# 1º TP[1]

# Execução e Escalonamento de Tarefas

Segunda-feira, 10 de outubro de 2022.

---

O objetivo deste trabalho é desenvolver dois serviços de sistema operacional: Execução e Escalonamento de Tarefas (*Task Execution and Scheduling*). Esses serviços devem ser implementados em um programa chamado tes, que quando for executado deve exibir o *prompt* abaixo.

```
tes >
```

## 1. Execução de Tarefas

Este serviço permite executar simultaneamente até quatro programas da linguagem LPAS, definida no Apêndice I. Para iniciar a execução de um programa considere duas opções, o usuário digita:

i. O nome de um programa que deseja executar.

```
tes > soma
```

ii. Os nomes de dois a quatro programas separados por espaço para serem executados simultaneamente.

```
tes > add sub mul div
```

Quando os programas LPAS encerrarem sua execução, o *prompt* tes deve ser exibido novamente. Para encerrar o programa tes e retornar ao sistema operacional deve-se digitar o comando exit.

```
tes > exit
```

O programa tes deve ler as tarefas de um arquivo texto de extensão lpa,. Veja os exemplos de programas LPAS no anexo deste texto e no Apêndice I.

Para facilitar para o usuário a identificação de qual programa que está em execução, por exemplo, lendo ou escrevendo, sempre que um instrução LPAS de leitura (READ) ou escrita (WRITE) for executada, ela deve ser precedida do nome da tarefa, assim:

```
add -> READ: 10
sub -> READ: 8
sub -> READ: 20
add -> READ: 3
add -> WRITE: 13
sub -> WRITE: -12
```

[1] Atualizado em 14/10/2022.

Para cada grupo de duas tarefas a serem executadas, o programa deve criar um processo usando a chamada de sistema fork. Considerando o exemplo acima, em que é solicitado a execução de quatro programas LPAS (add, sub, mul e div), um processo tes1 executará as tarefas add e sub, e um outro processo tes2 executará as tarefas mul e div. Nesse exemplo, o programa terá então três processos, o processo pai tes e os processos filhos tes1 e tes2. Se for executado um ou dois programas LPAS, deve-se criar apenas um processo filho. Sendo assim, o serviço de executar as instruções do programa LPAS e de revezar essas tarefas no processador deve ser implementado nos processos filhos. A função do processo pai será:

1. Exibir o *prompt* tes;
2. lê e carregar os programas LPAS do disco para a memória;
3. criar o(s) processo(s) filho(s);
4. aguardar a conclusão do(s) processo(s) filho(s);
5. finalizar o programa quando o usuário digitar exit.

Portanto, dentre as funções de um sistema operacional, pode-se dizer que o processo pai implementa a carga do programa do disco para memória e a criação de processos, enquanto que os processos filhos são executores e escalonadores de tarefas.

## 2. Escalonamento de Tarefas

Este serviço deve implementar um escalonador preemptivo que utiliza o algoritmo de escalonamento *Round-Robin* para realizar a execução simultânea das tarefas.

Considere a fatia de tempo (*quantum*) igual a 2 ut (unidades de tempo) e que cada ut corresponde a 1 instrução LPAS. Sendo assim, a tarefa que possui oito instruções LPAS necessita de 8 ut para executar essas instruções, logo o seu tempo de CPU (tempo de processamento) é igual a 8 ut.

O algoritmo *Round-Robin* deve considerar o instante de tempo em que cada tarefa entra na fila do processador (fila de tarefas prontas) para execução. Esse instante é obtido de acordo com ordem das tarefas ao executá-las, por exemplo, de acordo com o *prompt* abaixo, a tarefa add entra na fila do processador no instante de tempo zero, a tarefa sub ingressa no instante 1 ut, mul em 2 ut e div em 3 ut.

```
tes > add sub mul div
```

Sempre que o programa LPAS executa uma instrução de entrada dados (READ) a tarefa deve ter a sua execução interrompida e permanecer bloqueada (suspensa) até que ocorra um evento que permita desbloqueá-la, ou seja, até que o usuário digite o número e tecle ENTER. Essa instrução é a simulação de uma chamada de sistema que é usada para informar ao programa tes que a tarefa deve ser suspensa e só retornar a ficar pronta após 3 ut (unidades de tempo). Logo, o tempo de E/S dessa instrução é sempre igual a 3 ut. Se uma tarefa possui 4 instruções READ, o seu tempo total de E/S será de 12 ut.

Qualquer instrução inválida no programa LPAS deve ter sua execução abortada pelo processo filho. Um relatório com os erros de execução deve ser exibido para informar quais são as tarefas e qual o erro ocorrido. Use o modelo de mensagem abaixo.

- Erros de execução LPAS:

1. add: instrução LPAD inválida
2. div: argumento de instrução LPAS inválido

Após concluir a execução de todas as tarefas LPAS, o processo filho deve exibir um relatório com o resultado da execução das tarefas gerenciadas por ele com as seguintes informações para cada tarefa.

1. Tempo de CPU (ut).

2. Tempo de E/S (ut).

3. A taxa percentual de ocupação do processador (CPU) em relação ao tempo total de sua utilização por todas as tarefas.

E os tempos abaixo (em segundos) para todos os processos que executaram.

4. Tempo médio de execução (*turnaround time*).

5. Tempo médio de espera (*waiting time*).

Deve-se usar o leiaute abaixo para exibir esse relatório.

- Processo tes1

  - Tarefa: add.lpas
  Tempo de CPU = 7 ut
  Tempo de E/S = 6 ut
  Taxa de ocupação da CPU = 47%

  - Tarefa: expression.lpas
  Tempo de CPU = 8 ut
  Tempo de E/S = 0 ut
  Taxa de ocupação da CPU = 53%

  - *Round-Robin*
  Tempo médio de execução = 7,5 s
  Tempo médio de espera = 2,5 s

- Processo tes2

  - Tarefa: sub.lpas
  Tempo de CPU =
  .....
  - Tarefa: mul.lpas
  Tempo de CPU =
  .....
  - *Round-Robin*
  Tempo médio de execução =
  .....

☞ **Atenção:** Os relatórios com os erros de execução dos programas LPAS e o resultado de execução das tarefas devem ser exibidos por cada processo filho. Portanto, os tempos médio de execução e espera sempre serão calculados para duas tarefas.

## - Critérios de avaliação

O trabalho será avaliado considerando:

1. Estrutura da solução:

    a. A validação dos dados fornecidos pelo usuário.
    b. A lógica empregada na solução do problema.
    c. O funcionamento do programa.
    d. Escrever funções específicas, ou seja, com atribuição clara e objetiva.
    e. Não escrever código redundante.
    f. Código-fonte sem erros e sem advertências do compilador.
    g. Código-fonte legível, indentado, organizado e comentado.
    h. Identificadores significativos para aprimorar a inteligibilidade do código-fonte.

2. O programa deve ser desenvolvido no sistema operacional Linux usando apenas a Linguagem C padrão ISO[2] e a GNU C *Library*[3], que inclui a API POSIX. Programas desenvolvidos em outras linguagens, mesmo que parcialmente, receberão nota zero.

3. Para que o programa seja avaliado, o código deve executar com sucesso. Programas que apresentarem erros de compilação, ligação ou *segmentation fault* receberão nota zero.

4. Trabalhos com plágio, ou seja, programas com código fonte copiados de outra pessoa (cópia integral ou parcial) receberão nota zero.

5. O desenvolvimento do trabalho é individual.

6. O programa deve ser composto de um único arquivo de cabeçalho (tes.h) e um único código-fonte (tes.c).

7. Os códigos-fontes (.c), os arquivos de cabeçalho (.h) e as tarefas LPAS (.lpas) devem ser codificados como UTF-8.

8. É proibido modificar os nomes de arquivos, identificadores, os protótipos de função, as declarações e/ou definições de funções fornecidos neste texto ou em anexo.

9. Deve-se implementar o programa usando as definições fornecidas no arquivo anexo tes.h.

[2] Documentação da Linguagem C padrão ISO: https://en.cppreference.com/w/c
[3] Documentação da GNU C *Library*: https://www.gnu.org/software/libc/

10. É permitido acrescentar novas declarações e/ou definições de tipos de dados, variáveis, constantes e funções, desde que estejam de acordo com os critérios acima.

## - Instruções para entrega do trabalho

1. Compacte os dois arquivos para criar um arquivo 7z com o seu nome e sobrenome, por exemplo: NelsonMandela. Use o software livre 7-Zip, que está disponível em https://www.7-zip.org/download.html.
2. Envie o arquivo 7z para o e-mail marlon.silva@ifsudestemg.edu.br.

## - Data de entrega

Segunda-feira, 24 de outubro de 2022 até às 23:59.

## - Valor do trabalho

10,0 pontos

Prof. Márlon Oliveira da Silva
marlon.silva@ifsudestemg.edu.br

**Apêndice I**

## LPAS - Linguagem de Programação para Aritmética Simples

Uma linguagem de máquina denominada LPAS - Linguagem de Programação para Aritmética Simples - é usada para escrever programas que executam as operações aritméticas somente com números inteiros. As instruções LPAS são apresentadas na Tabela 1.

LPAS possui a seguinte organização:

1. Cada linha de código deve ter apenas uma única instrução.
2. As instruções e os seus argumentos variáveis sejam todos escritos em maiúsculo.
3. Os comentários devem iniciar com ponto-e-vírgula.
4. Os programas LPAS são executados por uma Máquina de Execução (ME) ou processador.
5. A ME é responsável por carregar o programa para a memória e executá-lo.
6. A ME possui duas memórias (código e dados) e um único registrador que permite armazenar um número inteiro.
7. A memória de código armazena os programas LPAS a serem executados pela ME.
8. A memória de dados armazena as variáveis do programa que está em execução.

Exemplos de programas LPAS:

```
; Programa: add.lpas
; Descrição: Realização a soma de 2 números inteiros.
;
READ X      ; X recebe o valor lido do teclado
READ Y      ; Y recebe o valor lido do teclado
LOAD X      ; carrega o valor de X no registrador (registrador ⇦ X)
ADD Y       ; soma Y com X e coloca o resultado no registrador (registrador ⇦ registrador + Y)
STORE Z     ; armazena o resultado da soma na variável Z (Z ⇦ registrador)
WRITE Z     ; escreve na tela o valor de Z (o resultado da soma)
HALT        ; finaliza o programa

; Programa: expression.lpas
; Descrição: Calcula a expressão 20 + ((99 * 8) / 2)  -  3.
;
LOAD 99     ; registrador = 99
MUL 8       ; registrador = 792
DIV 2       ; registrador = 396
ADD 20      ; registrador = 416
SUB 3       ; registrador = 413
STORE X     ; X = registrador, ou seja, X recebe 413
WRITE X     ; exibe na tela o número 413 (o resultado da expressão aritmética)
HALT        ; finaliza o programa
```

**Tabela 1:** Instruções de máquina LPAS

| Código | Instrução | Descrição |
|---|---|---|
| 10 | READ | Exibe o *prompt* "READ: ", lê o seu valor do teclado e armazena na variável.<br><br>Ex.: **READ X** ; lê um valor do teclado e armazena na variável X, p. ex., se X for 5<br>READ: 5 |
| 11 | WRITE | Escreve na tela a mensagem "WRITE: " seguida do valor armazenado em uma variável.<br><br>Ex.: **WRITE X** ; escreve na tela o valor da variável X, p. ex., se X for 10<br>WRITE: 10 |
| 20 | LOAD | Carrega o valor de uma variável para o registrador.<br><br>Ex.: **LOAD X** ; registrador ⇦ X |
| 21 | STORE | Armazena o valor do registrador em uma variável.<br><br>Ex.: **STORE X** ; X ⇦ registrador |
| 30 | ADD | Adiciona o valor do registrador ao valor de uma variável ou número e armazena o resultado no registrador.<br><br>Ex.: **ADD X** ; registrador ⇦ registrador + X<br>**ADD 20** ; registrador ⇦ registrador + 20 |
| 31 | SUB | Subtrai o valor do registrador do valor de uma variável ou número e armazena o resultado no registrador.<br><br>Ex.: **SUB X** ; registrador ⇦ registrador - X<br>**SUB 20** ; registrador ⇦ registrador – 20 |
| 32 | MUL | Multiplica o valor do registrador com o valor de uma variável ou número e armazena o resultado no registrador.<br><br>Ex.: **MUL X** ; registrador ⇦ registrador * X<br>**MUL 20** ; registrador ⇦ registrador * 20 |
| 33 | DIV | Divide o valor do registrador pelo valor de uma variável ou número e armazena o resultado no registrador.<br><br>Ex.: **DIV X** ; registrador ⇦ registrador / X<br>**DIV 20** ; registrador ⇦ registrador / 20 |
| 40 | HALT | Finaliza o programa. |