Universidade São Judas Tadeu
Ciência da Computação

Cecília de Oliveira Martins - RA:81620964
Dennis Siqueira de Oliveira - RA:81621468
Douglas de Oliveira Lima - RA:817124153
Gabriel Ferreira da Silva - RA: 81621851
Wellington Shiniti Kawashima - RA: 81622278

Projeto Integrado - Escalonamento de Processos

Butantã - SP
2019

Cecília de Oliveira Martins - RA:81620964
Dennis Siqueira de Oliveira - RA:81621468
Douglas de Oliveira Lima - RA:817124153
Gabriel Ferreira da Silva - RA: 81621851
Wellington Shiniti Kawashima - RA: 81622278

Projeto Integrado - Escalonamento de Processos

Trabalho de Projeto Integrado do curso de Ciência da Computação apresentado a Universidade São Judas Tadeu.
Orientadora: Prof.ª. Dra. Milkes Alvarenga

Butantã - SP
2019

Lista de figuras

Lista de Tabelas

Sumário

1. Introdução

Este trabalho tem como objetivo a apresentação do estudo realizado sobre escalonamento de processos. Nele serão encontradas definições, exemplos, figuras e citações sobre o assunto.

No universo da computação os Sistemas Operacionais exercem um papel de muita a importância, mais especificamente é um intermediário entre o usuário e hardware do computador. O escalonamento de processos é utilizado no gerenciamento da execução das tarefas (programas), designando a melhor forma de executá-las para o sistema operacional, assim apresentando uma melhor usabilidade para o usuário.

Este trabalho está organizado em 3 capítulos. No capítulo 2, será abordado os conceitos de processos de um programa/tarefa. No capítulo 3, optamos em abordar o conceito de escalonamento de processos. No capítulo 4, demonstramos os principais tipos escalonamento.

A metodologia utilizada foi pesquisa utilizada por livro, enriquecida com alguns artigos em sites sobre o tema.

2. Processos

Um processo é um programa em execução incluindo os valores correntes de todos os registradores do hardware, e das variáveis manipuladas por ele no curso da execução. [6]

Os processadores executam instruções que são representados em algoritmos, onde um algoritmo é programado em alguma linguagem de programação (C, C#, JAVA, Python, etc.). [6]

Um processo pode ser iniciado de acordo com os seguintes eventos:

- Início do sistema operacional;
- Execução de chamadas ao sistema de criação de processos;
- Solicitação do usuário para criar um novo processo;
- Início de um trabalho em um lote.

E um processo pode ser finalizado com as seguintes condições:

- Saída Normal;
- Saída por Erro;
- Erro Fatal;
- Cancelamento por outro processo;

2.1 Hierarquia de Processos

A execução de um programa pode gerar a criação de novos processos filhos que por sua vez, também podem criar novos processos filhos, isto é chamado de grupo de processos. [6]

2.2 Classificação dos processos

Os processos são classificados de duas formas, processos pesados e processos leves. [6]

Os processos pesados são os processos tradicionais, possuem uma thread inicial que começa a sua execução (ex: main de um programa C), executam um código sequencial e normalmente são carregados do disco para execução. [6]

Os processos leves (threads) são criados para permitir paralelismo na execução de um processo pesado. As principais características dos processos leves são:

- Cada um roda um código sequencial;
- Possuem sua própria pilha de execução e o seu próprio program conter;
- Compartilham o uso do processador;
- Podem criar processos (threads) filhos;
- Compartilham o mesmo espaço de endereçamento (dados globais).

Um processo (Pesado ou Leve) está no estado “executando” se suas instruções estão sendo executadas pelo processador. Está em um estado “pronto” se possui as condições para ser executado e está esperando pelo processador, e está em um estado “bloqueado” se está à espera de alguma condição para rodar, por exemplo, está à espera de uma operação de entrada e saída. A Figura 1 “Estados de execução do programa” a seguir ilustra a transição de estados do processo. [6]

Figura 1: Estados de execução do programa

Um processo passa pelo estado de “pronto” para o estado de “executando” quando é selecionado, ganha o processador e suas instruções começam a ser executadas. Passa de um estado “bloqueado” para “pronto” quando ocorre o evento pelo qual estava esperando e passa de um estado de “executando” para bloqueado quando necessita esperar pela ocorrência de um evento (ex: operação de entrada e saída). A transição de "executando” para “pronto” ocorre em sistemas que atribuem fatias de tempo do processador aos processos em execução. Quando esse tempo termina, o processo perde o processador e passa para o estado “pronto”, à espera novamente do processador. [6]

Threads são processos leves, definidos para permitir paralelismo em um programa de aplicação. Um programa com múltiplos pontos de execução possui tantas threads de execução quantos forem esses pontos. As Threads possuem sua própria pilha de execução e o seu próprio program counter, cada uma roda um código sequencial, definido no programa de aplicação (processo pesado) no qual foram definidas, compartilham o uso do processador, podem criar processos (threads) filhos e compartilham o mesmo espaço de endereçamento (dados globais) do processo pesado ao qual pertencem. Portanto, quando uma thread altera um valor compartilhado, todas percebem essa mudança e os arquivos abertos por uma thread são disponíveis às outras threads do mesmo processo pesado. Para manter a coerência dos dados, mecanismos de sincronização devem ser utilizados pelo programador. [6]

As threads sofrem transição entre os estados de “pronto”, “executando”, “bloqueado”, não existe estado “trocado”, isto é, uma thread não pode durante a sua execução ser retirada da memória principal e gravada em disco para liberar memória e permitir que um novo programa possa ser carregado para execução. A operação de “troca” não existe para um thread, somente para todo o processo pesado a qual a thread faz parte. Outra característica é que a terminação do processo pesado termina todas as threads do processo. [6]

3. Escalonamento de Processos

Escalonamento de processos é o ato de se realizar o chaveamento dos processos ativos, de acordo com regras bem estabelecidas, fazendo assim que todos os processos tenham chances de utilizar a UCP (Unidade Central de Processamento). É responsável por garantir produtividade e eficiência para o Sistema de Computação, sempre visando pelos melhores resultados possíveis. [5]

O escalonador é a parte do SO (Sistema Operacional) encarregada de decidir qual dentre os processos prontos serão colocados em execução. Vários critérios podem ser definidos para a avaliação dos escalonadores. [5]

Os mais frequentes são:

- Utilização da CPU: A CPU tem de ficar o mais ocupada possível. A porcentagem de tempo em que a CPU ficou trabalhando deve ser em torno de 40% a 90%.
- Tempo de Execução (TurnARound): Mede o tempo decorrido entre a criação e o encerramento da tarefa, computando todos os tempos de processamento e de espera.
- Tempo de Espera (Waiting Time): Tempo total perdido pela tarefa na fila de prontos, aguardando o processador.
- Tempo de Resposta: Tempo decorrido entre a chegada de um evento ao sistema e o resultado imediato de seu processamento.
- Eficiência (Produção ou Throughput): Indica o grau de utilização do processador na execução das tarefas do usuário.
- Justiça: Distribuição do processador entre as tarefas prontas.

3.1. Quando se deve escalonar?

Somente se escalona quando:

- Se cria um novo processo;
- No término de um processo;
- Quando um processo é bloqueado;
- Quando um processo executa o evento de E/S (Entrada/Saída).

O algoritmo de escalonamento pode ser:

- **Não Preemptivo**: O processo executa até o fim, sem ser interrompido;
- **Preemptivo**: O processo executa em fatias de tempo ***(quantum)*** determinado pelo sistema operacional.

Basicamente, no escalonamento *Preemptivo*, o sistema operacional pode interromper um processo em execução e passá-lo para o estado de pronto, com o objetivo de alocar outro processo na UCP. No escalonamento *Não-Preemptivo*, quando um processo está em execução, nenhum evento 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. [5]

3.2. Dispatcher

Dispatcher nada mais é do que o responsável pela troca de contexto dos processos após o escalonador terminar qual processo deve fazer uso do processador. Essa função envolve o seguinte:

- Troca de contexto
- Passar para o modo usuário
- Pular para a posição adequada no programa de usuário para reiniciar esse programa

O dispatcher deve ser o mais rápido, levando em conta que ele é chamado durante cada troca de processo. O tempo necessário para o dispatcher interromper um processo e iniciar a execução de outro é chamado de “*Latência de Dispatch*”.

Em sistemas multiprogramados, múltiplos processos são mantidos na memória principal, cada um alternando o uso dos processados. Como fator principal da multiprogramação, três tipos de escalonamento são possíveis:

- **Long-term Scheduling**: Determina os processos que serão admitidos no sistema.
- **Medium-term Scheduling**: Determina a adição de um número de processos que estão parcialmente ou completamente na memória
- **Short-term Scheduling**: Determina quais processos serão executados pelo processador.

Long-term Scheduling e **Medium-term Scheduling** estão diretamente relacionados com aspectos de performance, ou seja, grau de multiprogramação. É utilizado quando o processo deverá ser admitido no sistema e quando tomar decisão de trocar parte do processo da memória primária para a memória secundária.

Short-term Scheduling aborda com alto grau de performance o escalonamento de processos que estão prontos para executar na memória principal.

3.3. Escalonamento FCFS (First-Come, First Served)

Também denominada **FIFO** (**First In, First Out**) é provavelmente a forma mais elementar de escalonamento, consiste em simplesmente atender as tarefas em sequência, à medida em que elas se tornam prontas (conforme a sua ordem de chegada na fila de tarefas). Este algoritmo tem como sua principal vantagem a simplicidade. [3]

Para demonstrarmos um exemplo do funcionamento do **FCFS**, consideremos as tarefas na fila de tarefas prontas, com suas durações previstas de processamento e datas de ingresso no sistema, descritas na seguinte tabela:

Tabela 1: Tarefas FCFS

Tarefa	***t1***	***t2***	***t3***	***t4***
Ingresso	0	0	1	3
Duração	5	2	4	3

O diagrama da figura 2 “Escalonamento FCFS” mostra o escalonamento do processador usando o algoritmo FCFS cooperativo (ou seja, sem *quantum* ou outras interrupções). Os quadros sombreados representam o uso do processador (observe que em cada instante apenas uma tarefa ocupa o processador). [3]

Os quadros brancos representam as tarefas que já ingressaram no sistema e estão aguardando o processador (tarefas prontas). [3]

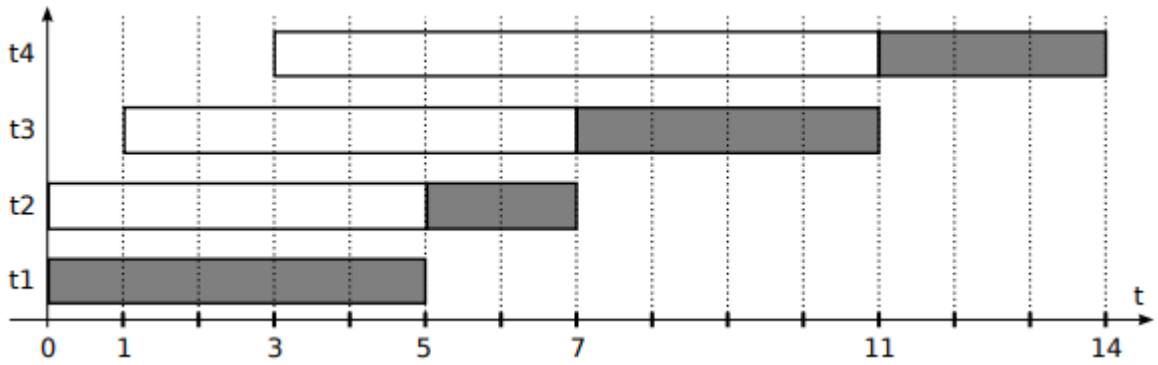

Figura 2:Escalonamento FCFS.

Calculando o tempo médio de execução (T_t, a média de t_t (t_i)) e o tempo médio de espera (T_w, a média de t_w, (t_i)) para o algoritmo FCFS, temos:

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(5-0) + (7-0) + (11-1) + (14-3)}{4}$$
$$= \frac{5+7+10+11}{4} = \frac{33}{4} = 8.25s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(0-0) + (5-0) + (7-1) + (11-3)}{4}$$
$$= \frac{0+5+6+8}{4} = \frac{19}{4} = 4.75s$$

O escalonamento FCFS não leva em conta a importância das tarefas nem seu comportamento em relação aos recursos. Por exemplo, com esse algoritmo as tarefas orientadas a entrada/saída (I/O) irão receber menos tempo de processador que as tarefas orientadas a processamento (pois geralmente não usam integralmente seus *quantums* de tempo) o que pode ser prejudicial para aplicações interativas. [3]

3.4. SJF (Shortest Job First)

A ideia deste algoritmo é passar o controle para o processador que tenha o menor tempo de execução. Tal algoritmo possui algumas características:

- Não-preempção;
- Processos pequenos tem tempo de espera médio menor que os processos grandes;
- Pode causar postergação indefinida nos processos grandes;
- Minimiza o tempo médio de espera de um conjunto de processos, pois os processos menores são executados mais rapidamente.

O escalonamento pode ser implementado usando uma lista ordenada crescente dos tempos de vida dos processos. Os processos são inseridos ordenadamente na lista e no escalonamento basta pegar o primeiro da lista.[1]

Uma desvantagem para implementar esse tipo de escalonamento é prever o futuro. A duração exata de um processo é desconhecida, pois depende, entre outras coisas, dos dados de entrada do programa em execução. [1]

Porém mesmo não sendo possível implementar essa política, ela é útil pois oferece um limite teórico para o tempo médio de espera e a partir dela, pode-se implementar aproximações, usando, por exemplo, cálculos estatísticos sobre dados dos últimos processos executados. Pode ser provado matematicamente que esta estratégia sempre proporciona os menores tempos médios de espera. [1]
O diagrama abaixo (Figura 3: Escalonamento SJF) representa o escalonamento:

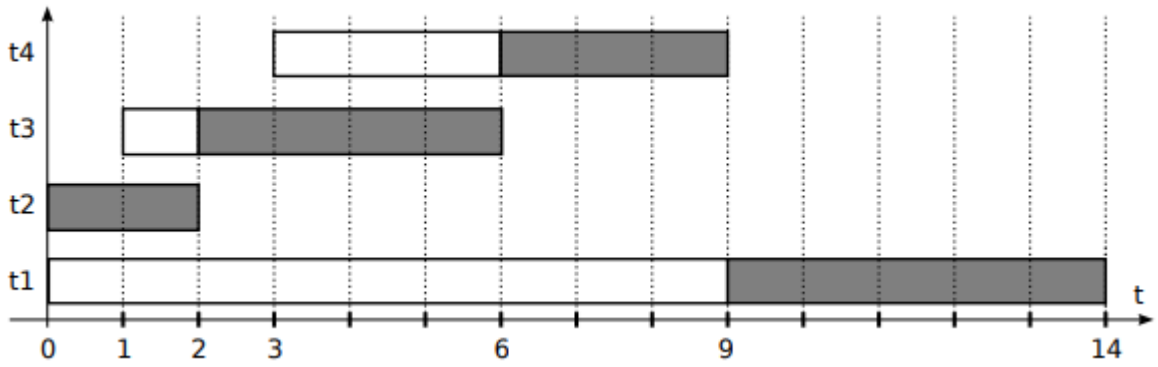

Figura 3: Escalonamento SJF.

Calculando o tempo médio de execução T_t e o tempo médio de espera T_w para o algoritmo SJF, temos que:

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(14-0) + (2-0) + (6-1) + (9-3)}{4}$$
$$= \frac{14+2+5+6}{4} = \frac{27}{4} = 6.75s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(9-0) + (0-0) + (2-1) + (6-3)}{4}$$
$$= \frac{9+0+1+3}{4} = \frac{13}{4} = 3.25s$$

Deve-se observar que o comportamento expresso na figura 2 corresponde à versão cooperativa do algoritmo SJF: o escalonador aguarda a conclusão de cada tarefa para decidir quem irá receber o processador. No caso preemptivo, o escalonador deve comparar a duração prevista de cada nova tarefa que ingressa no sistema com o tempo restante de processamento das demais tarefas presentes, inclusive aquela que está executando no momento.[1]

3.5. Por prioridade

A ideia por trás deste algoritmo é simples: para cada processo, é atribuída uma prioridade e o controle do processador é passado ao processo com maior prioridade. [2]

Este algoritmo pode ser implementado na forma preemptiva ou não-preemptiva. Alguns sistemas utilizam o zero como a maior prioridade enquanto outros utilizam o zero como a menor prioridade. Para evitar a execução indefinidamente de processos com prioridades altas, o **scheduler** pode reduzir a prioridade do processo em execução a cada tique do relógio (clock do processador) [**Tanenbaum, 2003b**]. Assim, considerando preempção neste escalonamento, caso a prioridade do processo ativo seja menor do que a prioridade de um processo na fila de processos, então haverá substituição de processo. [2]

No escalonamento por prioridades, a cada tarefa é associada uma prioridade, geralmente na forma de um número inteiro. Os valores de prioridade são então usados para escolher a próxima tarefa a receber o processador a cada troca de contexto.[2]

Se o processo que sofreu redução em sua prioridade voltar para a fila de processos, sua prioridade volta para o valor original [Oliveira et al., 2001]. Por outro lado, um **quantum** máximo (tempo máximo) pode ser definido para cada processo, correspondendo ao quanto tempo ele pode executar. Quando esgotado esse quantum, o controle do processador é passado para o próximo processo com prioridade mais alta, ou seja, haverá substituição de processos. Em caso de empate as prioridades de processos, o desempate é feito recorrendo ao algoritmo **FCFS (First Come, First Served)**.[2]

O diagrama da Figura 4: Escalonamento por prioridade, mostra o escalonamento do processador usando o algoritmo por prioridades em modo cooperativo (sem *quantum* ou outras interrupções).[2]

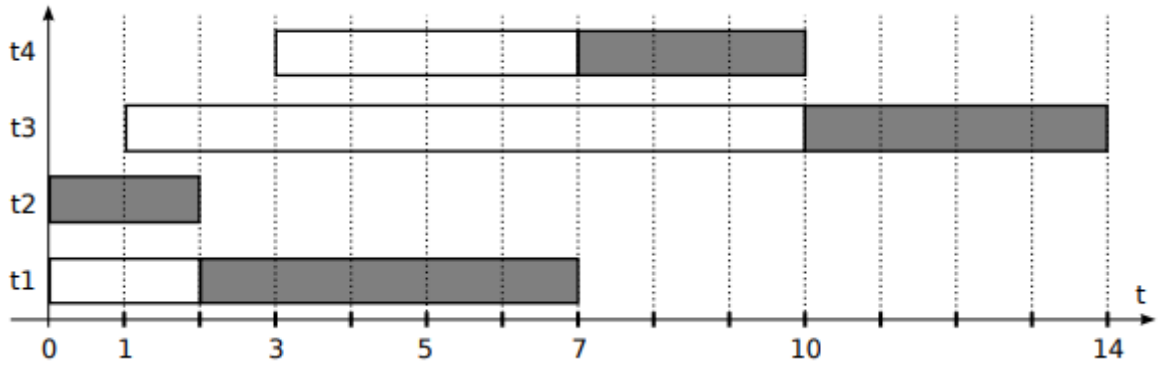

Figura 4: Escalonamento por prioridade (cooperativo)

Quando uma tarefa de maior prioridade se torna disponível para execução, o escalonador pode decidir entregar o processador a ela, trazendo a tarefa atual de volta para a fila de prontas, nesse caso, temos um escalonamento por prioridades preemptivo, cujo comportamento é apresentado na Figura 5: Escalonamento por prioridade (Note que, quando t_4, ingressa no sistema, ela recebe o processador t_1 volta a esperar na fila de prontas).[2]

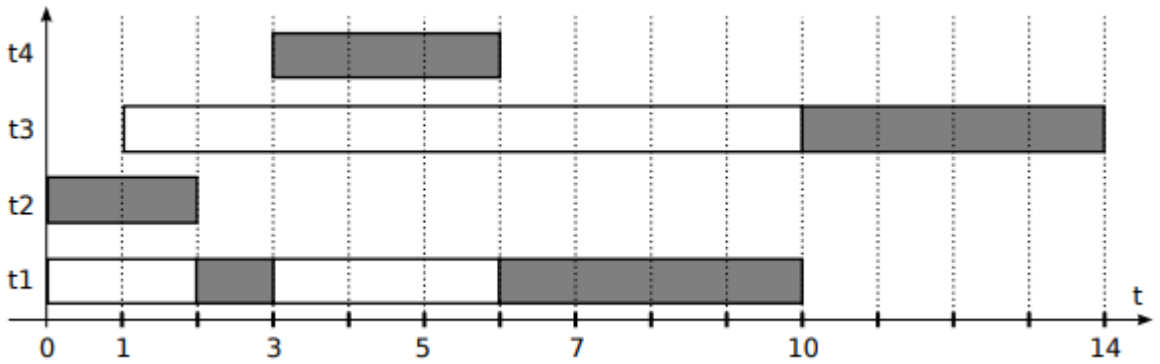

Figura 5: Escalonamento por prioridade (preemptivo)

3.6 Round-Robin

Este algoritmo define uma unidade de tempo denominada ***quantum*** ou ***time slice*** e funciona semelhante ao **FCFS** porém cada processo recebe uma fatia de tempo de processador para ser executado [Arruda, 2008]. Assim, o escalonamento aloca o primeiro processo da fila de processos ao processador durante uma unidade de tempo (*quantum*). Se o processo não terminar a execução após esta unidade de tempo, ocorre a troca de contexto e ele é reinserido no fim da fila de processos. Se ele terminar, o escalonador passa o controle do processador para o próximo da fila.[4]

A Figura 6 a seguir mostra o escalonamento Round-Robin. É importante se atentar que as execuções das tarefas não obedecem uma sequência ordinária, mas uma sequência bem mais complexa. Isso ocorre por causa da ordem das tarefas na fila de tarefas prontas.[4]

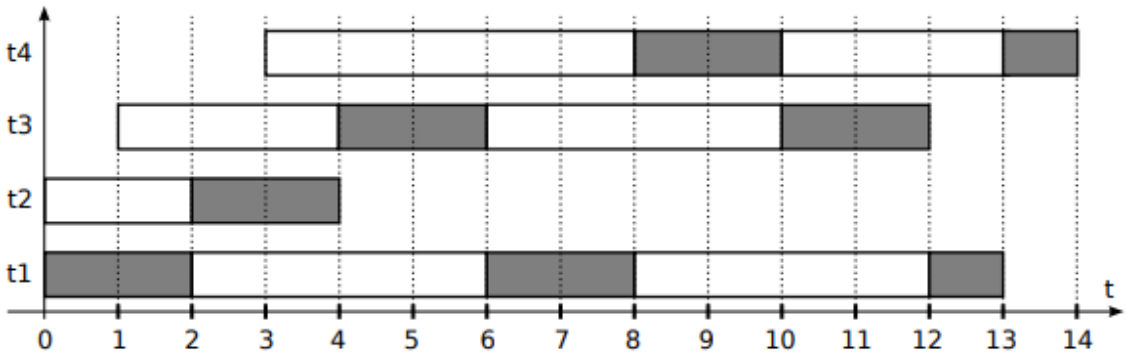

Figura 6: Escalonamento Round-Robin

A escolha de um *quantum* é crítica e deve ser feita com cuidado. Com um *quantum* pequeno, o sistema operacional é forçado a interromper os processos mais frequentes, afetando o desempenho, pois operações fora de contexto não são instantâneas. Porém com *quantum* grande, pode-se perder a aparência do paralelismo na execução dos processos.[4]

3.7 Escalonamento Lotérico

Se trata de um escalonamento por distribuição de cartelas (Fichas) randomicamente, numeradas cada um com seu processo. Cada ficha dá direito ao acesso a um tipo de recurso do computador. Durante o escalonamento cada processo é sorteado para sua vez. Os processos prioritários podem receber mais fichas assim aumentando suas chances de execução. [7][8]

Em suma, cada Thread recebe fichas e os processos mais importantes recebem fichas extras para aumentar a probabilidade de ser executado primeiro.

Além disso:

- Processos cooperativos podem trocar as fichas (Ex: Clientes podem passar cartelas para servidores).
- Caso o mesmo processo seja sorteado duas vezes não há necessidade de troca de contexto.
- Prioridade pode ser implementada conferindo mais fichas ao mesmo processo
- Em teoria, um processo pode nunca ser sorteado. Na prática, as probabilidades garantem que isso não ocorra.
- É um escalonamento responsivo (Sem filas)

3.8 Escalonamento por Fração Justa (Fair-Share)

O Escalonamento por Fração Justa leva em consideração o fato de mais de um usuário estar utilizando a máquina. Cada usuário terá sua fração da CPU, dando uma parcela de tempo. Caso existam dois usuários usando a máquina, os dois terão a mesma quantidade de processamento da CPU, independentemente do número de processos que cada usuário tenha. A fração alocada ao usuário é garantida pelo escalonador conforme a “noção” de justiça sobre a prioridade do usuário (Ex: Se um usuário A possui mais processos que um usuário B e os dois têm a mesma prioridade, os processos de A serão mais demorados que os do B. Ou então, se o A estiver executando 9 processos e o B somente 1, não seria justo o primeiro ter 90% da CPU). [8][9]

Resumindo: O Escalonamento tenta ser o mais justo possível com os usuários, também variando com o número de usuários autenticados para o escalonamento. Faz que um processo só seja escalonado caso ele pertença a um usuário ainda com tempo de CPU para utilizar, e também faz com que os recursos não utilizados sejam distribuídos aos outros usuários [8][9]

4. Conclusão

O desenvolvimento deste trabalho pode nos mostrar uma abordagem funcional de como um escalonamento de processos, analisando os principais tipos de escalonamentos e mostrando seus algoritmos e em que situações eles operam.

Ao mostrar uma abordagem funcional de como é um escalonamento de processos, analisando os principais tipos de escalonamentos e entendendo como funcionam suas regras e situações de cada uma opera.

Passamos a compreender a importância que o escalonamento tem para o sistema, o tornando mais eficiente e rápido, e que, apesar de serem coisas com bases e estruturas básicas para o ensino, consegue atingir uma grande complexidade dependendo da forma que se implementa o algoritmo, variando muito de um para o outro.

5. Referências Bibliográficas

[1] Reis, F.P.; Parreira, P.A.; Xavier, H.A - TBC-SO/WEB - Um software educacional para o ensino de políticas de escalonamento de Processos e de Alocação de Memória em Sistemas Operacionais - acessado em 02/03/2019 às 15:06

[2] Rocha, A.R.; Schneider, A.; Alves, J. C.; Silva, R., M.A. WxProc - Um Simulador de Políticas de Escalonamento Multiplataforma. INFOCOMP - Journal of Computer Science. Vol. III, N. 1: p.43-47, 2004. - acessado em 02/03/2019 às 11:45

[3] Maziero C.A. - Escalonamento FCFS – UFPR - acessado em 02/03/2019 às 09:33

[4] Johann M.O - Operating Systems - acessado em 02/03/2019 às 10:28

[5] Prof. Dr. Márcio Andrey Teixeira - Escalonamento de Processos – IFSP - acessado em 02/03/2019 às 10:15

[6] Celso Maciel da Costa – Sistemas Operacionais Programação Concorrente com Pthreads. – acessado em 03/03/2019 às 15:33

[7]https://medium.com/@TDamiao/o-funcionamento-dos-seguintes-algoritmos-de-escalonamento-5519ea6d0ab4 - acessado em 17/03/2019 às 12:35

[8]http://www.facom.ufu.br/~abdala/so/08_GSI018_6p.pdf - acessado em 17/03/2019 às 12:48

[9]https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Algoritmos_de_escalonamento – acessado em 17/03/2019 às 12:28