EP1: Escalonadores de Processos

Anderson Andrei da Silva

MAC0442 Sistemas Operacionais

1. Introdução

A tarefa neste EP éimplementar um shell para permitir a interacão do usuário com o sistema operacional, e um simulador de processos com diversos algoritmos de escalonamento para esses processos.

2. O programa

O programa consiste em dois principais executáveis *ep1sh* e *ep1*. O *shell*, chamado de *ep1sh*, permite a invocação externa (execuçao) dos 3 binários abaixo com exatamente os argumentos abaixo :

- /bin/ping -c 10 www.google.com.br
- /usr/bin/cal 2017
- ./ep1 <scheduler> <trace> <output> [d] , onde:
  - <scheduler> é um inteiro que varia de 1 a 3 sendo respectivamente SJF, RoudRobin e Escalonamento com Prioridade;
  - <trace> é o nome do arquivo de trace a ser dado como entrada
  - <output> é o nome do arquivo de saída, onde serão gravados os resultados

  d é uma flag opcional, que habilita a exibição de mensagens que possibilitam acompanhar o adamento do programa

Note que : O segundo executável principal *ep1*, está listado à cima, no terceiro item.

O shell também tem os 2 comandos abaixo embutidos (internos) nele, que são implementados usando chamadas de sistema do Linux que não são da família de chamadas exec* ou similares. Esses comandos são executados sempre com os argumentos abaixo, que fazem exatamente o que esses 2 comandos fazem no shell bash:

- chown :<grupo> <arquivo no diretório atual>
- date

3. Entrada e saída de dados

O simulador de processos recebe como entrada um arquivo de trace, em texto puro, que possui várias linhas como a seguinte:

t0 dt deadline nome

- t0 : é o instante de tempo em segundos quando o processo chega no sistema
- dt : é o quanto de tempo real da CPU deve ser simulado para aquele processo
- deadline : é o instante de tempo antes do qual aquele processo precisa terminar
- nome : é uma string sem espaços em branco que identifica o processo.

Os parâmetros t0, dt e deadline são números de ponto flutuante representados apenas com dıgitos sem casas decimais, ou com no máximo 1 casa decimal separada por um ponto. Sempre havendo pelo menos 1 dígito antes do ponto. Cada linha do arquivo de entrada representa portanto um processo, que é simulado no simulador, como uma única thread. Cada thread é um loop que realiza operações que consomem tempo real. O simulador finaliza sua execução assim que todos os processos terminam de ser simulados.

Um arquivo será criado pelo simulador com 1 linha para cada processo e mais 1 linha extra no final. Cada linha por processo tem o seguinte formato:

nome tf tr

- nome : é o identificador do processo,
- tf : é o instante de tempo quando o processo terminou sua execução
- tr : é o tempo "de relógio" que o processo levou para executar, ou seja, tf-t0.

A linha extra contém um único número que informa a quantidade de mudanças de contexto que ocorreram durante a simulação. Ou seja, a quantidade de vezes que as CPUs deixaram de rodar um processo para rodar outro.

4. Algoritmos

Como citado no início, foram implementados nesse trabalho os algoritmos SJF, Roud Robin e Escalonamento por prioridade, em duas etapas principalmente. A primeira "genérica"sendo a mesma para todas, e conforme as especificidades de cada algoritmo temos as "próprias". Para trabalhar com os dados foram utilizadas as estruturas de dados pilha e vetores. Considerando o recebimento de dados, tratamento e identificação do que nosso shell irá executar e que então foi verificado e os dados direcionados para nosso *ep1*, seguimos primeiro de forma genérica :

- Todos os processos são enfileirados, ordenados pelo parametro t0, criando assim um *pool* de processos.
- Tendo o controle do tempo decorrido na execução do programa :
  - Os dados são retirados do nosso pool;
  - E se o instante que ele deveria entrar na fila de execução está ocorrendo ou já passou, ele é inserido na fila de execução;
  - O próximo processo é retirado do nosso *pool* e aguarda o seus instante de entrada. Se vários processos validam a condição anterior, todos eles são inseridos na fila de execução.

Observação: a forma de ordenação para inserção na fila de execução varia conforme o algoritmo e será especificado a seguir (*).

4.1. SJF, Shortest Job First

Como o próprio nome já diz, os processos com tempo de duração menor tem mais prioridade, assim sendo, para inserí-los na fila de execução (*) ordenaremos os processos pelo campo *dt*, assim os menores serão executados primeiro. Após a inserção dos processos na fila e enquanto ela não for vazia:

- Um processo por vez é retirado
- O mutex responsável pelo processo recebe lock
- A thread é criada e executa seu job (igual para todas)
- É utlizada a função join para aguardar que a thread retorne
- Se solicitado, as informações momentâneas são escritas na *stderr*
- As informações necessárias são escritas no arquivo de saída

- O mutex recebe lock
- Os contadores são atualizados
- É verificado junto ao temp de execução se mais processos vão entrar na fila de execução

4.2. Roud Robin

Para esse algoritmo definimos um *quantum* que é o tempo no qual todos os processos irão rodar por vez. Assim todos serão alternados conforme o tempo for passando. O processo é parecido com o de cima, com exceção do quantum, assim um processo só é executado por aquele período de tempo. Ao termino do tempo, é verificado se todo seu *dt* foi percorrido, se sim, o processo é finalizado, se não, o processo volta pra fila, de forma que outros possa executar o mesmo "quantum"de tempo que ele, até que ele seja executado novamente.

4.3. Escalonamento com prioridade

Já este algoritmo é identico ao Roud Robin, com a única diferença de que o quantum é específico para cada processo, de acordo com sua prioridade. Mas, da mesma forma que o anterior, se ao término de seu quantum, o processo não tiver terminado, ele volta para a fila e aguarda até ser executado novamente.

5. Análise dos Resultados

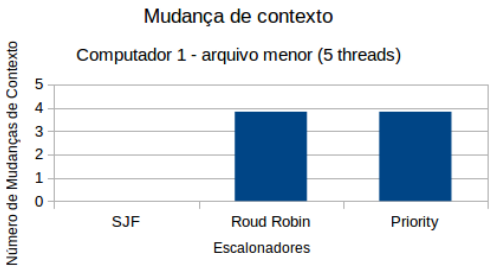
Foram feitas simulações na seguint estrutura:

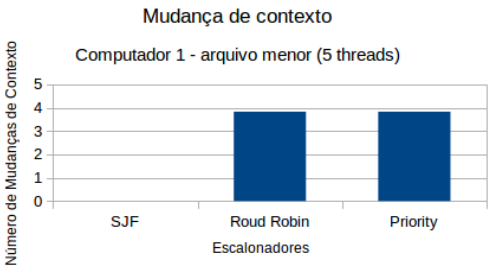
- 3 tipos de arquivos de trace: pequeno (5 threads), médios (10 threads) e grandes (15 threads).
- Cada tipo de arquivo foi gerado aleatóriamente 30 vezes
- Os dados de t0, dt e deadline variama dentro no intervalo real de 0 à 5 segundos
- Foram realizados simulações para analisar o **cumprimento de deadlines** e **mudanças de contexto**. Para tal, foram utilizados todos os arquivos de entrada para os dois casos.
- E ainda foram feitos as mesmas simulações em 2 computadores diferentes: (um Intel Core i7, 6GB e um Intel Core i3, 4GB).
- Obtivemos no total então 270 arquivos de saída e gerados 12 gráficos, 6 para cada computador, sendo 3 para cada um dos dois propósitos de análise, divididos pelo tamanho dos arquivos.

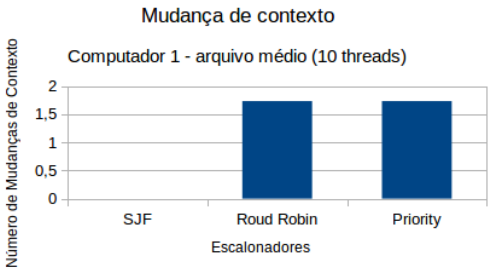
5.1. Resultados

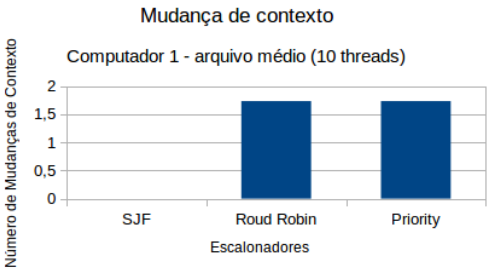
Decisões de projeto: Para fins de simulação foram definidos jobs que seriam executados em 1s, assim como quantums de 1s. E para o escalonador por prioridade, foi utilizada a função f(t0) = t0 + t0 * 0.1 para calcular o quantum de cada processo.
Observações: Como o intervalo dos dados, assim como o quantum foram pequenos, os gráficos dos dois computadores são identicos, pois não se foi usado grande processamento para que diferissem. Serão mostradas então todas as imagens apenas para fins de documentação.

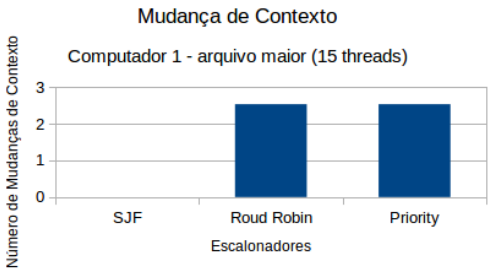
Mudança de contexto

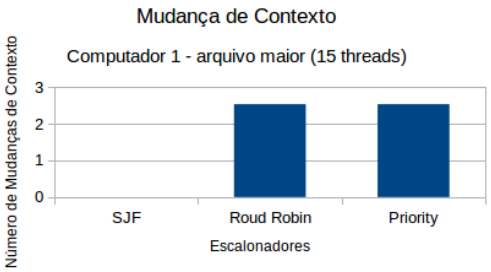








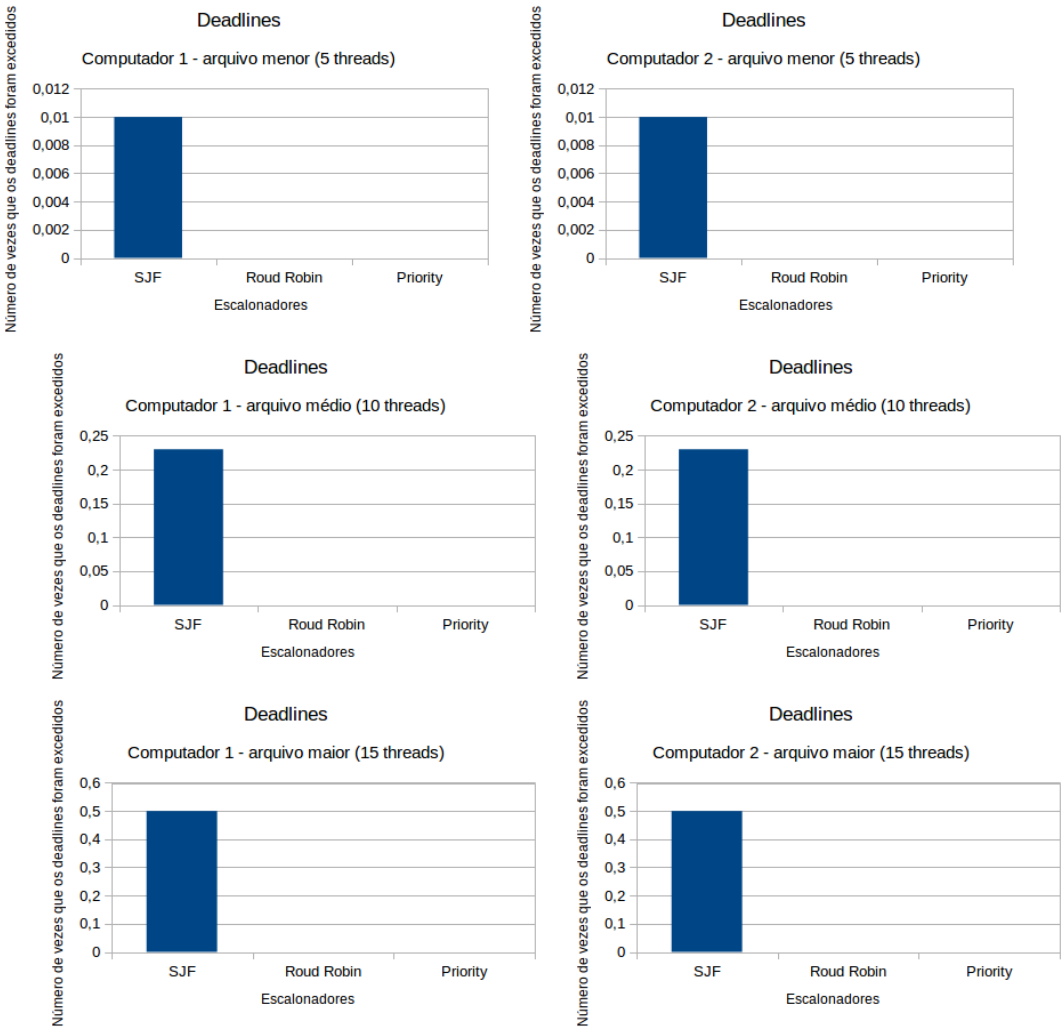




Então, apesar de não podermos concluir diferenças entre os computadores, podemos concluir que existem mais mudança de contexto em arquivos menores, ou seja, os escalonadores RoudRobin e Escalonador por Prioridade

são mais eficientes e melhor aproveitados quando se trata de uma maior quantidade de processos, já que mudanças de contexto são custosas e além de que o SJF não faz nenhuma.

Cumprimento de Deadlines



Aqui podemos concluir que o escalonador SJF não cumpre tão perfeitamente os deadlines dos processos, já que executa todo o processo até o fim, de uma vez. Assim como podemos concluir que esse número aumenta conforme o número de processos logo, o SJF não é tão eficiente para esses casos. Visualmente os escalonadores Roud Robin e o Escalonador com Prioridade o fazem melhor, apontado no trabalho pelo controle melhor do tempo, através dos quantums.