MAC0422 – Sistemas Operacionais – 1s2024
EP1 (Individual)
Data de entrega: 2/4/2024 até 8:00:00 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 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 `newsh`

O shell, chamado de `newsh`, 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/ps a`
- `/bin/ls --color=never -1t`
- `./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 <diretorio>`
- `rm <arquivo>`
- `uname -a`

Não se preocupe em tratar os erros dos 3 comandos acima. O usuário nunca vai colocar um nome de diretório que não existe ou um nome de arquivo que não existe. Para descobrir qual chamada de sistema deve ser usada na implementação de cada comando acima, olhe a manpage `syscalls` e depois a manpage da syscall específica. Na maioria das vezes, o nome da syscall a ser usada terá similaridade

com o nome do comando que deve ser implementado. Use essa informação como dica para descobrir qual syscall você precisará usar. As syscalls aceitas precisam estar definidas na manpage `syscalls`.

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 usuário, seguido de espaço e da hora atual (`HH:MM:SS`) entre colchetes, seguido de dois pontos e de um espaço em branco, como no exemplo abaixo que mostra o shell pronto para rodar o comando `ps`:

```
lidenbrook [13:04:08]: /bin/ps a
```

Tanto o usuário quanto a hora atual precisam ser capturadas pelo shell, para serem exibidas no prompt, usando chamadas de sistema seguindo as mesmas restrições explicadas acima referentes aos comandos `cd`, `rm` e `uname`, 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. No caso do nome do usuário, muito provavelmente a chamada de sistema mais próxima será uma que retorna o ID do usuário. Será necessário usar (ou implementar) outra função, não necessariamente outra syscall, que receba esse ID como argumento e retorne o nome (O mapeamento de ID para nome de usuário fica no arquivo `/etc/passwd`. Essa informação vai ser útil para você descobrir como converter o ID para o nome do usuário).

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 deadline t0 dt
```

`nome` é uma string sem espaços em branco de no máximo 16 caracteres ASCII que identifica o processo, `deadline` é o instante de tempo até o qual o usuário gostaria que aquele processo terminasse, `t0` é o instante de tempo em segundos quando o processo chega no sistema e `dt` é o quanto de tempo real dos núcleos de processamento deve ser destinado para aquele processo simulado. `deadline`, `t0` e `dt` são números naturais e `deadline` >= `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 11 1 10
```

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 11 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 11 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 usadas 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 escalonadores de processos existem. Neste EP o simulador deve implementar os seguintes escalonadores:

1. Shortest Job First
2. Round-Robin
3. Escalonamento com prioridade (usando o `deadline` como critério para definir a quantidade de quantums dada a cada processo)

A invocação do simulador no `newsh` 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
```

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 os núcleos de processamento deixaram de rodar um processo para rodar outro.

2 Requisitos

A compilação do código deve gerar dois executáveis. Um executável do `newsh` 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). 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 `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** 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 .c e 1 único arquivo .h com a implementação do shell;
- 1 único arquivo .c e 1 único arquivo .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 **.pdf** com no máximo 12 slides (contando todos os slides, inclusive capa, roteiro e referências, se houverem), justificando se o código 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. **Esses slides não serão apresentados. Eles devem ser preparados supondo que você teria que apresentá-los**;
- 3 arquivos de entrada usados para os experimentos relatados na apresentação.

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 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 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. 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.

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.