Infraestrutura de Software
Implementação 1

Atenção:

- **Respostas submetidas além do horário limite estabelecido serão descartadas;**
- **Os arquivos de implementação e Makefile deverão ser salvos dentro de um diretório cujo nome será formado pelas iniciais do e-mail em minúsculas. Exemplo: considerando que o e-mail é est@cesar.school, o diretório será** `est` **Qualquer diretório enviado fora desse padrão será automaticamente descartado;**
- **O diretório será compactado em um arquivo .** `tar`**, e este deve ser disponibilizado para o professor via Form do Google Classroom. O nome do arquivo será formado pelas iniciais do e-mail em minúsculas. Exemplo: considerando que o e-mail é est@cesar.school, o arquivo a ser entregue será** `est.tar` **. Qualquer arquivo enviado fora desse padrão será automaticamente descartado;**
- **Serão apenas aceitos arquivos submetidos via Form do Google Classroom;**
- **Identificada a cópia de qualquer questão, seja com relação a outra/o aluna/o ou de alguma outra fonte, TODA a submissão será descartada;**
- **Submissões que apresentarem erro na compilação feita por** `Makefile` **através simplesmente do comando** `make` **via terminal, serão descartadas.**

Interpretador por linha de comando (*shell*)

Um *shell*, mais simples que aquele executado num sistema Unix, deverá ser implementado nessa atividade, sendo este capaz de ser executado de duas formas: interativo ou *batch*. No primeiro, deve-se apresentar o *prompt* nomeado com o login de seu email, e a/o usuária/o do *shell* irá digitar o comando no *prompt*. No modo *batch*, o *shell* é iniciado tendo um arquivo como argumento (*batchFile)*, e este contem uma lista de comandos que devem ser executados. Neste último modo, o *prompt* não será apresentado, no entanto deve-se imprimir no terminal os comandos lidos do *batchFile* antes de executá-los. Nos dois formatos, o *shell* será finalizado com o comando `exit`.

Mais de um comando pode ser apresentado por linha de comando, com o ponto-e-vírgula (*;*) separando cada um. Por exemplo, se a/o usuária/o digitar

```
myLogin seq> /bin/ls; /bin/ps
```
(1)

, o *shell* deve executar ambos os comandos. Contudo, a execução irá depender do estilo (`style`) do *shell*: sequencial ou paralelo. No primeiro, cada ação deve ocorrer uma por vez, da esquerda para a direita, num mesmo processo filho. Assim, no exemplo (1), primeiro o `ls` deve ser executado, e quando finalizado, o `ps`. No estilo paralelo, ambas ações devem ser executadas em paralelo, por exemplo:

```
myLogin par> /bin/ls; /bin/ps
```
(2)

, nesse caso uma nova *thread* deve ser criada para cada comando, uma para o`/bin/ls` e outro para `/bin/ps`. Não há limites para o número de comando por linha.

Nos dois estilos, o *prompt* deve estar disponível apenas quando as duas ações terminarem. Perceba ainda que mais de duas ações podem ser combinadas. Para determinar o estilo, a/o usuária/o deve digitar o comando `style sequential` ou `style parallel` no *prompt*, sendo o sequencial o estilo *default*. Atenção para indicar no *prompt* qual estilo esta ativado: `seq` para sequencial, como no exemplo (1), ou `par` para paralelo, descrito no exemplo (2).

O *shell* ainda deve dar suporte:

1. *pipe* - a saída de um comando pode ser enviada como entrada de outro comando. Os comandos que estão sob o *pipe* estão separados pelo símbolo |. Por exemplo:

```
myLogin seq> ls -l | sort -k 5
```
(3)

, em que a saída do `ls -l` será a entrada do comando `sort`.

2. redirecionamento de entrada e saída: a entrada e saída de comandos podem ser enviadas ou obtidas de arquivos usando o redirecionamento. Alguns exemplos:

```
myLogin seq> date > datefile
```
(4)

, em que a saída do comando `date` é salvo no arquivo `datefile`

```
myLogin seq> a.out < inputifle
```
(5)

, em que o programa `a.out` recebe as entradas a partir do conteúdo do `inputfile`

```
myLogin seq> sort file.txt >> datafile
```
(6)

, em que o comando `sort` adiciona sua saída no fim do arquivo `datafile`

3. criar o comando `!!` (*history*) que permite o/a usuário/a executar o último comando executado no *prompt.* Caso nenhum comando tenha sido executado, a mensagem `No commands` deve ser apresentada no *prompt*.

4. executar comando em *background* através do caractere *&* ao fim do comando. Por exemplo, ao executar o comando (7)

```
myLogin seq> /bin/ls &
```
(7)

, a ação é colocada em *background*, e a informação sobre a mesma é apresentada no *prompt*:

```
myLogin par> [1] 1234
```
(8)

, em que é indicando o ID do processo (1234) e que é o primeiro colocado em *background* (`[1]`). Enquanto o comando estiver sendo executado em *background*, o *prompt* estará apto a aceitar novos comandos. Para o processo descrito em (8) retornar do *background*, deve-se digitar o comando `fg 1` no prompt.

Algumas especificações:

1. O programa deve usar `fork()`, `exec()`, `wait()`, `dup2()`, `pipe()`, `pthread_create()` e `pthread_join()`. O shell será um processo pai que executa, através de processos filhos (ou *threads*), o que for apresentado na linha de comando, seja digitada/a pela/o usuária/a ou lida do arquivo

2. O programa deve ser implementado em C e ser executável em sistemas Linux, Unix ou macOS, com a compilação feita por `Makefile`, através simplesmente do comando `make` via terminal, e retornar o arquivo com nome `shell` executável;

3. A execução deve ser realizada utilizando comando `./shell [batchFile]`, em que o *batchFile* é opcional. Se o argumento não estiver presente, o *shell* deve executar no modo interativo. ATENÇÃO: o nome do arquivo de entrada não necessariamente será `batchFile`.

O programa não deve falhar se encontrar um erro, precisando checar todos os parâmetros antes de aceita-los. Assim, seu programa deve apresentar um mensagem de erro coerente, ou continuar o processamento ou sair, a depender das situações, descritas abaixo:

1. imprimir uma mensagem e encerrar a execução quando:
    - um número incorreto de argumento para os comando `shell` for passado
    - se o arquivo a se utilizado no modo *batch* não existir ou não puder ser aberto

2. imprimir uma mensagem de erro e continuar processando
    - quando o comando não existe ou não puder ser executado

Algumas outras situações, que apesar de não serem erros, o *shell* deve tratar de forma coerente:

1. linha de comando vazia dentro do *prompt*
2. múltiplos espaços em branco dentro de uma linha de comando
3. espaços em branco antes ou depois do ponto-e-vírgula (*;*)
4. *batchFile* sem o comando `exit` e a/o usuária/o digitar `CTRL-D` no modo interativo

O que deve ser submetido pelo Form via Classroom:

- os arquivos com os códigos implementados
- incluir o `Makefile` para compilação e limpeza dos arquivos compilados

Pontuação

- Execução interativa, saída com exit e `style sequential` - 10%
    - Exemplo: `myLogin seq> /bin/ls; /bin/ps`
- Execução batch, saída com exit e `style sequential` - 10%
    - Exemplo: `./shell commands.txt`
- Execução batch e interativa, saída com exit e `style parallel` - 10%
    - Exemplo: `myLogin par> /bin/ls; /bin/ps`
- Execução batch e interativa, saída com exit, `style sequential` e PIPE - 10%
    - Exemplo: `myLogin seq> ls -l | sort -k 5; /bin/ps`
- Execução batch e interativa, saída com exit, `style parallel` e PIPE - 10%
    - Exemplo: `myLogin par> ls -l | sort -k 5; /bin/ps`

- Execução batch e interativa, saída com exit, `style sequential` e redirecionamento - 10%
    - Exemplo: `myLogin seq> date > datefile; /bin/ps`
- Execução batch e interativa, saída com exit, `style parallel` e redirecionamento - 10%
    - Exemplo: `myLogin par> date > datefile; /bin/ps`
- Execução batch e interativa, saída com exit, `style parallel` or `sequencial` e *history* - 10%
    - Exemplo: `myLogin seq> !!`
- Execução batch e interativa, saída com exit, `style parallel` or `sequencial` e *background* - 10%
    - Exemplo: `myLogin seq> date > datefile &`
- Tratamento de erro e outras situações - 10%