Trabalho 2 de Sistemas Operacionais - Escalonador de Processos

Nomes: Luiz Cezar Moreira de Campos Neto e Vinícius Renato Rocha Geraldo

Professor: Gerson Geraldo Homrich Cavalheiro

Pelotas, 2019

1. Introdução

Neste trabalho é realizado um simulador de escalonador de processos implementado na linguagem C++ com tratamentos de cada execução do processo pela quantidade de memória para execução em GB que é recebida, o número de slices, o tempo que o processo é recebido pelo sistema operacional e a sua prioridade.

2. Desenvolvimento

Para o desenvolvimento do simulador utilizamos da estrutura de dados *list* do C++ que contém uma facilidade em fazer tratamentos de algoritmos de fila e assim podendo fazer mudanças push e pop na fila para ir trocando a ordem de execução de cada processo na fila de prioridades. Como entrada do simulador fornecemos um arquivo contendo as seguintes descrições (Vale lembrar que os dados dos processos devem estar separados somente por vírgula):

- Chegada -> Determina o tempo em que o processo foi recebido pelo SO
- Duração -> Número de slices necessários para execução do processo
- Memória -> Quantidade de memória (em GB) necessário à execução do processo
- Prioridade -> Um valor entre 0 e 4, sendo 0 a maior prioridade

Com essa descrição recebemos então esse arquivo de processos para tratamento, onde neste arquivo contém uma lista de processos que é lido no nosso algoritmo e nisso coloca em uma lista de processos que chamamos de disco para fazer os tratamentos. Primeiro verificamos o tamanho de cada lista, de disco e das prioridades, para saber se existem processos que necessitam ser executados ou se já terminaram seu tempo de execução, após isso todos os processos inicializados no disco são separados em 4 listas de ordem de prioridades colocando sempre no final da fila o processo que chegar com a prioridade relacionada utilizando do algoritmo de escalonamento de Round Robin, .

Após feito todo esse tratamento são realizadas as operações necessárias para execução de cada processo na fila. Cada processo da fila chega com uma memória para ser executada e com a quantidade de slices necessárias. Primeiramente o disco é percorrido para buscar processos que podem ser executados, esses mesmos processos são colocados nas suas filas de prioridades correspondentes. Depois basicamente é testada se a memória necessária para executar o processo é menor ou igual a memória da máquina, caso a memória do processo seja maior que a memória disponível na máquina, o processo é descartado, tendo em vista que a sua memória extrapola a memória da máquina, fazendo com que tal processo jamais seja executado. Logo após, é realizado um segundo teste que verifica se o contador de memória a ser executado no ciclo atual

é maior do que a memória disponível, caso seja, o processo em questão é colocado no final da fila e o próximo é testado e assim por diante. Caso o processo selecionado seja no qual já havia sido selecionado anteriormente para ir para o processador, esse processo permanece no topo da fila mas não vai ao processador. O processador executa uma slice do processo e após essa execução o processo é colocado no final da sua respectiva fila, e após dez execuções os processos de prioridades originais 1, 2 e 3 tem sua prioridade trocada.

Com base nesse escopo de trabalho analisamos a duração projetada com a duração observada do processo tratado pelo CPU. Notamos que quanto menor a quantidade de CPUs que colocamos como de entrada no escalonador notamos que a duração observada da execução pode ser até 4 vezes maior que a duração projetada. Quando aumentamos a quantidade de núcleos de CPUs percebemos um grande impacto no desempenho do escalonador, onde assim analisamos que uma maior quantidade de núcleos podemos utilizar mais processos executando em cada CPU e sendo assim aproximando ao valor de saída da duração projetada. Já em relação a chegada e lançamento do processo está diretamente ligado a prioridade de execução do processo podendo assim mostrar o atraso que o processo teve em relação a chegada ao SO.

3. Resultados e Discussões

Como resultados temos de saída do nosso escalonador as seguintes informações:

- Chegada -> Representa o tempo em que o processo foi recebido pelo SO
- Lançamento - > Tempo em que o processo foi lançado efetivamente (se igual à chegada, não houve atrasos)
- Duração projetada -> É a duração informada no arquivo de entrada para o processo
- Duração observada -> Contabiliza o tempo necessário para execução do processo

Com esse escopo de resultado abaixo temos algumas execuções do nosso programa para uma seguinte entrada de processos:

Processos do arquivo

- 0, 12, 128, 1
- 0, 14, 256, 1
- 4, 8, 256, 2
- 5, 20, 64, 3
- 8, 15, 128, 0
- 12, 23, 64, 0
- 20, 20, 256, 4
- 6, 12, 128, 4
- 27, 32, 64, 2

Número de CPUs = 1 / Quantidade de Memória = 320GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	65
0	1	14	72
4	58	8	75
5	82	20	136
6	102	12	141
8	8	15	25
12	13	23	34
20	103	20	136
27	59	32	117

Tabela 1. 1 CPU e 320GB de memória

Número de CPUs = 1 / Quantidade de Memória = 640GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	65
0	1	14	72
4	58	8	75
5	82	20	136
6	102	12	141
8	8	15	25
12	13	23	34

20	103	20	136
27	59	32	117

Tabela 2. 1 CPU e 640GB de memória

Número de CPUs = 2 / Quantidade de Memória = 320GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	23	14	39
4	33	8	42
5	5	20	68
6	52	12	69
8	8	15	15
12	12	23	23
20	53	20	65
27	34	32	49

Tabela 3. 2 CPUs e 320GB de memória

Número de CPUs = 2 / Quantidade de Memória = 640GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	28
0	0	14	35
4	23	8	35
5	39	20	65
6	49	12	67
8	8	15	15
12	12	23	23
20	50	20	62

27	30	32	47

Tabela 4. 2 CPUs e 640GB de memória

Número de CPUs = 3 / Quantidade de Memória = 320GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	23	14	37
4	4	8	13
5	18	20	42
6	23	12	40
8	8	15	15
12	12	23	23
20	24	20	34
27	27	32	32

Tabela 5. 3 CPUs e 320GB de memória

Número de CPUs = 3 / Quantidade de Memória = 640GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	13
0	0	14	18
4	4	8	13
5	18	20	42
6	23	12	40
8	8	15	15
12	12	23	23
20	24	20	34
27	27	32	32

Tabela 6. 3 CPUs e 640GB de memória

Número de CPUs = 4 / Quantidade de Memória = 640GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	0	14	14
4	4	8	14
5	8	20	25
6	18	12	26
8	8	15	15
12	12	23	23
20	21	20	22
27	27	32	32

Tabela 7. 4 CPUs e 640GB de memória

Número de CPUs = 4 / Quantidade de Memória = 960MB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	0	14	14
4	4	8	8
5	5	20	25
6	14	12	21
8	8	15	15
12	12	23	23
20	21	20	21
27	27	32	32

Tabela 8. 4 CPUs e 960GB de memória

Número de CPUs = 5 / Quantidade de Memória = 1024GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	0	14	14
4	4	8	8
5	5	20	20
6	6	12	16
8	8	15	15
12	12	23	23
20	20	20	20
27	27	32	32

Tabela 9. 5 CPUs e 1024GB de memória

Número de CPUs = 6 / Quantidade de Memória = 1024GB			
Chegada	Lançamento	Duração Projetada	Duração Observada
0	0	12	12
0	0	14	14
4	4	8	8
5	5	20	20
6	6	12	12
8	8	15	15
12	12	23	23
20	20	20	20
27	27	32	32

Tabela 10. 6 CPUs e 1024GB de memória

Analisando os resultados obtidos de cada tabela vemos que a quantidade de CPUs impacta diretamente na ordem de que cada CPU pode tratar um processo levando

em conta cada prioridade associada. Vemos nas tabelas de execuções que com apenas 1 CPU os tratamentos dos processos acabam sendo mais lentos que o esperado pois apenas contém um núcleo executando todas as chegadas e lançamentos do processo e conforme vamos aumentando as CPUs os processos conseguem ter uma demanda melhor de processamento. A partir do momento que temos mais CPUs precisamos também de memória suficiente para tratar vários processos em paralelo, por isso que normalmente os processos que chegam primeiro com uma alta prioridade e ocupam grande parte da memória sempre vão terminar antes, pois além de ter uma alta prioridade, esses processo ainda usam toda a memória. Outra situação é ilustrada na Tabela 3, onde temos 2 CPUs e dois processos que chegam ao mesmo tempo, mas um termina mais cedo do que o outro, isso se deve por conta de que não tem memória disponível para os dois executarem ao mesmo tempo, e como após o teste de memória o processo que não pôde ser executado vai para o final da fila, sempre o mesmo processo vai ser executado ciclo após ciclo (até que chegue um processo de prioridade mais alta). Podemos notar que apenas com 6 CPUs processando em paralelo que chegamos ao resultado ótimo, onde a duração observada é igual a duração projetada, situação ilustrada pela Tabela 10.

Outra situação interessante é ilustrada pela Tabela 4, onde de forma diferente do que ocorre na situação da Tabela 3, mencionada anteriormente, os dois processos que chegam no momento 0 podem ser executados de maneira paralela até o momento 8 onde chega um processo de prioridade 0, sendo assim, os dois processos que chegaram no momento 0 ficam se alternando em uma CPU enquanto isso a outra CPU fica exclusivamente para o processo de prioridade 0, até o momento 12, onde chega mais um processo de prioridade 0, sendo assim, as duas CPUs ficam para os processo de chegada 8 e 12, enquanto os processos de chegada 0 devem ficar esperando, por isso a duração projetada é diferente da duração observada indicada pela Tabela 3.

Pode-se concluir que em algumas ocasiões não basta aumentar o número de CPUs isoladamente, e o mesmo vale para a memória, o melhor resultado obtido deve possui um bom balanceamento entre os dois parâmetros, como foi visto nas Tabelas 9 e 10, na Tabela 9 ainda não se encontra o resultado ótimo, mas como foi visto na Tabela 10 não existe uma variação na memória que pudesse ocasionar um melhor resultado utilizando cinco CPUs, por conta de que, utilizando seis CPUs e com o mesmo valor de memória, o resultado ótimo foi obtido.