MAC0422 – Sistemas Operacionais – 1s2025
EP1 (Individual)
Data de entrega: 22/4/2025 até 13:00:00

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 três algoritmos de escalonamento para esses processos. Todos os códigos devem ser escritos em C para serem executados no GNU/Linux.

1.1 O shell `uspsh`

O shell, chamado de `uspsh`, a ser desenvolvido, deve ser capaz de permitir a invocação externa (execução) dos 3 executáveis abaixo com exatamente os argumentos abaixo. Não há necessidade de testar o shell para outros programas e nem para os programas abaixo com outros argumentos pois a correção será feita exatamente como informado abaixo:

- `/bin/ls -laF --color=never`
- `/bin/top -b -n 1 -p 1`
- `./ep1 <argumentos do EP1>`

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

- `cd <diretório>`
- `whoami`
- `chmod <permissão> <arquivo|diretório>`

Não se preocupe em tratar os erros dos 3 comandos acima. O usuário nunca vai colocar um nome de arquivo ou diretório que não existe ou uma permissão incorreta (considere que a permissão vai ser sempre na base octal com exatamente três dígitos e.g.: 644, 755, 012, etc...). Para descobrir qual chamada de sistema deve ser usada na implementação de cada comando acima, olhe a manpage `syscalls` e depois a manpage da chamada de sistema específica. Na maioria das vezes, o nome da chamada de sistema

a ser usada terá similaridade com o nome do comando que deve ser implementado ou com o que esse comando faz, em inglês. Use essa informação como dica para descobrir qual chamada você precisará usar. As chamadas de sistema aceitas precisam estar definidas na manpage `syscalls`. Note que nem sempre a chamada de sistema vai retornar a saída da forma que você precisa para exibir no shell, o que pode exigir a utilização de outras funções. Essas prováveis funções para tratar as saídas de cada chamada de sistema não precisam ser chamadas de sistema.

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 presente no fonte disponível em `https://ftp.gnu.org/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 computador seguido de dois pontos e do diretório atual, tudo entre colchetes, seguido de $ e de um espaço em branco, como no exemplo abaixo que mostra o shell pronto para rodar o comando `ls`:

```
[eclipse:/tmp]$ /bin/ls -1aF --color=never
```

Tanto o nome do computador quanto o diretório atual precisam ser capturados pelo shell, para serem exibidos no prompt, usando chamadas de sistema seguindo as mesmas restrições explicadas acima referentes aos comandos `cd`, `whoami` e `chmod`, ou seja, precisam estar definidas na manpage `syscalls` e não podem fazer parte da família de chamadas `exec*` ou similares. Cada funcionalidade precisa usar uma chamada de sistema específica, ou seja, será necessário usar duas para o prompt.

1.2 Simulador de processos

O simulador de processos deve receber como entrada um arquivo de trace[1], 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 32 caracteres ASCII 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 dos núcleos de processamento deve ser destinado para aquele processo simulado e `deadline` é o instante de tempo até o qual o usuário gostaria que aquele processo terminasse. `deadline`, `t0` e `dt` são números naturais e `deadline` $\geq$ `t0`+`dt`.

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 2 3
```

é de se esperar, num cenário ideal, que no instante de tempo 1 segundo uma thread seja criada para representar o `processo0`, que ela inicie sua execução, e que no instante de tempo 3 segundos ela

[1] Esse termo (trace) costuma ser usado em referência a entradas e saídas de simuladores. Mesmo em textos em português é comum usar o termo trace em inglês, sem tradução

termine de executar. Nesse caso o processo terá finalizado dentro do seu tempo limite de 3 segundos, deixando o usuário feliz :-).

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 pelo núcleo de processamento (ou pelos núcleos de processamento em casos onde o computador tenha mais de 1 núcleo). Nessas situações o escalonador de processos implementado no simulador terá um papel fundamental e provavelmente levará a diferentes resultados. Você terá que pesquisar quais funções precisará utilizar para garantir que os núcleos sejam usados corretamente por cada processo simulado. Pesquise as funções da família `pthread_`, tanto as vistas em sala de aula como outras. A depender dos resultados preliminares, considere utilizar as funções que possuem `affinity` no nome e que têm por objetivo fazer uma thread ser executada em um núcleo específico. Vale observar que o simulador será corrigido em um computador com mais de 1 núcleo de processamento, portanto, teste o seu código em algum computador com essa característica.

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

1. First-Come First-Served (FCFS)
2. Shortest Remaining Time Next (SRTN)
3. Escalonamento com prioridade (usando o `deadline`, o instante de tempo atual e o objetivo de cumprir o maior número de *deadlines* possíveis como critérios para definir a quantidade de quantums dada a cada processo)

A invocação do simulador no `uspsh` 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 tr tf cumpriu
```

Onde `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` e `cumpriu` vale 1 se $\texttt{tf} \leq \texttt{deadline}$ e 0 caso contrário.

A linha extra deve conter um único número que informará a quantidade de preempções que ocorreram durante a simulação. Ou seja, a quantidade de vezes que os núcleos de processamento foram forçados a deixar de rodar um processo para rodar outro, antes do anterior terminar.

2 Requisitos

A compilação do código deve gerar dois executáveis. Um executável do `uspsh` e um executável 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). EPs escritos em outra linguagem ou utilizando bibliotecas extra para implementar o shell, 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 `https://www.ibm.com/docs/en/i/7.5?topic=category-pthread-apis`.

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 em termos de formato e de nomes** 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**:

- 1 único arquivo `uspsh.c` e 1 único arquivo `uspsh.h` com a implementação do shell;
- 1 único arquivo `ep1.c` e 1 único arquivo `ep1.h` com a implementação do simulador de processos;
- 1 único arquivo `LEIAME` em formato texto puro explicando como compilar e executar os dois programas;
- 1 único arquivo `Makefile` para gerar os dois executáveis;
- 1 único arquivo `slides-ep1.pdf` em formato PDF com no máximo 12 slides (contando todos os slides, inclusive capa, roteiro e referências, se houverem), justificando se o código do simulador de processos é determinístico ou não para as entradas testadas, explicando o algoritmo do escalonamento com prioridade que foi projetado e resumindo os resultados obtidos com experimentos que mostrem que os 3 algoritmos de escalonamento se comportaram como esperado ou não a depender da entrada usada. **Esses slides não serão apresentados. Eles devem ser preparados supondo que você teria que apresentá-los**. Não coloque nos slides conteúdo que não foi pedido. Por exemplo, não precisa repetir o enunciado, não precisa explicar nada sobre a implementação do shell e nem explicar os algoritmos FCFS ou SRTN;
- 2 arquivos de entrada `entrada-esperado.txt` e `entrada-inesperado.txt` usados para os experimentos relatados na apresentação. Cada arquivo precisa ter no máximo 50 processos e a soma dos tempos de execução (`dt`) de todos os processos não pode passar de 120 segundos.

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 preempções. Essas 2 métricas devem ser exibidas para todos os escalonadores com traces de 2 tipos: com resultados esperados (em que o escalonamento com prioridade tenha resultado melhor que o SRTN, que por sua vez seja melhor que o FCFS) e com resultados inesperados (em que o escalonamento FCFS tenha resultado melhor ou igual que o SRTN, que por sua vez seja melhor ou igual que o escalonamento com prioridade). Além disso, deve-se realizar os experimentos em duas máquinas com diferentes quantidades de unidades de processamento. Caso você conclua que seu código não é determinístico para os traces utilizados, cada valor a ser apresentado nos gráficos deverá 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 um dos 3 escalonadores. Assim, haverá um total de 8 gráficos com 3 barras

cada: 2 gráficos para cumprimento de *deadline* (1 para cada arquivo) na máquina A, 2 gráficos para quantidade de preempções na máquina A, 2 gráficos para cumprimento de *deadline* na máquina B e 2 gráficos para quantidade de preempções na máquina B. Além de apresentar os gráficos, os slides devem ter uma breve análise crítica dos resultados com base em como os arquivos de entrada foram projetados. Note que para que bons resultados sejam obtidos, bons arquivos de entrada devem ser criados. Compreender os algoritmos de escalonamento é essencial para que sejam bolados bons arquivos de entrada capazes de ressaltar as características de cada algoritmo. Não esqueça de informar nos slides a configuração dos computadores utilizados para executar os experimentos. Para garantir que os tempos requisitados sejam respeitados ao máximo, evite rodar outros processos que consumam muita CPU e memória enquanto os experimentos estiverem sendo executados. O ideal mesmo é que você mantenha apenas os terminais com o código do EP rodando. 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-seu_nome`. Por exemplo: `ep1-katie_bouman`. Entregas que sejam tarbombs ou que, quando descompactadas, 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 individualmente.

Obs.: não inclua no `.tar.gz` itens que não foram pedidos neste enunciado, como por exemplo, dotdirs como o `.git`, dotfiles como o `.gitignore`, saídas para diversas execuções, arquivos pré-compilados, etc.... A presença de conteúdos não solicitados no `.tar.gz` levarão a um desconto de 1,0 na nota final do EP.