo demais pode provocar uma resposta ruim a solicitações interativas curtas. Um quantum em torno de 20-50 ms é muitas vezes bastante razoável.

Escalonamento por prioridades

O escalonamento circular pressupõe implicitamente que todos os processos são de igual importância. Muitas vezes, as pessoas que são proprietárias e operam computadores multiusuário têm ideias bem diferentes sobre o assunto. Em uma universidade, por exemplo, uma ordem hierárquica começaria pelo reitor, os chefes de departamento em seguida, então os professores, secretários, zeladores e, por fim, os estudantes. A necessidade de levar em consideração fatores externos leva ao **escalonamento por prioridades**. A ideia básica é direta: a cada processo é designada uma prioridade, e o processo executável com a prioridade mais alta é autorizado a executar.

Mesmo em um PC com um único proprietário, pode haver múltiplos processos, alguns dos quais são mais importantes do que os outros. Por exemplo, a um processo daemon enviando mensagens de correio eletrônico no segundo plano deve ser atribuída uma prioridade mais baixa do que a um processo exibindo um filme de vídeo na tela em tempo real.

Para evitar que processos de prioridade mais alta executem indefinidamente, o escalonador talvez diminua a prioridade do processo que está sendo executado em cada tique do relógio (isto é, em cada interrupção do relógio). Se essa ação faz que a prioridade caia abaixo daquela do próximo processo com a prioridade mais alta, ocorre um chaveamento de processo. Como alternativa, pode ser designado a cada processo um quantum de tempo máximo no qual ele é autorizado a executar. Quando esse quantum for esgotado, o processo seguinte na escala de prioridade recebe uma chance de ser executado.

Prioridades podem ser designadas a processos estaticamente ou dinamicamente. Em um computador militar, processos iniciados por generais podem começar com uma prioridade 100, processos iniciados por coronéis a 90, maiores a 80, capitães a 70, tenentes a 60 e assim por diante. Como alternativa, em uma central de computação comercial, tarefas de alta prioridade podem custar US\$

ROUND-ROBIN

Informações de entrada (definidas previamente):

- Quantum (constante) = intervalo de tempo dado pelo processador para execução de cada processo. Ex.: 20ms
- Tempo de uso de CPU de cada processo (variável). Ex.: 5~50ms.
- Intervalo de chegada de processos (variável).

$$\Delta T_F = \frac{\sum_{i=1}^m t_f}{m}$$

Informações de saída:

- Tempo médio do processo na fila. **Soma do tempo de fila de todos os processos dividido pelo número de processos (Δt_f)**.
- Tempo médio de uso de CPU dos processos. **Soma do tempo de uso de CPU de todos os processos dividido pelo número de processos (Δt_{CPU})**.
- Tempo médio de processo no sistema ($\Delta T_P = \Delta t_f + \Delta t_{CPU}$).

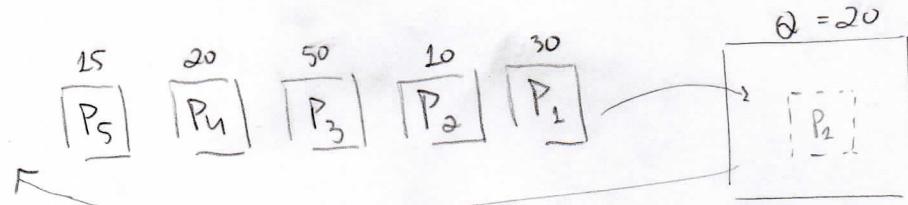
$$\Delta T_{CPU} = \frac{\sum_{i=1}^m t_{CPU}}{m}$$

ALGORITMO:

1. Se o processador estiver ocioso, o primeiro processo da fila entra em execução.
2. Se ao final do *quantum* dado ao processo ele não for finalizado, então o processo volta ao final da fila com o novo tempo necessário para que seja finalizado ($T_P - Q$).
3. Se um processo é finalizado durante o *quantum* o processador é liberado para o próximo processo (primeiro da fila)

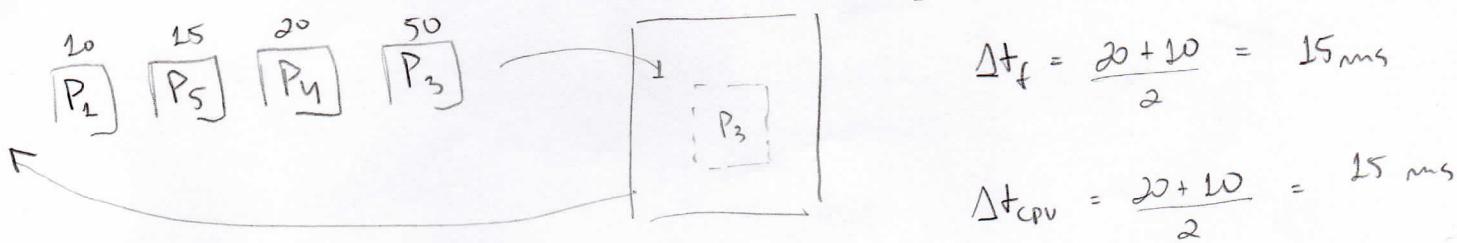
EXEMPLO GRÁFICO

tempo em milisegundos



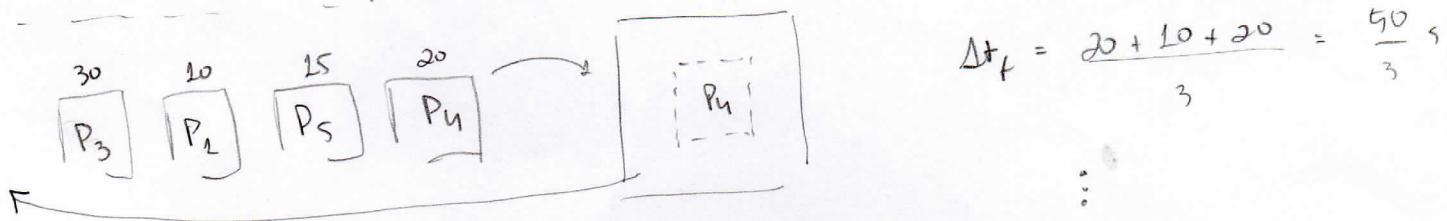
$$\Delta t_f = \frac{20}{1} = 20 \text{ ms}$$

$$\Delta t_{cpu} = \frac{20}{1} = 20 \text{ ms}$$



$$\Delta t_f = \frac{20 + 10}{2} = 15 \text{ ms}$$

$$\Delta t_{cpu} = \frac{20 + 10}{2} = 15 \text{ ms}$$



$$\Delta t_f = \frac{20 + 10 + 20}{3} = \frac{50}{3} \text{ s}$$

* **OBS:** Nesse período de tempo novos processos podem entrar na fila.

* Adicionar tempo de checamento?

Isso seria o tempo médio de fila sempre será igual ao tempo médio de processamento.