# Relatório sobre Gerência do Processador

**André Renato, Diego Prestes de Sousa, Vinícius Reinert**

**Resumo.** O relatório a seguir descreve a gerência do processador ao lidar com processos e aborda os métodos que o processador utiliza para tratar processos; indo desde as funções básicas como escalonamento, falando sobre os critérios de escalonamento e definindo alguns conceitos já abordados nas aulas de Princípios de Arquitetura de Computadores como *throughput*, tempo de processador e tempo de espera; traçando um paralelo com o objeto de estudo atual: Sistemas Operacionais. Dentre os tópicos avançados iremos discutir sobre as funções mais básicas, indo até o mais básico sobre os métodos mais avançados de gerência do processador como as técnicas de escalonamento de processos e políticas de escalonamento, tanto para sistemas de tempo compartilhado e sistemas de tempo real. Vale dizer também que grande parte desse material foi retirado do livro "Arquitetura de Sistemas Operacionais, 5ª edição" e o livro *"Operating system concepts 8th ed. Hoboken"*, e complementarei com fontes externas conforme avançamos no tema e todas as referências estarão no final do relatório.

## Introdução

Com o surgimento dos sistemas multiprogramáveis, nos quais múltiplos processos poderiam permanecer na memória principal compartilhando o uso da UCP, a gerência do processador tornou-se uma das atividades mais importantes em um sistema operacional. A partir do momento em que diversos processos podem estar no estado de pronto, critérios devem ser estabelecidos para determinar qual processo será escolhido para fazer uso do processador. Os critérios utilizados para esta seleção compõem a chamada política de escalonamento, que é a base da gerencia do processador e da multiprogramação em um sistema operacional.

Como dito no resumo, aqui abordaremos as funções básicas de escalonamento, políticas para estes escalonamentos e alguns algoritmos, descrevendo os mecanismos existentes e implementados na gerência do processador.

## A Estrutura do Relatório

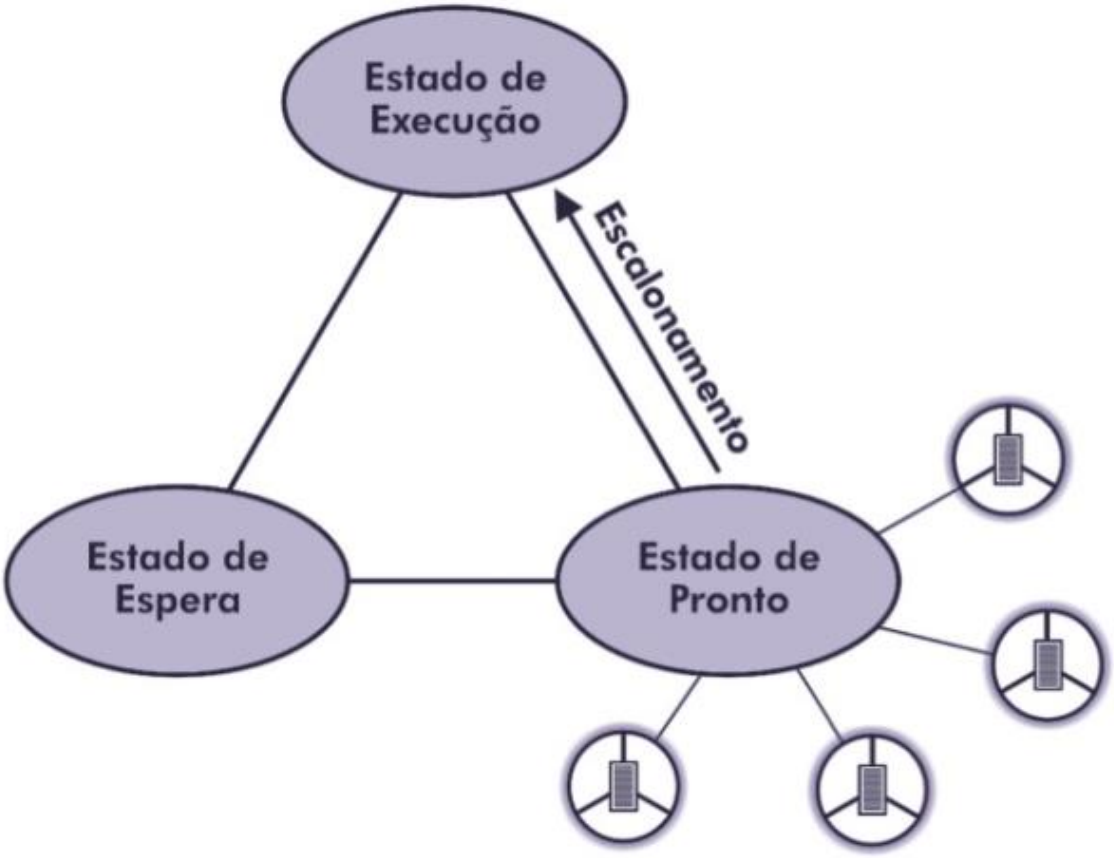
De certa forma, todas as aulas, trabalhos de TCC, seminários e artigos que encontrei sobre o tópico "Gerência do Processador" citam o livro de Arquitetura de Sistemas Operacionais, inclusive usei outra versão do livro como fonte de inspiração também para traçar um paralelo entre um e outro – a princípio esse livro de Arquitetura de Sistemas Operacionais é a bíblia para quem curte o assunto, quase todas as referências citam esse livro e não encontrei outros livros que

falassem especificamente da gerência do processador sobre processos. Portanto, resumir esse relatório com cortes do texto do livro e incluir outras fontes seria injusto, pois o livro já sintetiza bem e explica de forma muito objetiva; então, faremos assim: seguiremos a base que o livro nos fornece, e com base nas referências que coletei conforme ia estruturando o relatório (e a apresentação que faremos dia 30 de novembro de 2020) vou tecendo comentários e fazendo complementações que possam simplificar o entendimento. Todos os tópicos a partir de agora acompanham uma conclusão, além dos comentários que seguem parafraseando o autor do livro com minhas falas, baseadas em tudo que vimos na aula e tudo que coletei até aqui. Receio que ficará mais extenso do que o planejado; mas, como diz minha mãe: "melhor sobrar do que faltar".

O relatório foi construído de forma que não será necessária leitura prévia do livro para o entendimento completo e para adentrar no contexto que proponho no relatório. Então, apesar de extenso, o relatório contém muitas informações coletadas do livro que serviu de "*Roadmap*" neste desenvolvimento e também coletadas de muitos outros lugares pela internet. Mas também apoiado fortemente no livro *"Operating system concepts 8th ed. Hoboken"*.

## Funções Básicas

A política de escalonamento de um sistema operacional tem diversas funções básicas, como a de manter o processador ocupado a maior parte do tempo, balancear o uso da UCP entre processos, privilegiar a execução de aplicações críticas, maximizar o throughput do sistema e oferecer tempos de resposta razoáveis para usuários interativos. Cada sistema operacional possui sua política de escalonamento adequada ao seu propósito e às suas características. Sistemas de tempo compartilhado, por exemplo, têm requisitos de escalonamento distintos dos sistemas de tempo real.




A imagem acima representa bem o conceito de escalonamento. Um processo em execução, se não acabado, vai (ou, muitas vezes, volta) para o estado de espera, e há alguns processos (aquelas bolinhas ligadas ao estado de pronto são processos) que estão aguardando serem "escalados" para execução. Como um corredor olímpico pronto para correr assim que escutar o som da largada.

A rotina do sistema operacional que tem como principal função implementar os critérios da política de escalonamento é denominada escalonador (*scheduler*). Em um sistema multiprogramável, o escalonador é fundamental, pois todo o compartilhamento do processador é dependente desta rotina.

Outra rotina importante na gerência do processador é conhecida como *dispatcher*, responsável pela troca de contexto dos processos após o escalonador determinar qual processo deve fazer uso do processador. O período de tempo gasto na substituição de um processo em execução por outro é denominado latência do *dispatcher*.

Em ambientes que implementam apenas processos, o escalonamento é realizado com base nos processos prontos para execução. Em sistemas que implementam threads, o escalonamento é realizado considerando os threads no estado de pronto, independentemente do processo. Neste caso, podemos considerar o processo como a unidade de alocação de recursos, enquanto o thread

é a unidade de escalonamento. Ao longo de todo o relatório o termo “processo” será usado de maneira genérica como o elemento selecionado pelo escalonador. Afinal, não estamos abordando processos em si neste relatório, mas sim a maneira com que o sistema operacional gerencia este processo.

## Conclusão sobre Funções Básicas

Aqui, como o próprio título do tópico diz, vendo as noções básicas sobre escalonamento. Aqui não há uma estratégia ou critério de escalonamento em si, mas sim uma noção de como as coisas funcionam de forma superficial, como se dá o conceito de escalonamento, e não suas implementações propriamente ditas. Como descrito abaixo da imagem que acompanha o texto acima.

## Critérios de Escalonamento

As características de cada sistema operacional determinam quais são os principais aspectos para a implementação de uma política de escalonamento adequada. Por exemplo, sistemas de tempo compartilhado exigem que o escalonamento trate todos os processos de forma igual, evitando, assim, a ocorrência de *starvation*, ou seja, que um processo fique indefinidamente esperando pela utilização do processador. Já em sistemas de tempo real, o escalonamento deve priorizar a execução de processos críticos em detrimento da execução de outros processos.

**NOTA**: *“Starvation”* é quando um processo nunca é executado, pois sempre aparece processos com maior prioridade o impedindo de executar. Então este evento ocorre. *Starvation* também é conhecido como “inanição”.

Segue aqui os principais critérios levados em consideração em uma política de escalonamento:

## Utilização do processador

Na maioria dos sistemas é desejável que o processador permaneça a maior parte do seu tempo ocupado. Uma utilização na faixa de 30% indica um sistema com uma carga de processamento baixa, enquanto na faixa de 90% indica um sistema bastante carregado, próximo da sua capacidade máxima.

## Throughput

Throughput representa o número de processos executados em um determinado intervalo de tempo. Quanto maior o throughput, maior o número de tarefas executadas em função do tempo. A maximização do throughput é desejada na maioria dos sistemas.

## Tempo de Processador / Tempo de UCP

Tempo de processador ou tempo de UCP é o tempo que um processo leva no estado de execução durante seu processamento. As políticas de escalonamento não influenciam o tempo de processador de um processo, sendo este tempo função apenas do código da aplicação e da entrada de dados.

## Tempo de Espera

Tempo de espera é o tempo total que um processo permanece na fila de pronto durante seu processamento, aguardando para ser executado. A redução do tempo de espera dos processos é desejada pela maioria das políticas de escalonamento.

## Tempo de Turnaround

Tempo de turnaround é o tempo que um processo leva desde a sua criação até seu término, levando em consideração todo o tempo gasto na espera para alocação de memória, espera na fila de pronto (tempo de espera), processamento na UCP (tempo de processador) e na fila de espera, como nas operações de E/S. As políticas de escalonamento buscam minimizar o tempo de turnaround.

## Tempo de Resposta

Tempo de resposta é o tempo decorrido entre uma requisição ao sistema ou à aplicação e o instante em que a resposta é exibida. Em sistemas interativos, podemos entender como o tempo decorrido entre a última tecla digitada pelo usuário e o início da exibição do resultado no monitor. Em geral, o tempo de resposta não é limitado pela capacidade de processamento do sistema computacional, mas pela velocidade dos dispositivos de E/S. Em sistemas interativos, como aplicações on-line ou acesso à Web, os tempos de resposta devem ser da ordem de poucos segundos.

## Conclusão sobre Critérios de Escalonamento

As políticas de escalonamento podem ser classificadas segundo a possibilidade de o Sistema Operacional interromper um processo em execução e substitui-lo por um outro, atividade está conhecida como preempção. Sistemas operacionais que implementam escalonamento com preempção são mais complexos, contudo, possibilitam políticas de escalonamento mais flexíveis. Agora que temos os conceitos básicos, e, caso necessário durante a leitura, é sempre possível voltar aqui e relembrar desses conceitos.

## Escalonamento Não Preemptivo e Escalonamento Preemptivo

Em sistemas operacionais, preemptividade ou preempção é a capacidade de tirar de execução um processo em detrimento de outro. Esta é uma característica que não é importante apenas nos sistemas operacionais em tempo real. Este tipo de intervenção por parte dos escalonadores dos sistemas operacionais pode ocorrer – Embora não seja limitada apenas a isso – a otimizar a entrada/saída de dados em tempo real, como é o caso da gravação de áudio. Um exemplo de uma tarefa não-preemptiva é o processamento de interrupções.

As políticas de escalonamento podem ser classificadas segundo a possibilidade de o sistema operacional interromper um processo em execução e substitui-lo por um outro, esta atividade é conhecida como preempção. Sistemas operacionais que implementam escalonamento com preempção são mais complexos, contudo, possibilitam políticas de escalonamento mais flexíveis.

O escalonamento não preemptivo foi o primeiro tipo de escalonamento implementado nos sistemas multiprogramáveis, onde predominava tipicamente o processamento batch. Nesse tipo de escalonamento, quando um processo está em execução nenhum evento externo pode ocasionar a perda do uso do processador. O processo somente sai do estado de execução caso termine seu processamento ou execute instruções do próprio código que ocasionem uma mudança para o estado de espera.

No escalonamento preemptivo, o sistema operacional pode interromper um processo em execução e passa-lo para o estado de pronto, com o objetivo de alocar outro processo na UCP. Com o uso da preempção, é possível ao sistema priorizar a execução de processos, como no caso de aplicações de tempo real, em que o fator tempo é critico. Outro benefício é a possibilidade de implementar políticas de escalonamento que compartilhem o processador de uma maneira mais uniforme, distribuindo de forma balanceada o uso da UCP entre os processos.

## Conclusão sobre Escalonamento Preemptivo e Não-Preemptivo

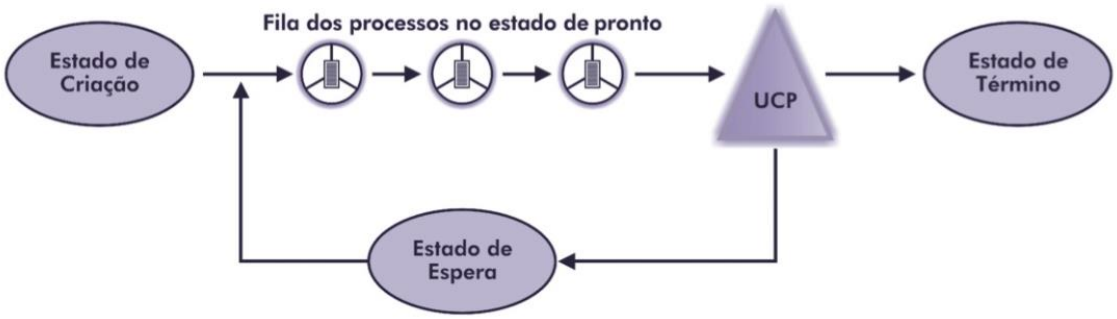
Atualmente, a maioria dos sistemas operacionais implementa políticas de escalonamento preemptivas que, apesar de tornarem os sistemas mais complexos, possibilitam a implementação dos diversos critérios de escalonamento apresentados.

Sistemas de tempo real costumam ser implementados em situações críticas, um bom exemplo de ambiente onde é implementado esse sistema é o sistema de saúde, mas cenários como usinas nucleares e indústrias petroquímicas são bons exemplos desse sistema. Esse tipo de sistema é, até onde vejo, o melhor exemplo do uso da preempção, pois em situações críticas como essas que se vê necessário o melhor uso do tempo para o processador. Onde falhas podem ser críticas.

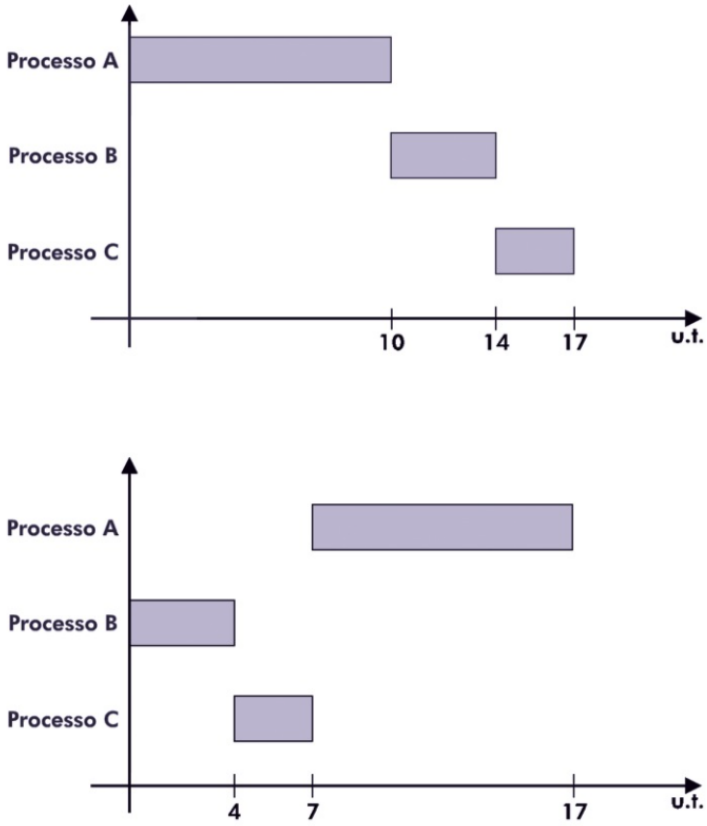
Preempção é utilizado em sistemas operacionais de computador doméstico, sim; mas se um arquivo de texto demorar 2 segundos a mais para abrir não vai acabar com a vida de alguém, ou gerar um acidente químico/nuclear que pode deixar uma região do globo inabitável (isso, claro, em exemplos catastróficos). É observando situações críticas que entendemos as vantagens e a otimização que a preempção oferece.

## Escalonamento *First-In-First-Out* (FIFO)

No escalonamento *first-in-first-out* (FIFO *scheduling*), também conhecido como *first-come-first- served* (FCFS *scheduling*), o processo que chegar primeiro ao estado de pronto é o selecionado para execução. Este algoritmo é bastante simples, sendo necessária apenas uma fila, onde os processos que passam para o estado de pronto entram no seu final e são escalonados quando chegam ao seu início. Quando um processo vai para o estado de espera, o primeiro processo da fila de pronto é escalonado. Todos os processos quando saem do estado de espera entram no final da fila de pronto. A figura abaixo torna a visualização de todo o procedimento mais palpável:




Na figura abaixo é possível comparar o uso do escalonamento FIFO em duas situações distintas, em que os processos A, B e C são criados no instante de tempo 0, com os tempos de processador 10, 4 e 3, respectivamente. A diferença entre os exemplos da figura abaixo é o posicionamento dos processos na fila de pronto. O tempo médio de espera dos três processos no exemplo (a) é igual a (0 + 10 + 14) /3 = 8 u.t., enquanto no exemplo (b) é de aproximadamente (7 + 0 + 4) /3 = 3,7 u.t. A diferença entre as médias é bastante significativa, justificada pela diferença dos tempos de processador entre os processos.




| Processo | Tempo de processador (u.t.) |
|---|---|
| A | 10 |
| B | 4 |
| C | 3 |

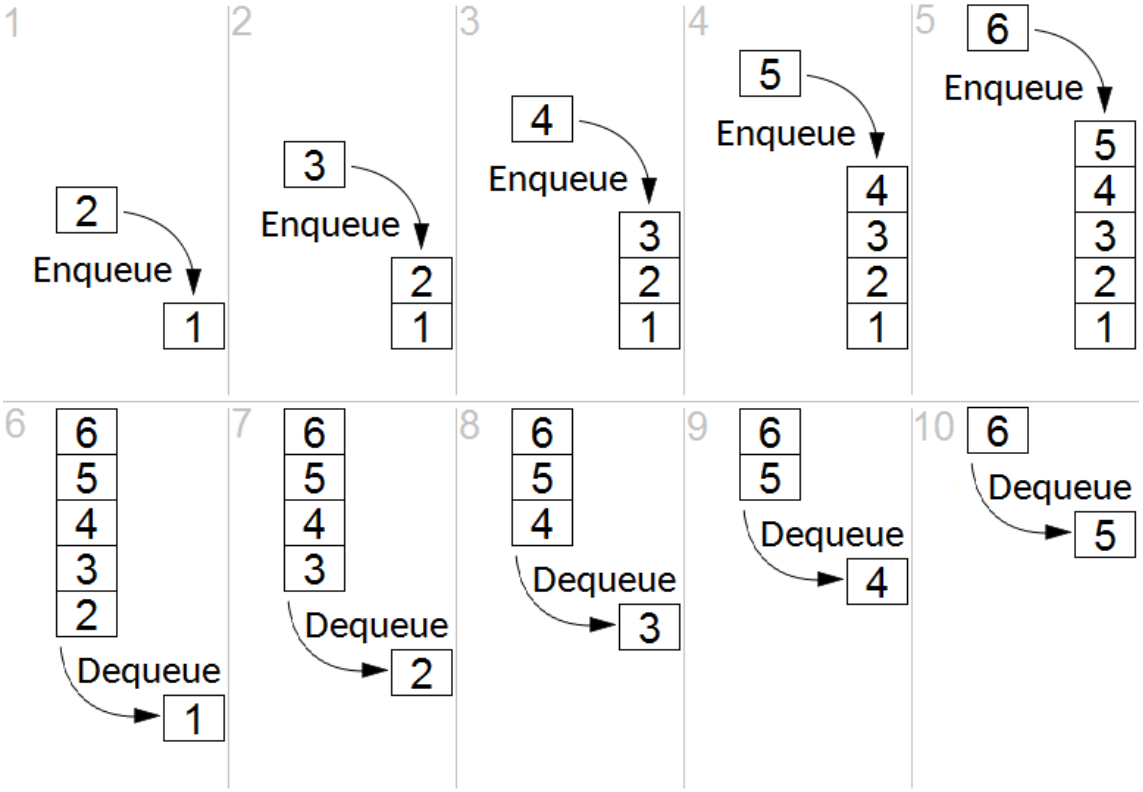
Apesar de simples, o escalonamento FIFO apresenta algumas deficiências. O principal problema é a impossibilidade de prever-se quando um processo terá sua execução iniciada, já que isso varia em função do tempo de execução dos demais processos posicionados à sua frente na fila de pronto. Como vimos no exemplo da figura anterior, o algoritmo de escalonamento não se preocupa em melhorar o tempo médio de espera dos processos, utilizando apenas a ordem de chegada dos processos à fila de pronto. Este problema pode ser mais bem percebido nos tempos de *turnaround* dos processos que demandam menor tempo de UCP.

Outro problema neste tipo de escalonamento é que processos *CPU-bound* levam vantagem no uso do processador sobre processos *I/O-bound*. No caso de existirem processos *I/O-bound* mais importantes do que os *CPU-bound*, não é possível tratar este tipo de diferença.

O escalonamento FIFO é do tipo não preemptivo e foi inicialmente implementado em sistemas monoprogramáveis com processamento batch, sendo ineficiente, se aplicado da forma original em sistemas interativos de tempo compartilhado. Atualmente, sistemas de tempo compartilhado utilizam o escalonamento FIFO com variações, permitindo, assim, parcialmente sua implementação.

## Conclusão sobre FIFO, *First-in-First-out*

As aplicações do FIFO normalmente se dão em circuitos eletrônicos de *Buffer* e controle de fluxo, desde o *Hardware* até o *Software*. Na forma de um *hardware* o *FIFO* consiste basicamente de um conjunto de ler e escrever ponteiros, armazenamento e lógica de controle. Armazenamento pode ser *SRAM*, *flip-flop*, fechos ou qualquer outra forma adequada de armazenamento. Para o *FIFO*, de tamanho não trivial, uma *SRAM* de porta dupla geralmente é utilizada quando uma porta é usada para a escrita e a outra para leitura.




Infelizmente, o texto acima é mais complexo do que eu gostaria, mas foi baseado em um artigo que encontrei sobre algoritmo de fila simples. Mas, o importante é que o FIFO é um algoritmo de fila simples, o primeiro que entrar é o primeiro que irá ser executado (a imagem acima é autoexplicativa). Isso, obviamente, é falho em vários sentidos e muito pouco otimizado. Aprendemos algoritmos parecidos com esse nas aulas de estrutura de dados. Consigo pensar em várias formas em que um *software* monopoliza o processador e gera vários subprocessos e processos novos que irão concorrer com os processos que o sistema já tem na fila para executar, tornando o processador uma espécie de "*zumbi*".

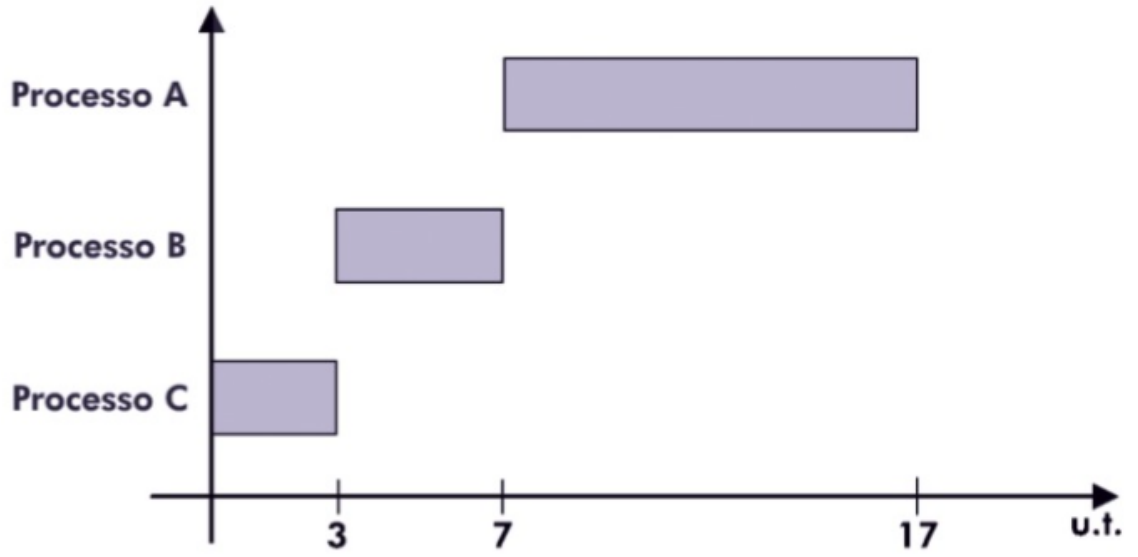
```
@echo off

cd backend

call npm i

cd ..

cd frontend

call npm i

cd ..

call docker run --name plather-db -p 27017:27017 mongo:latest
```

Este é um exemplo de algoritmo de batch feito por mim mesmo a algumas semanas atrás. Note que quando eu digo "*call*" estou chamando um processo externo ao meu console do Windows para executar, e todos os processos executam em ordem. Ainda seria possível aprimorar esse pequeno pedaço de *software* implementando o FIFO assíncrono, que utiliza diferentes *clocks* para leitura e escrita, isso iria otimizar meu minúsculo algoritmo de batch.

## Escalonamento *Shortest-Job-First* (SJF)

No escalonamento *shortest-job-first* (SJF *scheduling*), também conhecido como *shortest-process-next* (SPN *scheduling*), o algoritmo de escalonamento seleciona o processo que tiver o menor tempo de processador ainda por executar. Dessa forma, o processo em estado de pronto que necessitar de menos tempo de UCP para terminar seu processamento é selecionado para execução.

Na figura abaixo é mostrado como ficaria o escalonamento utilizando o algoritmo SJF a partir dos mesmos processos utilizados no exemplo da figura anterior, onde implementamos o FIFO. O tempo médio de espera dos três processos, igual a (7 + 3 + 0) /3 = 3,3 u.t., é inferior ao apresentado no escalonamento FIFO.



Esta implementação foi utilizada nos primeiros sistemas operacionais com processamento exclusivamente batch. Para cada novo processo admitido no sistema, um tempo de processador era associado ao seu contexto de software. Como não é possível precisar o tempo de processador previamente para cada processo, uma estimativa era realizada com base em analises estatísticas de execuções passadas dos programas. O tempo estimado era informado ao sistema operacional, que o utilizava como no algoritmo de escalonamento. Caso o tempo informado fosse muito inferior ao tempo real, o sistema operacional poderia interromper a execução do processo. O principal problema nesta implementação é a impossibilidade de estimar o tempo de processador para processos interativos, já que a entrada de dados é uma ação imprevisível.

Uma maneira de implementar o escalonamento SJF em sistemas interativos foi considerar o comportamento do processo neste ambiente. A característica de um processo interativo é ser escalonado, executar algumas poucas instruções e fazer uma operação de E/S. Após o término da operação, o processo é posteriormente escalonado, executa mais algumas instruções e faz outra operação de E/S. Este comportamento se repete até o término do processamento. Neste caso, o escalonamento é realizado com base no tempo que um processo irá utilizar a UCP na próxima vez em que for escalonado, e não mais no tempo total de processador que utilizará até o término do seu processamento, como era o caso do FIFO que vimos anteriormente.

Um problema existente nesta implementação é não ser possível ao sistema operacional saber quanto tempo um processo permanecerá utilizando a UCP na próxima vez em que for escalonado, contudo é possível prever o tempo com base no seu comportamento passado. O tempo pode ser estimado com base na média exponencial dos tempos passados, onde $t_n$ é o tempo do processador utilizado no último escalonamento e $\delta_{n+1}$ o valor estimado que será utilizado no escalonamento seguinte:

$$\delta_{n+1} = \alpha\, t_n + (1 - \alpha)\, \delta_n$$

O valor de $t_n$ representa a informação mais recente, enquanto $\delta_n$ possui o valor histórico. Com a determinação do valor da constante $\alpha$ ($0 < \alpha < 1$) é possível ponderar uma maior importância na estimativa para os valores recentes ou passados. Um valor de $\alpha = 1/2$ considera os valores recentes e passados com o mesmo peso.

Uma implementação do escalonamento SJF com preempção é conhecida como escalonamento *shortest-remaining-time* (SRT *scheduling*). Nesta política, toda vez que um processo no estado de pronto tem um tempo de processador estimado menor do que o processo em execução o sistema operacional realiza uma preempção, substituindo-o pelo novo processo.

De modo semelhante ao SJF, o sistema operacional deve ser o responsável por estimar os tempos de processador dos processos, e o risco de *starvation* continua presente.

## Conclusão sobre SJF, Shortest-Job-First

Sua vantagem sobre o escalonamento FIFO está na redução do tempo médio de *turnaround* dos processos, porém no SJF é possível haver starvation para processos com tempo de processador muito longo ou do tipo *CPU-bound*.

Segundo o artigo que encontrei sobre SJF esta forma de escalonamento é vantajosa por sua simplicidade e também porque minimiza o tempo médio que cada processo leva desde quando ele é criado até o fim de sua execução, incluindo aqui o tempo de espera entre o momento em que ele é criado e o momento em que é selecionado para executar. No entanto, essa estratégia pode levar a inanição de processos com longos tempos de execução caso processos curtos sejam continuamente adicionados ao escalonador. Vimos o conceito de inanição (ou *starvation*) anteriormente.

Um algoritmo que implementa o SJF é o "*High Response Ratio Next*". Este algoritmo leva em conta o "envelhecimento" dos processos:

$$rr = \frac{tempo\_de\_espera + tempo\_de\_servico}{tempo\_de\_servico}$$

Esta forma de escalonamento é muito semelhante ao SJF, porém leva em consideração o envelhecimento do processo, como já disse anteriormente. Esta abordagem evita que *threads* longas não fiquem esperando por tempo indeterminado. Por exemplo: vários processos minúsculos entram na frente de um processo gigantesco, e aquele processo enorme fica ali, passando fome, sem recursos para execução. No caso do SJF, onde essa situação não é tratada, o processo grande está passível de inanição. Enquanto o HRRN (*High Response Ratio Next)* soluciona esse problema levando em consideração o tempo que aquele processo está esperando para execução.

## Escalonamento Cooperativo

O escalonamento cooperativo é uma implementação que busca aumentar o grau de multiprogramação em políticas de escalonamentos que não possuam mecanismos de preempção, como o FIFO e o SJF não preemptivo. Neste caso, um processo em execução pode voluntariamente liberar o processador, retornando à fila de pronto e possibilitando que um novo processo seja escalonado, permitindo assim uma melhor distribuição no uso do processador.

A principal característica do escalonamento cooperativo está no fato de a liberação do processador ser uma tarefa realizada exclusivamente pelo processo em execução, que de uma

maneira cooperativa libera a UCP para um outro processo. Neste mecanismo, o processo em execução verifica periodicamente uma fila de mensagens para determinar se existem outros processos na fila de pronto.

Como a interrupção do processo em execução não é responsabilidade do sistema operacional, algumas situações indesejadas podem ocorrer. No caso de um processo não verificar a fila de mensagens, os demais não terão chance de ser executados até a liberação da UCP pelo processo em execução. Isto pode ocasionar sérios problemas para a multiprogramação cooperativa, na medida em que um programa pode permanecer por um longo período de tempo alocando o processador.

## Conclusão sobre Escalonamento Cooperativo

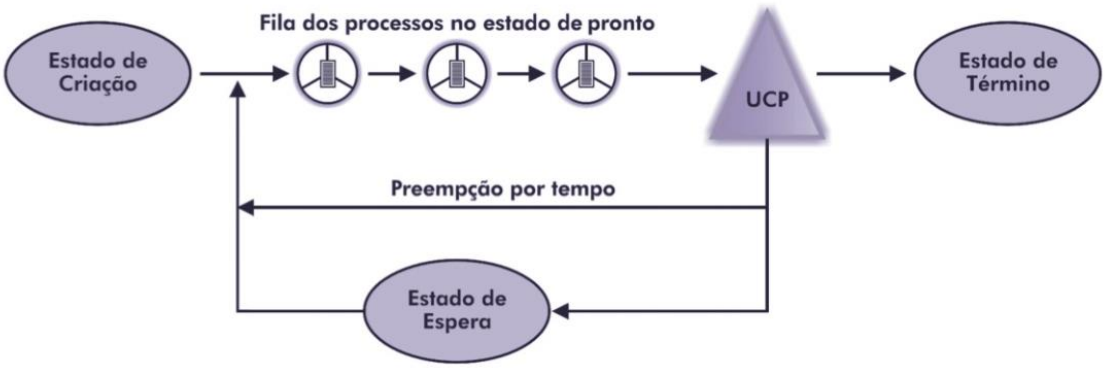
Essa política de escalonamento parece estar um pouco obscura, visto que a grande maioria dos exemplos que encontrei de escalonamento cooperativo faziam citação direta deste livro que estou usando como base. Nada mais que "um exemplo deste tipo de escalonamento pode ser encontrado nos primeiros sistemas operacionais da família Microsoft Windows, sendo conhecido como multitarefa cooperativa". Mas, apesar de não encontrar nenhum exemplo legitimo descrito em artigos pela internet, é possível dizer com base no pouco que encontrei que o escalonamento cooperativo foi uma porta de entrada para preempção, pois não era preempção propriamente dita, porém existia o mecanismo de passar um processo em execução para a fila de pronto e puxar outro para a execução, o que, como dito no texto acima, otimizava a distribuição do processador muito melhor, mas ainda não é uma política de escalonamento complexa que permite variações e condições especificas para tratar as diversas situações que podem vir a ocorrer durante o gerenciamento dos processos.

## Escalonamento Circular

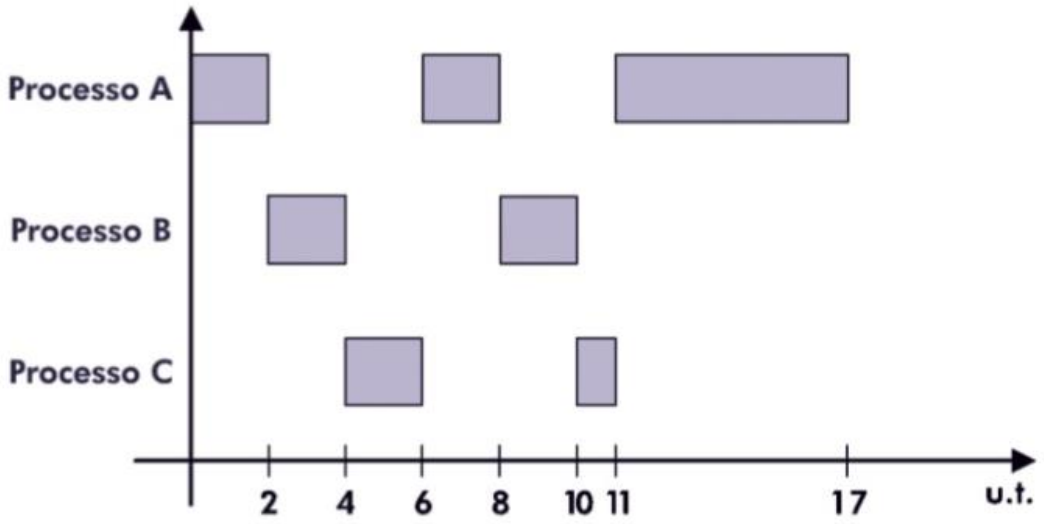
O escalonamento circular (*round Robin scheduling*) é um escalonamento do tipo preemptivo, projetado especialmente para sistemas de tempo compartilhado. Esse algoritmo é bastante semelhante ao FIFO, porém quando um processo passa para o estado de execução existe um tempo-limite para o uso contínuo do processador denominado fatia de tempo (*time-slice*) ou *quantum*.

No escalonamento circular, toda vez que um processo é escalonado para execução uma nova fatia de tempo é concedida. Caso a fatia de tempo expire, o sistema operacional interrompe o processo em execução, salva seu contexto e direciona-o para o final da fila de pronto. Esse mecanismo é conhecido como preempção por tempo.

A figura abaixo ilustra o escalonamento circular, em que a fila de processos em estado de pronto é tratada como uma fila circular. O escalonamento é realizado alocando a UCP ao primeiro processo da fila de pronto. O processo permanecerá no estado de execução até que termine seu processamento, voluntariamente passe para o estado de espera ou que sua fatia de tempo expire, sofrendo, neste caso, uma preempção pelo sistema operacional. Após isso, um novo processo é escalonado com base na política de FIFO.




A próxima figura, que segue abaixo deste texto, exemplifica o escalonamento circular com três processos, em que a fatia de tempo é igual a 2 u.t. No exemplo não está sendo levado em consideração o tempo de latência do *dispatcher*, ou seja, o tempo de troca de contexto entre os processos.



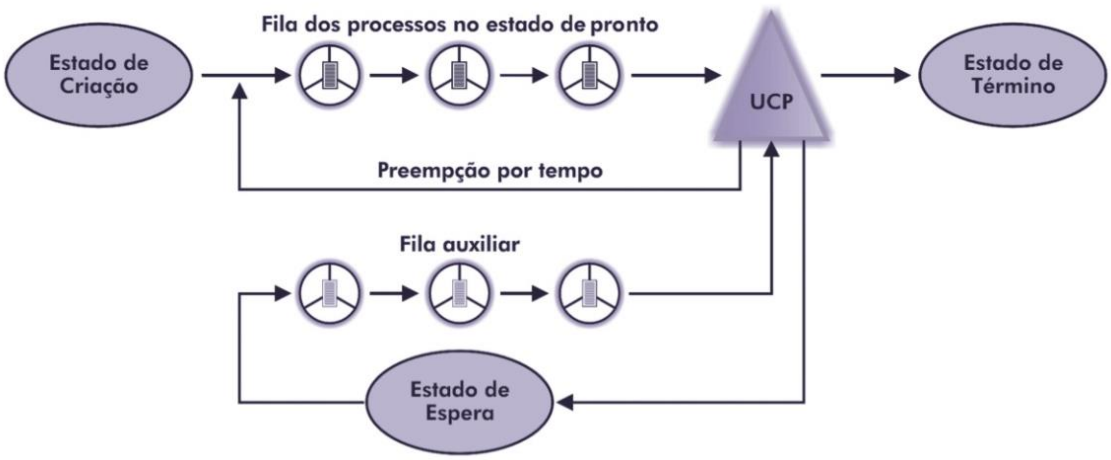

O valor da fatia de tempo depende da arquitetura de cada sistema operacional e, em geral, varia entre 10 e 100 milissegundos. Este valor afeta diretamente o desempenho da política de escalonamento circular. Caso a fatia de tempo tenha um valor muito alto, este escalonamento tenderá a ter o mesmo comportamento do escalonamento FIFO. Caso o valor do *time-slice* seja pequeno, a tendência é que haja um grande número de preempções, o que ocasionaria excessivas

mudanças de contexto, prejudicando o desempenho do sistema e afetando o tempo de turnaround dos processos.

A principal vantagem do escalonamento circular é não permitir que um processo monopolize a UCP, sendo o tempo máximo alocado continuamente igual à fatia de tempo definido no sistema. No caso de sistemas de tempo compartilhado, onde existem diversos processos interativos concorrendo pelo uso do processador, o escalonamento circular é adequado.

Um problema presente nesta política é que processos *CPU-bound* são beneficiados no uso do processador em relação aos processos *I/O-bound*. Devido às suas características, os processos CPU-bound tendem a utilizar por completo a fatia de tempo, enquanto os processos *I/O-bound* têm mais chances de passar para o estado de espera antes de sofrerem preempção por tempo. Estas características distintas ocasionam um balanceamento desigual no uso do processador entre os processos.

Um refinamento do escalonamento circular, que busca reduzir este problema, é conhecido como escalonamento circular virtual, ilustrado na figura que segue logo abaixo deste trecho do texto. Neste esquema, processos que saem do estado de espera vão para uma fila de pronto auxiliar. Os processos da fila auxiliar possuem preferência no escalonamento em relação à fila de pronto, e o escalonador só seleciona processos na fila de pronto quando a fila auxiliar estiver vazia. Quando um processo é escalonado a partir da fila auxiliar, sua fatia de tempo é calculada como sendo o valor da fatia de tempo do sistema menos o tempo de processador que o processo utilizou na última vez em que foi escalonado a partir da fila de pronto. Estudos comprovam que, apesar da maior complexidade na implementação, o balanceamento do uso do processador neste escalonamento é mais equilibrado.




Esse esquema em que o sistema operacional busca ajustar dinamicamente a ordem de escalonamento dos processos objetivando o balanceamento no uso do processador é denominado mecanismo adaptativo.

## Conclusão sobre Escalonamento Circular

Esta é a primeira política de escalonamento que vemos que é preemptiva. Encontrei vários documentos na internet que falavam sobre, mas, novamente, todos parafraseavam o autor do livro e não adicionavam nada muito relevante, nem mesmo um exemplo ou caso de uso. Mas baseado no artigo de Arpaci-Dusseau e Remzi H. que encontrei, posso exemplificar o escalonamento circular como:

*A fatia de tempo (time-slice ou quantum, como também é chamado essa fatia de tempo dedicada ao processo) é 100ms (ms = milissegundos) e o processo leva 250ms para sua conclusão. O escalonamento circular suspenderá a tarefa após os 100ms e guardará seu contexto para quando ela voltar a executar, e puxara o próximo processo da fila para a execução, concedendo a este novo processo os mesmos 100ms. Aquele processo do começo, a qual demos 100ms no começo do exemplo será executada (ou, se preferir, escalada para execução) 3 vezes, ficando algo como: 100ms + 100ms + 50ms. Totalizando o tempo necessário para a conclusão do processo.*

## Escalonamento por Prioridades

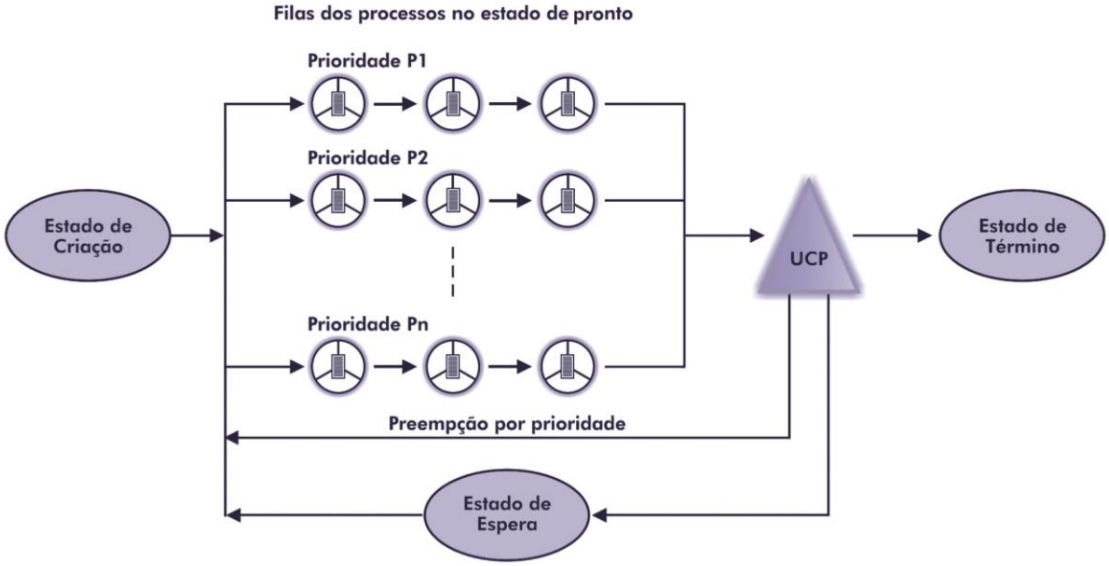
O escalonamento por prioridades é um escalonamento do tipo preemptivo realizado com base em um valor associado a cada processo denominado prioridade de execução. O processo com maior prioridade no estado de pronto é sempre o escolhido para execução, e processos com valores iguais são escalonados seguindo o critério de FIFO. Neste escalonamento, o conceito de fatia de tempo não existe, consequentemente um processo em execução não pode sofrer preempção por tempo.

No escalonamento por prioridades, a perda do uso do processador só́ ocorrerá no caso de uma mudança voluntária para o estado de espera ou quando um processo de prioridade maior passa para o estado de pronto. Neste caso, o sistema operacional deverá interromper o processo corrente, salvar seu contexto e colocá-lo no estado de pronto. Esse mecanismo é conhecido como preempção por prioridade. Após isso, o processo de maior prioridade é escalonado.

A preempção por prioridade é implementada através de uma interrupção de *clock*, gerada em determinados intervalos de tempo, para que a rotina de escalonamento reavalie as prioridades dos processos no estado de pronto. Caso haja processos na fila de pronto com maior prioridade do que o processo em execução, o sistema operacional realiza a preempção.

A figura abaixo ilustra o escalonamento por prioridades, no qual para cada prioridade existe uma fila de processos em estado de pronto que é tratada como uma fila circular. O escalonamento é realizado alocando o processador ao primeiro processo da fila de prioridade mais

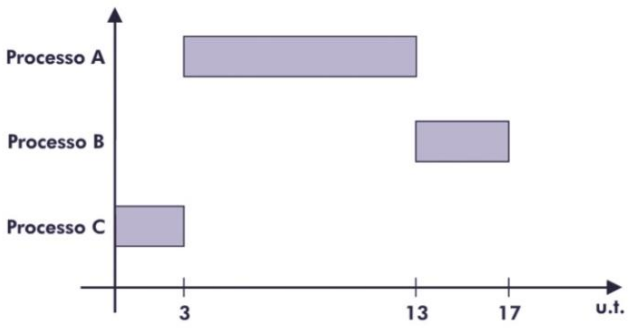
alta. O processo permanecerá no estado de execução até que termine seu processamento, voluntariamente passe para o estado de espera ou sofra uma preempção devido a um processo de maior prioridade.




A próxima figura, que segue abaixo deste trecho, exemplifica o escalonamento por prioridades com três processos de prioridades diferentes, onde 5 é o valor mais alto de prioridade.

O escalonamento por prioridades também pode ser implementado de uma maneira não preemptiva. Neste caso, processos que passem para o estado de pronto com prioridade maior do que a do processo em execução não ocasionam preempção, sendo apenas colocados no início da fila de pronto.

Cada sistema operacional implementa sua faixa de valores para as prioridades de execução. Alguns sistemas associam as maiores prioridades a valores altos, enquanto outros sistemas utilizam valores baixos. No caso do *OpenVMS*, a prioridade do processo pode variar de 0 a 31, sendo 31 a maior prioridade. No *IBM-AIX* a prioridade varia de 0 a 127, porém os valores mais baixos possuem maior prioridade de execução.




| Processo | Tempo de processador (u.t.) | Prioridade |
|---|---|---|
| A | 10 | 2 |
| B | 4 | 1 |
| C | 3 | 3 |

A prioridade de execução é uma característica do contexto de software de um processo, e pode ser classificada como estática ou dinâmica. A prioridade estática não tem o seu valor alterado durante a existência do processo, enquanto a prioridade dinâmica pode ser ajustada de acordo com critérios definidos pelo sistema operacional. A possibilidade de alterar o valor da prioridade de um processo ao longo de seu processamento permite ajustar o critério de escalonamento em função do comportamento de cada processo no sistema.

Um dos principais problemas no escalonamento por prioridades é o *starvation*. Processos de baixa prioridade podem não ser escalonados, permanecendo indefinidamente na fila de pronto. Uma solução para este problema, possível em sistemas que implementam prioridade dinâmica, é a técnica de *aging*. Este mecanismo incrementa gradualmente a prioridade de processos que permanecem por muito tempo na fila de pronto.

O escalonamento por prioridades possibilita diferenciar os processos segundo critérios de importância. Com isso, processos de maior prioridade são escalonados preferencialmente. Isto é bastante útil, tanto em sistemas de tempo real e nas aplicações de controle de processo como também em aplicações de sistemas de tempo compartilhado, em que, às vezes, é necessário priorizar o escalonamento de determinados processos.

## Conclusão sobre Escalonamento por Prioridades

A política de escalonamento por prioridades já resolve vários problemas que vimos, por exemplo, nas outras políticas de escalonamento até aqui. O importante é manter o escalonamento por prioridades como algo “democrático”, em que existe mecanismos que se atentam aos processos que estão sempre esperando e ficam envelhecendo na fila de pronto. Esse mesmo mecanismo (ou no caso, com a mesma ideia: prevenir que um processo entre em *starvation*) é visto no SJF, que nesse caso, processos enormes são sempre deixados por último e podem sofrer a inanição. E no caso do SJF, temos o HRRN, que é muito similar ao SJF, porém leva em consideração o envelhecimento do processo, e quando esse tempo fica muito alto, o processo entra em execução mesmo sendo maior que os outros que também aguardam na fila de pronto.

O sistema de escalonamento também é muito usado em sistemas de tempo real, visto que é necessário a otimização do tempo para tarefas críticas. Basta lembrar dos exemplos que citei no começo do relatório, nas conclusões sobre escalonamentos preemptivos e não-preemptivos.

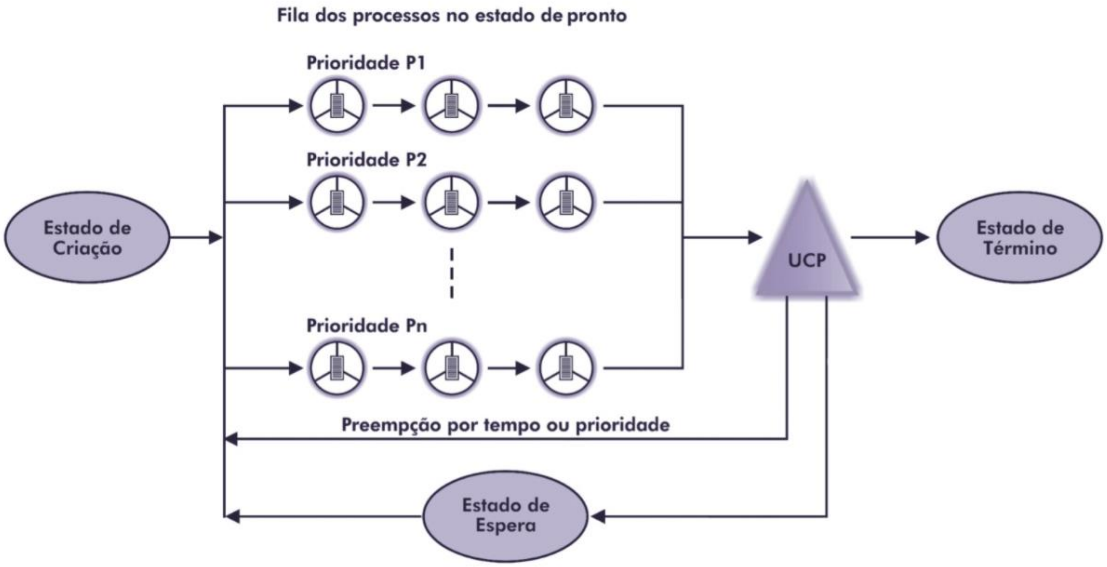
## Escalonamento Circular com Prioridades

O escalonamento circular com prioridades implementa o conceito de fatia de tempo e de prioridade de execução associada a cada processo. Neste tipo de escalonamento, um processo permanece no estado de execução até que termine seu processamento, voluntariamente passe para

o estado de espera ou sofra uma preempção por tempo ou prioridade, como é possível ver na imagem que vêm a seguir.

A principal vantagem deste escalonamento é permitir o melhor balanceamento no uso do processador em sistemas de tempo compartilhado. Processos com o perfil *I/O-bound* devem receber do administrador do sistema prioridades com valores maiores que as dos processos *CPU-bound*. Isso permite ao sistema operacional praticar uma política compensatória entre processos de perfis distintos, compartilhando o processador de forma mais igualitária. Este tipo de escalonamento é amplamente utilizado em sistemas de tempo compartilhado, como o Windows e o Unix.

O escalonamento circular com prioridades possui duas variações, dependendo se a prioridade é do tipo estática ou dinâmica. No escalonamento circular com prioridades estáticas, a prioridade definida no contexto de software de cada processo permanece inalterada ao longo da sua existência.




No caso do escalonamento circular com prioridades dinâmicas, é possível que a prioridade de um processo seja alterada dinamicamente pelo administrador do sistema ou, em algumas políticas, pelo próprio sistema operacional.

O ajuste dinâmico das prioridades dos processos pelo sistema operacional com base no seu comportamento é outro exemplo de um mecanismo adaptativo. Neste caso específico, o mecanismo possibilita um refinamento no balanceamento do uso do processador por processos de diferentes perfis com base no tipo de operação de E/S realizada. Algumas destas operações são mais lentas, fazendo com que um processo permaneça mais tempo no estado de espera sem chances de competir pelo uso do processador. Neste caso, quando o processo sai do estado de espera, um acréscimo temporário à sua prioridade é concedido pelo sistema operacional. Dessa

forma, os processos têm mais chances de serem escalonados rapidamente, compensando parcialmente o tempo gasto no estado de espera. É importante perceber que os processos *CPU-bound* não são prejudicados com esta política, pois podem ser executados durante o período em que os processos *I/O-bound* estão no estado de espera.

## Conclusão sobre Escalonamento Circular com Prioridades

Aqui entramos numa política de escalonamento amplamente usada, e que faz todo o sentido. Pois a preempção por tempo evita o *starvation* que abordamos na política de escalonamento anterior. Enquanto o escalonamento por prioridades está sujeito a deixar processos no estado de inanição e o escalonamento circular está sujeito a deixar a prioridade de lado, e acabar não "priorizando" processos que precisam de atenção. Esta política resolve muito dos problemas vistos nos tópicos anteriores. A preempção que ocorre nesse caso torna todo o gerenciamento mais "democrático".

## Escalonamento por Múltiplas Filas

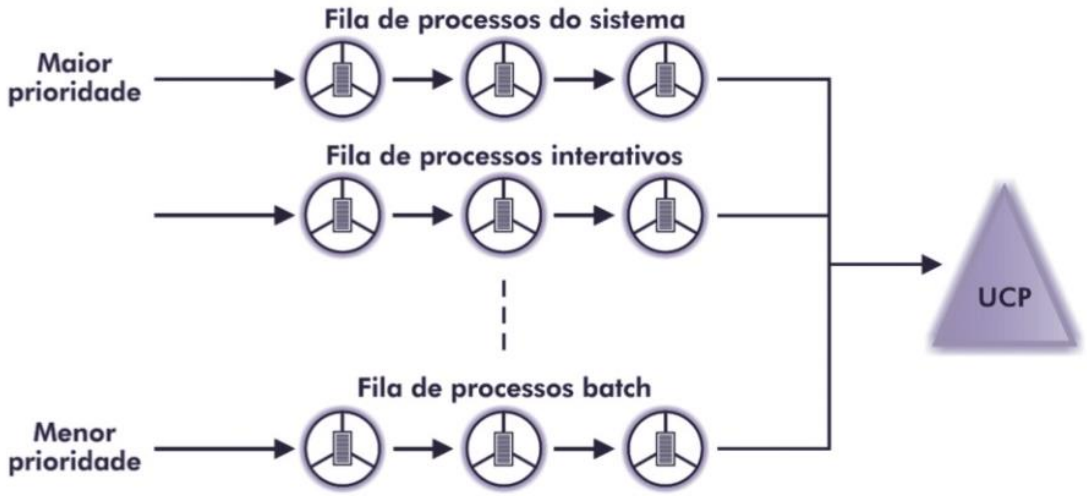
No escalonamento por múltiplas filas (*multilevel queue scheduling*) existem diversas filas de processos no estado de pronto, cada qual com uma prioridade específica. Os processos são associados às filas em função de características próprias, como importância para a aplicação, tipo de processamento ou área de memória necessária.

Como processos possuem características de processamento distintas, é difícil que um único mecanismo de escalonamento seja adequado a todos. A principal vantagem de múltiplas filas é a possibilidade da convivência de mecanismos de escalonamento distintos em um mesmo sistema operacional. Cada fila possui um mecanismo próprio, permitindo que alguns processos sejam escalonados pelo mecanismo FIFO, enquanto outros pelo circular.

Neste mecanismo, o processo não possui prioridade, ficando desta forma, esta característica associada à fila. O processo em execução sofre preempção caso um outro processo entre em uma fila de maior prioridade. O sistema operacional só pode escalonar processos de uma determinada fila caso todas as outras filas de maior prioridade estejam vazias. Uma boa prática é classificar os processos em função do tipo de processamento realizado e associa-los adequadamente às respectivas filas.

Para exemplificarmos esse escalonamento, considere que os processos sejam divididos em três grupos: sistema, interativo e batch – representado na figura abaixo –. Os processos do sistema devem ser colocados em uma fila de prioridade mais alta que a dos outros processos, implementando um algoritmo de escalonamento baseado em prioridades. Os processos de usuários interativos devem estar em uma fila de prioridade intermediaria, implementando o

escalonamento circular. O mesmo mecanismo de escalonamento pode ser utilizado na fila de processos batch, com a diferença de que esta fila deve possuir uma prioridade mais baixa.

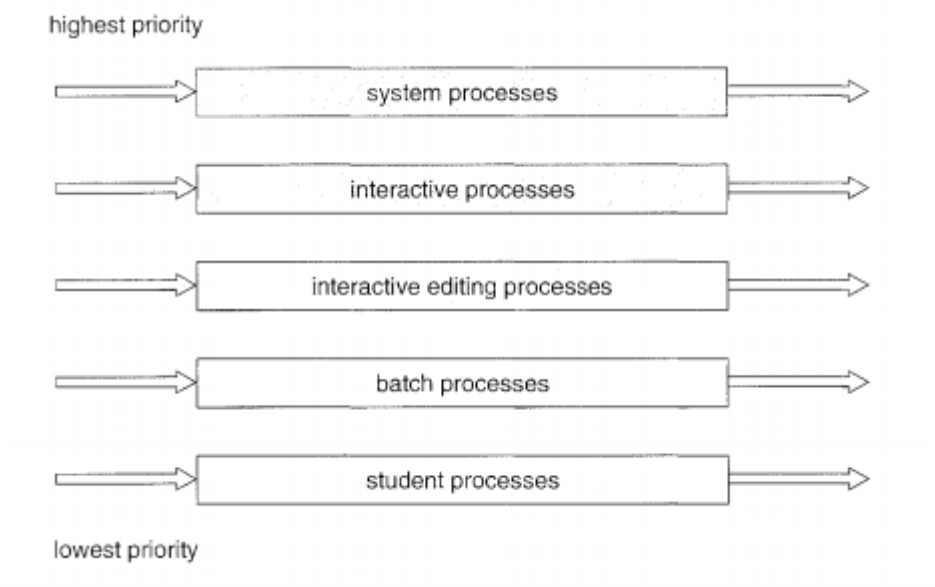



Uma desvantagem deste escalonamento é que no caso de um processo alterar seu comportamento no decorrer do tempo o processo não poderá ser redirecionado para uma outra fila mais adequada. A associação de um processo à fila é determinada na criação do processo, permanecendo até o término do seu processamento.

## Conclusão sobre Escalonamento por Múltiplas Filas

Além de maximizar o *throughput* e minimizar o *turnaround*, aumentar a utilização da CPU e minimizar tempo de espera e resposta, essa política de escalonamento também não degrada o sistema, pois segundo Silberschatz, Galvin e Gagne, no livro "*Operating system concepts 8th ed. Hoboken*" mais formas de escalonamento ocorrerão, fazendo com que o máximo de processos receba recursos, aplicando-se prioridade dinâmica e formando um ciclo que evita que eles entrem em *starvation*. Na imagem a baixo é possível ver com mais clareza como isso se aplicaria em um sistema.

Cada fila tem prioridade absoluta sobre as filas de prioridade mais baixa. Por exemplo: nenhum processo na fila de *batch* poderia ser executado a menos que as filas *system*, *interactive* e *interactive editing* estejam todas vazias. Se um processo de *interactive editing* entrou na fila de processos prontos para execução enquanto um processo em *batch* estava em execução, o processo em *batch* seria interrompido.

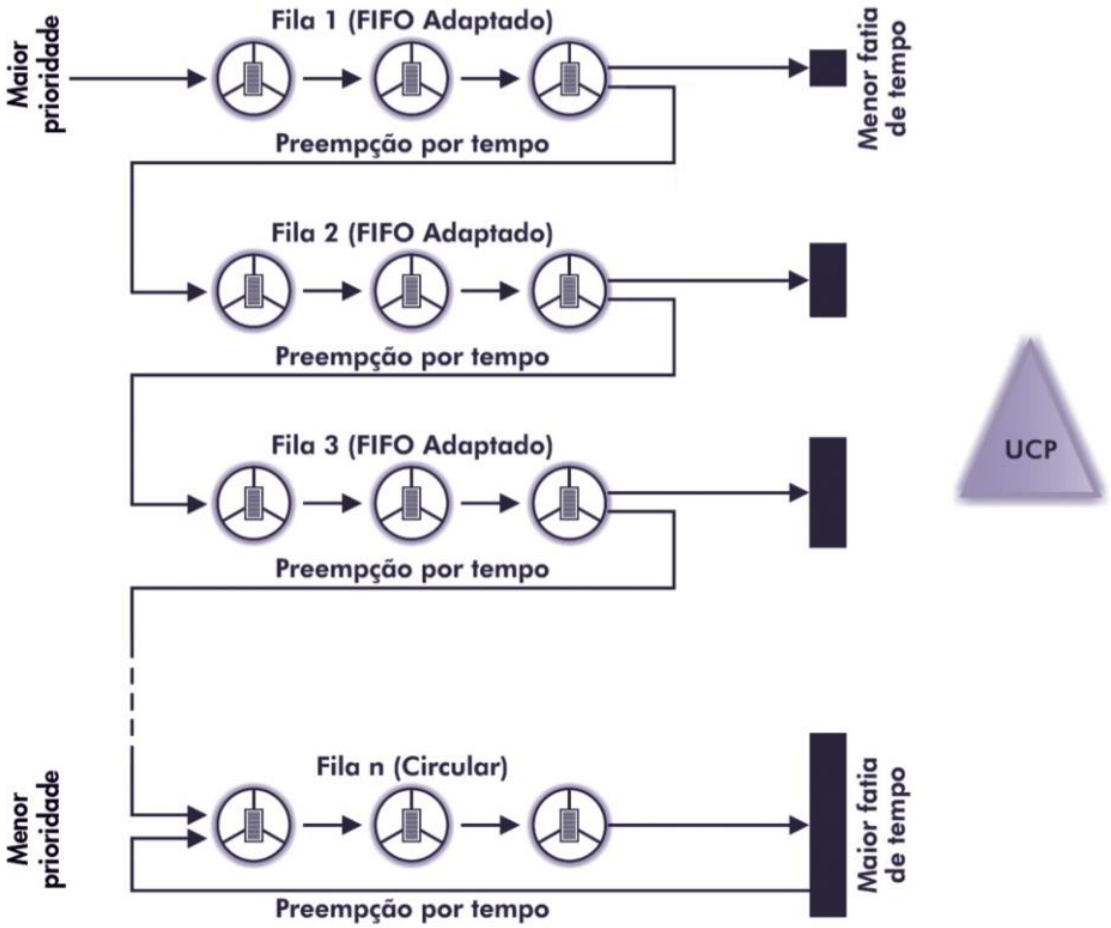



## Escalonamento por Múltiplas Filas com Realimentação

O escalonamento por múltiplas filas com realimentação (*multilevel feedback queues scheduling*) é semelhante ao escalonamento por múltiplas filas, porém os processos podem trocar de filas durante seu processamento. Sua grande vantagem é permitir ao sistema operacional identificar dinamicamente o comportamento de cada processo, direcionando-o para filas com prioridade de execução e mecanismo de escalonamento mais adequados ao longo de seu processamento.

Esse esquema permite que os processos sejam redirecionados entre as diversas filas, fazendo com que o sistema operacional implemente um mecanismo adaptativo. Os processos não são previamente associados às filas de pronto, e, sim, direcionados pelo sistema para as filas existentes com base no seu comportamento.

Um mecanismo FIFO adaptado com fatia de tempo é implementado para escalonamento em todas as filas, com exceção da fila de menor prioridade, que utiliza o escalonamento circular. O escalonamento de um processo em uma fila ocorre apenas quando todas as outras filas de prioridades mais altas estiverem vazias. A fatia de tempo em cada fila varia em função da sua prioridade, ou seja, quanto maior a prioridade da fila menor a sua fatia de tempo, é possível visualizar de maneira mais clara na imagem abaixo. Assim, a fatia de tempo concedida aos processos varia em função da fila de pronto na qual ele se encontra. Um processo, quando criado, entra no final da fila de maior prioridade, porém durante sua execução, a cada preempção por tempo, o processo é redirecionado para uma fila de menor prioridade.




Esse escalonamento atende às necessidades dos diversos tipos de processos. No caso de processos *I/O-bound*, um tempo de resposta adequado é obtido, já́ que esses processos têm prioridades mais altas por permanecerem a maior parte do tempo nas filas de maior prioridade, pois dificilmente sofrerão preempção por tempo. Por outro lado, em processos *CPU-bound* a tendência é de que, ao entrar na fila de mais alta prioridade, o processo ganhe o processador, gaste sua fatia de tempo de tempo e seja direcionado para uma fila de menor prioridade. Dessa forma, quanto mais tempo do processador um processo utiliza, mais ele vai caindo para filas de menor prioridade.

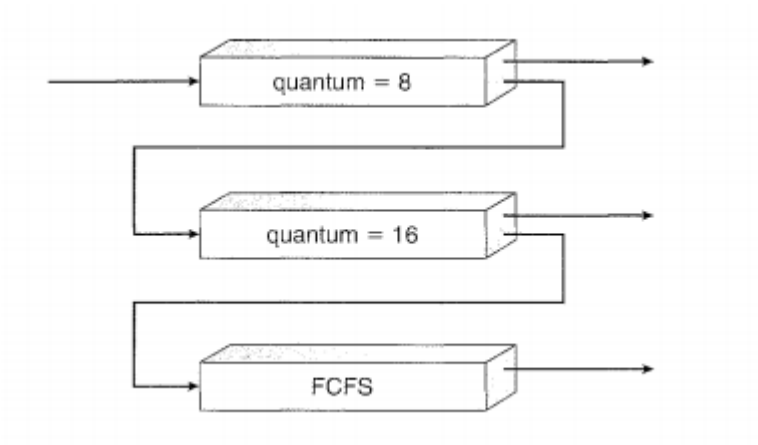
O escalonamento por múltiplas filas com realimentação é um algoritmo de escalonamento generalista, podendo ser implementado em qualquer tipo de sistema operacional. Um dos problemas encontrados nesta política é que a mudança de comportamento de um processo *CPU-bound* para *I/O-bound* pode comprometer seu tempo de resposta. Outro aspecto a ser considerado é sua complexidade de implementação, ocasionando um grande *overhead* ao sistema.

## Conclusão sobre Escalonamento por Múltiplas Filas com Realimentação

Ainda nas palavras do livro *Operating system concepts 8th ed. Hoboken:* o escalonamento por múltiplas fitas é definido pelos seguintes parâmetros:

1. O número de filas;
2. O algoritmo de escalonamento de cada fila;
3. O método usado para determinar quando atualizar um processo para uma fila de maior prioridade;
4. O método usado para determinar quando rebaixar um processo para uma fila de prioridade mais baixa;
5. O método usado para determinar em qual fila um processo entrará quando aquele processo precisa de serviço.

A definição de um escalonador por múltiplas fitas o torna o mais genérico algoritmo de escalonamento de CPU. Pode ser configurado para corresponder a um sistema em projeto. Infelizmente, é também o algoritmo mais complexo, uma vez que definir o melhor escalonador requer alguns meios para selecionar valores para todos os parâmetros.



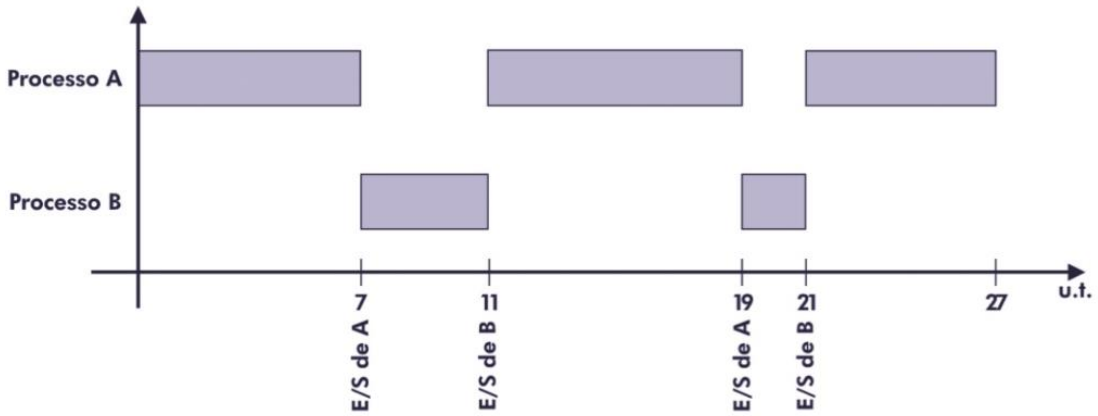

Este algoritmo de escalonamento dá maior prioridade a qualquer processo com um *burst* de CPU de 8 milissegundos ou menos. Esse processo irá rapidamente obter a CPU, terminar seu *burst* de CPU e vai para o próximo burst de I/O. Processos que precisam mais do que 8, mas menos de 24 milissegundos também são servidos rapidamente, embora com menor prioridade do que processos mais curtos. Processos longos vão automaticamente para a fila 2 e são atendidos na ordem FCFS com quaisquer ciclos de CPU que sobraram das filas 0 e 1. Esse exemplo é, em minha opinião, muito bem compreensível. Pois agora as políticas de escalonamento estão cada vez mais complexas, e mais atuais também.

## Política de Escalonamento em Sistemas de Tempo Compartilhado

Em geral, sistemas de tempo compartilhado caracterizam-se pelo processamento interativo, no qual usuários interagem com as aplicações exigindo tempos de respostas baixos. A escolha de uma política de escalonamento para atingir este propósito deve levar em consideração o compartilhamento dos recursos de forma equitativa para possibilitar o uso balanceado da UCP

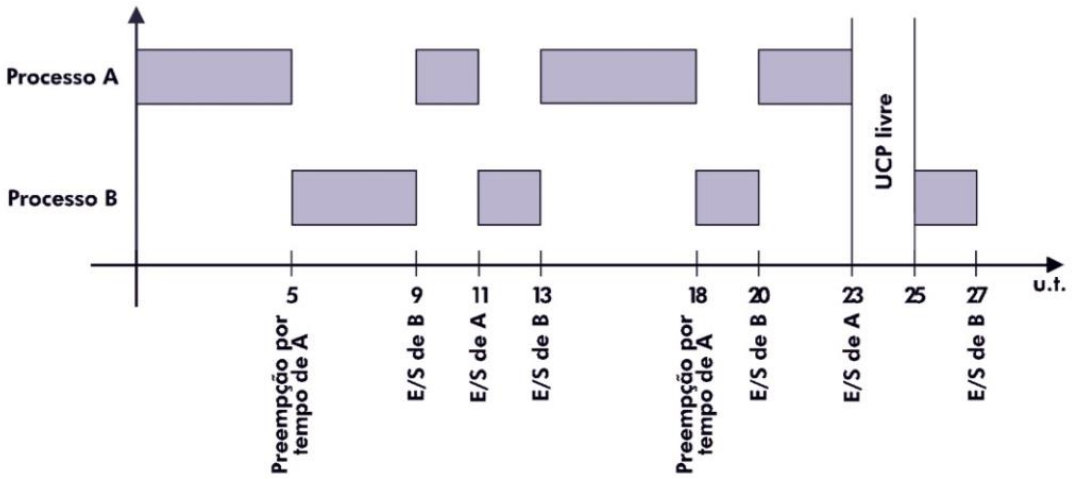
entre processos. Neste item, serão analisados os comportamentos de um processo *CPU-bound* e outro *I/O-bound*, nos principais escalonamento apresentados.

Na figura abaixo, os processos A (*CPU-bound*) e B (*I/O-bound*) são escalonados segundo o mecanismo FIFO. Se contabilizarmos o uso da UCP para cada processo no instante de tempo 27, constataremos que o processador está sendo distribuído de forma bastante desigual (21 u.t. para o Processo A e 6 u.t. para o Processo B). Como a característica do Processo B é realizar muitas operações de E/S, em grande parte do tempo o processo permanecerá no estado de espera.




| Processo | Tempo de processador (u.t.) | Característica |
|---|---|---|
| A | 21 | CPU-bound |
| B | 6 | I/O-bound |

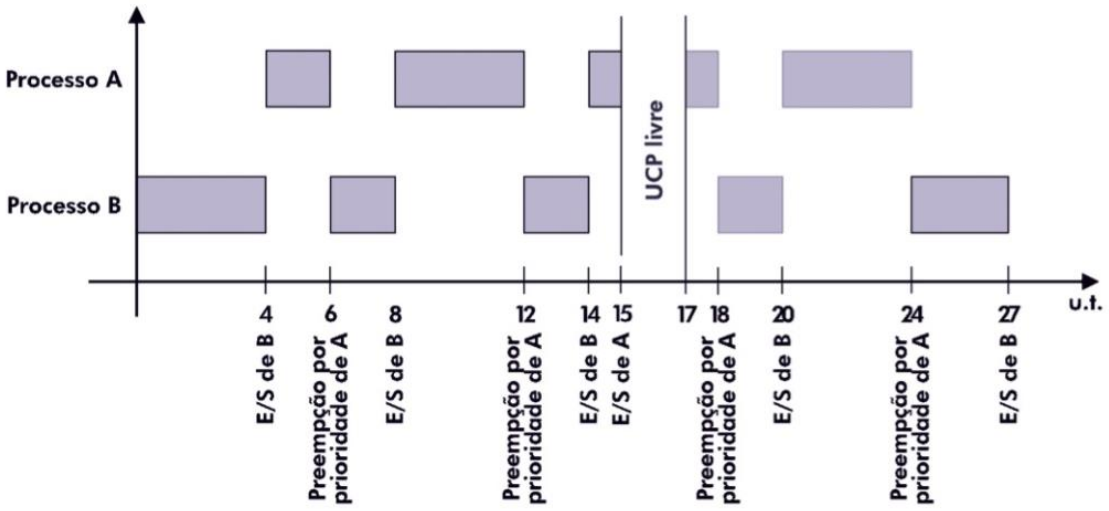
A próxima figura, que segue logo abaixo deste trecho apresenta os mesmos processos A e B, só que aplicando o escalonamento circular com fatia de tempo igual a cinco unidades de tempo. Com isso, o escalonamento circular consegue melhorar a distribuição do tempo de processador (15 u.t. para o Processo A e 10 u.t. para o Processo B) em relação ao escalonamento FIFO, porém ainda não consegue implementar um compartilhamento equitativo entre os diferentes tipos de processos. Esta deficiência no escalonamento circular deve-se ao fato de todos os processos serem tratados de uma maneira igual, o que nem sempre é desejável.




| Processo | Tempo de processador (u.t.) | Característica |
|---|---|---|
| A | 15 | CPU-bound |
| B | 10 | I/O-bound |

Em um escalonamento onde todos os processos são tratados igualmente, processos *CPU-bound* sempre levam vantagem sobre processos *I/O-bound* no uso do processador. Como no escalonamento circular um processo *I/O-bound* compete pelo processador da mesma forma que um processo *CPU-bound* e o processo *I/O-bound* passa a maior parte do tempo no estado de espera, o processo *CPU-bound* tem mais oportunidade de ser executado.

No escalonamento circular com prioridades é possível associar prioridades maiores aos processos *I/O-bound*, a fim de compensar o excessivo tempo gasto por este tipo de processo no estado de espera. No exemplo da que segue logo após esse trecho do texto, o Processo B tem uma prioridade superior à do Processo A, obtendo-se, desta forma, um maior grau de compartilhamento no uso do processador (12 u.t. para o Processo A e 13 u.t. para o Processo B).




| Processo | Tempo de processador (u.t.) | Característica | Prioridade |
|---|---|---|---|
| A | 12 | CPU-bound | Baixa |
| B | 13 | I/O-bound | Alta |

## Conclusão sobre Política de Escalonamento em Sistemas de Tempo Compartilhado

Estas políticas de escalonamento são, por exemplo, aquelas que usamos em casa este computador que uso para editar este relatório que está com cara de seminário. Infelizmente não encontrei muitos documentos que falassem sobre essa política de escalonamento, contudo, vale lembrar que vimos muito isso nas aulas de Sistemas Operacionais lecionada pelo professor Fábio, e forma simples e objetiva é possível fazer algumas afirmações que nos fazem entender melhor essa política de escalonamento:

1. Um processo *CPU-bound* sempre leva vantagem sobre um processo *I/O-bound*. Isso porque o processo *I/O-bound* estará em espera na maior parte do tempo. Por exemplo: este editor de texto que estou usando está – esperando – eu salvar o documento para efetuar o processo de salvamento em disco. Isto é, enquanto eu não apertar CTRL + B no meu Office Word ele não salvará, e outros processos estão executando à frente deste processo de salvamento, pois esse processo depende da minha interação.
2. Os sistemas com política de escalonamento de tempo compartilhado costumam aumentar a prioridade de processos *I/O-bound* afim de minimizar o tempo desses processos, já que se encontram a maior parte do tempo no modo de espera.

Embora os sistemas com prioridade dinâmica sejam mais complexos de implementar que os sistemas com prioridade estática, o tempo de resposta oferecido compensa. Atualmente, a maioria dos sistemas operacionais de tempo compartilhado utiliza o escalonamento circular com prioridades dinâmicas.

## Política de Escalonamento em Sistemas de Tempo Real

Diferentemente dos sistemas de tempo compartilhado, nos quais a aplicação não é prejudicada pela variação no tempo de resposta, algumas aplicações especificas exigem respostas imediatas para a execução de determinadas tarefas. Neste caso, a aplicação deve ser executada em sistemas operacionais de tempo real, onde é garantida a execução de processos dentro de limites rígidos de tempo, sem o risco de a aplicação ficar comprometida. Aplicações de controle de processos, como sistemas de controle de produção de bens industriais e controle de tráfego aéreo, são exemplos de aplicação de tempo real.

O escalonamento em sistemas de tempo real deve levar em consideração a importância relativa de cada tarefa na aplicação. Em função disso, o escalonamento por prioridades é o mais adequado, já que para cada processo uma prioridade é associada em função da importância do processo dentro da aplicação. No escalonamento para sistemas de tempo real não deve existir o conceito de fatia de tempo, e a prioridade de cada processo deve ser estática.

## Conclusão sobre Política de Escalonamento em Sistemas de Tempo Real

Em sistemas onde as noções de tempo e de concorrência são tratadas explicitamente, conceitos e técnicas de escalonamento formam o ponto central na previsibilidade do comportamento de sistemas de tempo real. Nos últimos anos, uma quantidade significativa de novos algoritmos e de abordagens foi introduzida na literatura tratando de escalonamento de tempo real. Infelizmente muitos desses trabalhos definem técnicas restritas e consequentemente de uso limitado em aplicações reais.

Podemos ver observando a política de escalonamento de tempo compartilhado e traçando um paralelo com a política de escalonamento em sistemas de tempo real que enquanto a primeira se preocupa mais com o conforto do usuário final, como um documento de texto que salva quase que imediatamente, a segunda se preocupa em tempo de resposta para processos que exigem atenção, afinal, esses sistemas de tempo real são usados nas áreas de saúde, trafego aéreo, usinas nucleares e outros campos onde exige um sistema de resposta imediata, e onde a otimização e a priorização dos processos certos é importantíssima. Seguindo essa linha e sumarizando as características da política de escalonamento em sistemas de tempo real em comparação à política de escalonamento de tempo compartilhado podemos dizer:

1. O uso dos recursos computacionais de um sistema de tempo real é definido pela prioridade que a aplicação indicar. Exemplo: uma daquelas máquinas do hospital que monitora batimentos cardíacos e sinais vitais de um paciente.
2. Não há fatia de tempo. Exemplo: seguindo o exemplo anterior, imagine parar por alguns segundos de monitorar os batimentos cardíacos e sinais vitais de um paciente, não é uma opção válida, então a política de *time-slice* ou *quantum*, como também é chamada, está fora de questão.
3. Esses sistemas não são destinados ao usuário final, como dito anteriormente. São sistemas com aplicações para situações críticas. Aplicações mais comerciais usaram quase sempre (se não sempre) sistemas de tempo compartilhado para oferecer melhor experiência de usuário.

Apesar de não ter encontrado muita coisa sobre sistemas de tempo real para complementar essa parte do trabalho, ou pelo menos nada que fosse compreensível para nós aspirantes de sistemas operacionais, lendo algo que parece um seminário ou artigo científico da UFPE encontrei esse escalonamento que achei válido citar:

## Escalonamento *"Earliest Deadline First"* (EDF)

O *"Earliest Deadline First"* (EDF) define um escalonamento baseado em prioridades: a escala é produzida em tempo de execução por um escalonador preemptivo dirigido a prioridades. É um esquema de prioridades dinâmicas com um escalonamento *"on-line"* e dinâmico. O EDF é um algoritmo ótimo na classe dos escalonamentos de prioridade dinâmica. As premissas que determinam o modelo de tarefas no EDF são:

1. Os processos são periódicos e independentes.
2. O *"deadline"* de cada processo coincide com o seu período ($D_i=P_i$). (Onde D é *deadline* e P é período)
3. O tempo de computação ($C_i$) de cada processo é conhecido e constante ("*Worst Case Computation Time*").
4. O tempo de chaveamento entre tarefas é assumido como nulo.

A política de escalonamento no EDF corresponde a uma atribuição dinâmica de prioridades que define a ordenação das tarefas segundo os seus *"deadlines"* absolutos ($d_i$). A tarefa mais prioritária é a que tem o *"deadline"* $d_i$ mais próximo do tempo atual. A cada chegada de tarefa a fila de prontos é reordenada, considerando a nova distribuição de prioridades. A cada ativação de uma tarefa $T_i$, um novo valor de *"deadline"* absoluto é determinado considerando o número de períodos que antecede a atual ativação ($k$): $d_{ik}=kP_i$. No EDF, a escalabilidade é também verificada em tempo de projeto, tomando como base a utilização do processador.

## Conclusão do Relatório sobre Gerência do Processador

Acredito que após toda essa leitura vou acabar lendo mais sobre Sistemas Operacionais, principalmente esses que usamos para nossas atividades do dia-a-dia. Observando a gestão do processador sobre os processos é possível enxergar falhas, falhas que podem ser facilmente utilizadas para, literalmente, arrebentar com uma CPU; tanto com softwares maliciosos que conseguem confundir as políticas de escalonamento do processador em si quanto softwares que roubam recursos do processador e rodam em background. Particularmente adoro segurança de computadores e sempre penso nisso quando abordo um tema de computação.

Parte da metodologia de desenvolvimento do assunto foi inspirada no site *Refactoring Guru*, site que aborda padrões de projetos e trabalha muito bem os assuntos com uma metodologia muito simples e utilizando exemplos da vida real para ilustrar situações de software.

Muitas fontes usavam o mesmo livro que usamos na aula, inclusive esse livro também tem o problema dos filósofos e os garfos, então esse livro é, basicamente, o guia para sistemas operacionais. Igualmente citado, mas por fontes estrangeiras, foi o livro *Operating system concepts 8th ed. Hoboken*, que é um livro de quase mil páginas que utilizei para minhas conclusões sobre muitos tópicos, essas e as outras referências que utilizei ao longo do trabalho estão à disposição nas referências do relatório.

Observando todas essas políticas de escalonamento e tipos de sistemas e situações e variáveis que são levadas em consideração quando não foram antes porque solucionamos um problema mas então outro problema apareceu é até inspirador, pois em tecnologia sempre é assim: você não se livra de problemas, você só eleva os problemas para um nível de complexidade maior e resolve problemas que não eram possíveis resolver até então. Ainda há muito a se acrescentar na parte de sistemas operacionais, há mais complexidade a se descobrir na arquitetura dos processadores.

infelizmente os métodos de escalonamento em tempo real são, em sua maioria, propriedade privada, por conta da sensibilidade que muitos deles tratam, como disse anteriormente, se não teríamos a oportunidade de ver códigos de complexidade imensa.

Os primeiros escalonamentos eram falhos em alguns aspectos, mas ainda servem de molde para os novos, como o FIFO ainda é, em parte, usado nas múltiplas filas com realimentação. A evolução dos escalonamentos e o aumento da eficácia trazida para o processador nos permite visualizar as possibilidades e os futuros métodos de escalonamento que estão por vir.

## Referências


Autor desconhecido, ano desconhecido. Trabalho da USP sobre gerência do processador. https://www.ime.usp.br/~adao/TGP.pdf

Francis Berenger Machado e Luiz Paulo Maia, 2007. Arquitetura de Computadores edição 4. Baixado neste link: https://wiac.info/docdownloadv2-pdf-arquitetura-de-sistemas-operacionais-francis-berenger-machadopdf-dl_b38a3c4e20b54307a12b5905008fe50c

Professor Fernando de Siqueira. Ano desconhecido. Aula 8: Gerência do Processador https://sites.google.com/site/proffernandosiqueiraso/aulas/8-gerencia-do-processador

Professor Edwar Saliba Júnior, junho de 2009. Fundamentos de Sistemas Operacionais, Gerência de Processador. http://esj.eti.br/INED/FSO/FSO_Unidade_03_003.pdf

SILVA, Guilherme Baião S. Slides da disciplina de Sistemas Operacionais de Arquitetura Fechada. Faculdade INED, 2005.

*Refactoring Guru*, 2014. Padrões de Projeto de forma simples e objetiva. https://refactoring.guru/pt-br/design-patterns/strategy

*Master Of Sciences* Marcelo de Paiva Guimarães, ano desconhecido. Sistemas de Tempo Real. Doutorando da Universidade de São Paulo. http://www.lsi.usp.br/~paiva/sd/Sistemas%20de%20Tempo%20Real.pdf

Autor desconhecido, ano desconhecido. Sistemas Operacionais e Gerência do Processador (Escalonamento de Processos): UFSC https://www.gsigma.ufsc.br/~popov/aulas/so1/cap8so.html

Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C., 2014, *Operating Systems: Three Easy Pieces, Chapter Scheduling Introduction*. É possível baixar no link abaixo: http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

Autor desconhecido, 2017. Algoritmos de escalonamento de Processos. http://nasemanadaprova.blogspot.com/2015/12/algoritmos-de-escalonamento-de-processos.html

Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg (2008). *Operating system concepts* 8th ed. Hoboken, N.J.: Wiley. Pode ser baixado através desse link: http://www.uobabylon.edu.iq/download/M.S%202013-2014/Operating_System_Concepts,_8th_Edition%5BA4%5D.pdf

Autor desconhecido, ano desconhecido. Trabalho sobre Sistemas de tempo real da UFPE https://www.cin.ufpe.br/~if728/sistemas_tempo_real/livro_farines/cap2.pdf

Os slides da aula do professor :)