MAC0422 – Sistemas Operacionais – 2s2020
EP1
Data de entrega: 5/10/2020 até 8:00 da manhã

Prof. Daniel Macêdo Batista

1 Problema

A tarefa neste EP é implementar um shell para permitir a interação do usuário com o sistema operacional, e um simulador de processos com diversos algoritmos de escalonamento para esses processos. Todos os códigos devem ser escritos em C para serem executados no GNU/Linux.

1.1 O shell `bccsh`

O shell, chamado de `bccsh`, a ser desenvolvido, deve ser capaz de permitir a invocação externa (execução) dos 3 binários abaixo com exatamente os argumentos abaixo. Não há necessidade de testar o shell para outros programas e nem para os binários abaixo com outros argumentos:

- `/usr/bin/du -hs .`
- `/usr/bin/traceroute www.google.com.br`
- `./ep1 <argumentos do EP1>`

O shell também precisa ter os 3 comandos abaixo embutidos (internos) nele, que devem obrigatoriamente ser implementados usando chamadas de sistema do Linux que não sejam da família de chamadas `exec*` ou similares. Esses comandos devem ser executados sempre com os argumentos abaixo, que devem fazer exatamente o que esses 3 comandos fazem no shell `bash`:

- `mkdir <diretorio>`
- `kill -9 <PID>`
- `ln -s <arquivo> <link>`

Não se preocupe em tratar os erros dos 3 comandos acima. O usuário nunca vai colocar um nome de diretório que já existe, um número de processo que não existe ou um nome de arquivo que não existe.

O shell tem que suportar a listagem de comandos que foram executados previamente com o uso das teclas “para cima” e “para baixo”, bem como a edição desses comandos para serem executados, por meio das funcionalidades das bibliotecas GNU readline e GNU history. No Debian ambas fazem parte do pacote `libreadline-dev`. Mais informações podem ser vistas na documentação da biblioteca em

`ftp://ftp.gnu.org/pub/gnu/readline/` . Não há necessidade de utilizar outras funcionalidades das bibliotecas além das requisitadas no início deste parágrafo.

O prompt do shell deve conter o nome do usuário, seguido de '`@`' e do diretório atual entre chaves e seguido de um espaço em branco, como no exemplo abaixo que mostra o shell pronto para rodar o comando `kill`:

```
{daniel@/tmp/mac0422/} kill -9 12345
```

1.2 Simulador de processos

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

```
nome t0 dt deadline
```

`nome` é uma string sem espaços em branco de no máximo 30 caracteres que identifica o processo, `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 e `deadline` é o instante de tempo antes do qual aquele processo precisa terminar. `t0`, `dt` e `deadline` são números naturais.

Cada linha do arquivo de entrada representa portanto um processo, que deverá ser simulado no simulador, a ser implementado, como uma única thread. Cada thread precisa ser um loop que realize qualquer operação que consuma tempo real. Não há uma predefinição de qual deve ser essa operação.

Assim, se o simulador receber como entrada um arquivo que contenha apenas a linha:

```
processo0 1 10 11
```

é de se esperar, num cenário ideal, que no instante de tempo 1 segundo uma thread seja criada para representar o `processo0` e que no instante de tempo 11 segundos o `processo0` termine de executar.

O simulador deve finalizar sua execução assim que todos os processos terminarem de ser simulados.

O simulador será mais interessante de ser executado com arquivos de trace que permitam mais de um processo ao mesmo tempo competindo pela CPU (ou pelas CPUs em casos onde o computador tenha mais de 1 unidade de processamento). Nessas situações o escalonador de processos implementado no simulador terá um papel fundamental e provavelmente levará a diferentes resultados.

Diversos escalonadores de processos existem. Neste EP o simulador deve implementar os seguintes escalonadores:

1. First-Come First-Served
2. Shortest Remaining Time Next
3. Round-Robin

A invocação do simulador no `bccsh` deve receber como primeiro parâmetro obrigatório o número representando cada escalonador, conforme a listagem acima, como segundo parâmetro obrigatório o nome do arquivo de trace e como terceiro parâmetro obrigatório o nome de um arquivo que será criado pelo simulador com 1 linha para cada processo e mais 1 linha extra no final. Cada linha por processo deverá ter o seguinte formato:

```
nome tf tr
```

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

A linha extra deve conter um único número que informará 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.

O simulador deve receber ainda como quarto parâmetro opcional, o caractere `d`. Quando esse parâmetro for usado, o simulador deverá exibir os seguintes eventos, assim que eles acontecerem, na saída de erro (`stderr`):

- chegada de um processo no sistema, informando o conteúdo da linha daquele processo no trace
- uso da CPU por um processo, informando qual o processo que começou a usar a CPU e qual CPU ele está usando
- liberação da CPU por um processo, informando qual o processo que deixou de usar a CPU e qual CPU ele está liberando
- finalização da execução do processo, informando a linha que será escrita no arquivo de saída
- quantidade de mudanças de contexto

O formato de exibição dessas informações é livre.

2 Requisitos

A compilação do código deve gerar dois binários. Um binário do `bccsh` e um binário do simulador de processos (`ep1`).

Todo o código deve ser escrito em C e toda a gerência de threads deve ser feita utilizando POSIX threads (pthreads). Programas escritos em outra linguagem ou utilizando bibliotecas extra para gerenciar as threads, fazer a simulação de processos ou fazer o escalonamento terão nota ZERO.

Informações sobre como programar utilizando pthreads podem ser encontradas na página da wikipedia em `http://en.wikipedia.org/wiki/POSIX_Threads` ou no tutorial da IBM disponível em `http://www.ibm.com/developerworks/library/l-posix1/index.html`.

Não há necessidade de empacotar as bibliotecas GNU readline e GNU history pois elas já estarão instaladas na máquina que será usada na correção do EP1.

3 Sobre a entrega

Deve ser entregue um arquivo .tar.gz contendo os itens listados abaixo. EPs que não contenham **todos** os itens abaixo **exatamente como pedido** terão nota ZERO e não serão corrigidos. **A depender da qualidade do conteúdo entregue**, mesmo que o EP seja entregue, **ele pode ser considerado como não entregue, o que mudará o cálculo da média final**:

- código-fonte em C;
- arquivo LEIAME **em formato texto puro** explicando como compilar e executar os dois binários;
- Makefile ou similar para facilitar a compilação do código-fonte e a geração dos dois binários;

- apresentação **em .pdf** para ser apresentada em no máximo 15 minutos resumindo os resultados obtidos com diversos experimentos, explicando a arquitetura do shell e resumindo como foi feita a implementação dos escalonadores (se há uma camada comum a todos eles, como foi feito para interromper as threads e reinicializá-las, entre outras informações que forem julgadas como importantes, principalmente relacionadas com decisões de projeto que não ficaram especificadas no enunciado). **Esses slides não serão apresentados. Eles devem ser preparados supondo que você teria que apresentá-los.**

Os resultados devem ser exibidos com gráficos que facilitem observar qual foi o impacto dos diferentes escalonadores no cumprimento dos *deadlines* dos processos simulados e na quantidade de mudanças de contexto. Essas 2 métricas devem ser exibidas para todos os escalonadores com traces de 3 tipos: poucos processos, muitos processos, e um valor intermediário entre os poucos e os muitos. Além disso, deve-se realizar os experimentos em duas máquinas com diferentes quantidades de processadores. Cada valor a ser apresentado nos gráficos deve possuir média e intervalo de confiança de 30 medições com nível de confiança de 95%. Recomenda-se que sejam gerados gráficos em barra, onde cada barra represente o resultado de cada 1 dos 3 escalonadores. Assim, haverá um total de 12 gráficos (3 gráficos para cumprimento de *deadline* – 1 para cada quantidade de processos – na máquina A, 3 gráficos para quantidade de mudanças de contexto na máquina A, 3 gráficos para cumprimento de *deadline* na máquina B e 3 gráficos para quantidade de mudanças de contexto na máquina B). Além de apresentar os gráficos, os slides devem ter uma breve análise crítica dos resultados justificando se eles foram os esperados ou não. Se não foram, os motivos devem ser apresentados. A apresentação em .pdf vale 3,0 pontos.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep1-membros_da_equipe. Por exemplo: ep1-joao-maria. EPs que não gerem um diretório ou que gerem o diretório com o nome errado perderão 1,0 ponto.

A entrega do .tar.gz deve ser feita no e-Disciplinas.

O EP deve ser feito em dupla.

Obs.: não inclua no .tar.gz itens que não foram pedidos neste enunciado. Relatórios, saídas para diversas execuções e entradas usadas para testar o programa não devem ser entregues. No máximo um resumo sobre as entradas usadas pode ser colocado na apresentação, caso isso não ocupe muito espaço.