

2º Trabalho¹

Escalonamento de Tarefas e Gerência de Memória

Terça-feira, 16 de março de 2021.

Desenvolva um programa chamado *tmm* para realizar o Escalonamento de Tarefas e a Gerência de Memória (*Task Scheduling and Memory Management*) em um sistema computacional composto de memória, processador, disco rígido e tarefas. As tarefas representam um conjunto de instruções a serem executadas pelo processador e as instruções que elas podem executar são de apenas três tipos:

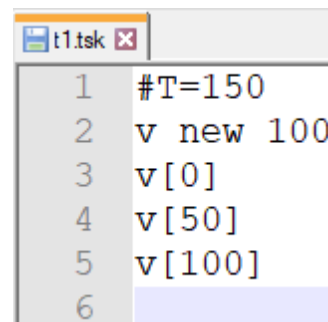
1. Alocação de memória
2. Acesso à memória
3. Acesso a disco

A tabela abaixo apresenta cada tipo.

Tipo	Instruções	Descrição
1	v new n	Alocação de memória: cria um vetor <i>v</i> com tamanho de <i>n</i> bytes de memória. Essa instrução solicita em tempo de execução uma ampliação ao tamanho da memória lógica do processo. Essa instrução simplesmente simula uma alocação de memória, nada realmente é alocado, ou seja, não é preciso realizar um <i>malloc</i> , mas registrar, contabilizar que o espaço de memória da tarefa aumentou.
2	v[p]	Acesso à memória: acessa a posição <i>p</i> do vetor <i>v</i> , a primeira posição válida do vetor é zero. Essa instrução simplesmente simula uma operação de acesso à memória, nada realmente é lido ou escrito na memória.
3	read disk	Acesso a disco: essa instrução simplesmente simula uma operação de entrada de dados, de acesso a disco, nada realmente é lido do disco. É a simulação de uma chamada de sistema que é usada para informar ao programa <i>tmm</i> que a tarefa deve ser suspensa e só retornar a ficar pronta após 5 ut (unidades de tempo). Portanto, o tempo de E/S é sempre igual a 5 ut. Se uma tarefa possui 4 instruções desse tipo, o seu tempo total de acesso a disco será de 20 ut.

O programa *tmm* deve ler as tarefas a serem executadas de um arquivo texto. Sendo assim, as instruções das tarefas são armazenadas em um arquivo de extensão *tsk* que deve ter uma única instrução por linha. Como no exemplo ao lado, que mostra a tarefa *t1.tsk*. Considere apenas uma linha em branco após a última instrução da tarefa.

Todo arquivo de tarefa deve começar com os caracteres **#T=** seguidos do tamanho em bytes da tarefa. No exemplo ao lado a tarefa *t1* possui 150 bytes. Esse valor corresponde ao tamanho da memória lógica da tarefa.



```

1 #T=150
2 v new 100
3 v[0]
4 v[50]
5 v[100]
6

```

Figura 1 - Tarefa *t1.tsk* com 4 instruções.

¹ Atualizado em 17/03/2021.

No *prompt* de comandos do sistema operacional o usuário deve especificar para o programa tsmm as tarefas que ele deseja executar. Veja o exemplo abaixo em que quatro tarefas são fornecidas, esse é o número máximo de tarefas que podem ser executadas ao mesmo tempo.

```
marlon@inspiron:~$ tsmm t1 t2 t3 t4
```

Figura 2 - Execução do programa tsmm.

O algoritmo de escalonamento *Round-Robin* deve ser implementado para revezar o processador entre as tarefas, permitindo a execução simultânea das tarefas. Considere a fatia de tempo igual a 2 ut (unidades de tempo) e que cada ut corresponde a 1 instrução. Sendo assim, a tarefa que possui quatro instruções, como a tarefa t1 da Figura 1, necessita de 4 ut para executar suas instruções, logo o seu tempo de CPU é igual a 4.

O tsmm utiliza a paginação como mecanismo de gerência de memória e possui as seguintes características:

Memória física = 64 KB

Memória lógica da maior tarefa = 4 KB

Páginas lógicas e físicas = 512 bytes

A gerência de memória do tsmm deve fazer as seguintes validações ao executar uma instrução.

Tipo	Instruções	Gerência de Memória
1	v new n	Verifica se o tamanho de memória a ser alocada pode ser atendido pela memória física, ou seja, se há espaço livre na memória física e também pela memória lógica da tarefa, que é limitada a 4 KB. O número de páginas lógicas da tarefa é calculado de acordo com o tamanho da memória lógica da tarefa e o valor de n dessa instrução. Por exemplo, como a tarefa t1 (Figura 1) possui memória lógica de 150 bytes e sua única instrução de tipo 1 aloca 100 bytes, essa tarefa terá uma memória lógica de 250 <i>bytes</i> , o que exigirá uma página lógica.
2	v[p]	Verifica se o acesso à posição p do vetor v é válido. Para isso o programa deve calcular o endereço lógico e físico de acordo com o funcionamento da técnica de paginação, ou seja, deve-se gerar um endereço lógico e obter um endereço físico usando a tabela de páginas da tarefa. O endereço lógico deve ser gerado considerando que a memória lógica começa no endereço zero e vai até o tamanho da memória lógica menos um. Por exemplo, a tarefa t1 (Figura 1), que possui 250 bytes de memória lógica, terá os endereços lógicos no intervalo de 0 a 249. O endereço da memória física para armazenamento das tarefas começa no endereço 20.480 e vai até 65.535, porque os primeiros 20 KB é usado pelo tsmm, ou seja, é reservado para uso do sistema.
3	read disk	Nenhuma verificação é necessária.

Qualquer instrução presente no arquivo da tarefa diferente das três instruções acima deve ter sua execução cancelada. Se forem passadas três tarefas ao programa e duas estiverem com as instruções corretas, cancela a execução de uma tarefa e executa as outras duas que possuem as instruções válidas. O programa deve informar quais tarefas não serão executadas por terem instruções inválidas, ou seja, que não são de nenhum dos três tipos de instruções permitidas, assim:

A tarefa t3 não será executada, pois tem instruções diferentes do tipo 1, 2 e 3.

Qualquer tentativa de acessar um endereço inválido por qualquer instrução de tipo 1 ou 2, deve fazer com que o programa tsmm aborte a execução da tarefa informando ao usuário o motivo. Por exemplo, a instrução `v[100]` da tarefa `t1` (Figura 1) solicita um acesso a um endereço que não pertence à memória lógica da tarefa, pois a posição 100 não foi alocada para o vetor `v`, apenas as posições de memória de 0 a 99. Portanto, como essa instrução não pode ser executada, a tarefa `t1` deve ser abortada e exibida a mensagem abaixo.

A tarefa `t1` tentou realizar um acesso inválido à memória: `v[100]`

O programa deve calcular o tempo total de processador de cada tarefa a ser executada contando o número de linhas de instruções armazenadas no arquivo texto da tarefa. No exemplo da tarefa `t1` (Figura 1), esse tempo é igual a 4 ut.

Uma estrutura para representar as informações da tarefa deve ter os seguintes campos:

estado: indica o estado atual da tarefa. Os valores válidos são: *nova*, *pronta*, *executando*, *suspensa* e *terminada*. O estado da tarefa deve ser atualizado segundo o seu ciclo de vida durante sua execução.

tempoCPU: indica o tempo total de processador que a tarefa necessita para execução de suas instruções.

tempoES: tempo total necessário para tarefa realizar todas as operações de entrada e saída de dados, ou seja, instruções do tipo 3.

Tamanho: indica, em bytes, o tamanho do espaço de memória da tarefa, considerando sua memória lógica e o somatório de todas as alocações de memória realizadas por instruções do tipo 1.

O algoritmo de escalonamento *Round-Robin* deve considerar o instante de tempo em que a tarefa entra na fila do processador (fila de tarefas prontas) para execução. Esse instante é obtido de acordo com ordem das tarefas no *prompt* de comandos do sistema operacional, por exemplo, considerando a Figura 2, a tarefa `t1` entrou no instante de tempo zero, a tarefa `t2` ingressou no instante de tempo 1 ut, `t2` em 2 ut e `t3` em 4 ut.

Após executar todas as tarefas, o programa deve exibir um relatório de execução com as seguintes informações para cada tarefa.

1. Tempo médio de execução (*turnaround time*).
2. Tempo médio de espera (*waiting time*).
3. A taxa percentual de ocupação do processador em relação ao tempo total de sua utilização por todas as tarefas.
4. A taxa percentual de ocupação do disco em relação ao tempo total de sua utilização por todas as tarefas.
5. Os instantes de tempo que cada tarefa usou o processador e o disco, por exemplo, considerando a Figura 3, o processo `P1` usou processador e o disco assim:

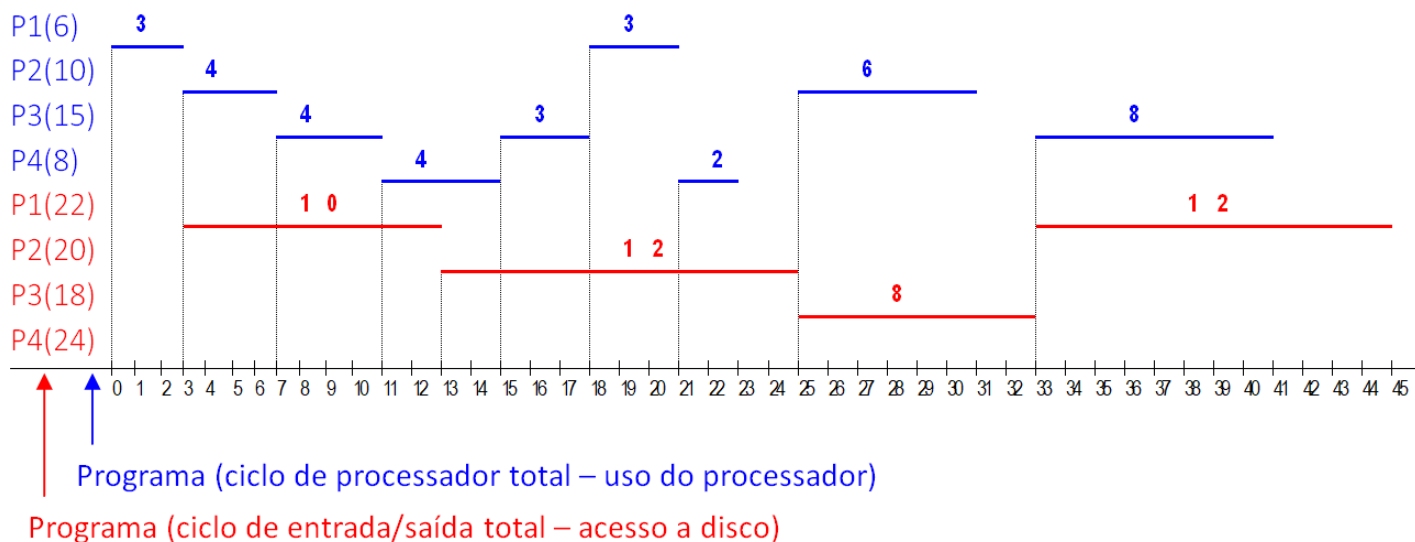
Processador: 0 a 3 ut, 18 a 21 ut
Disco.....: 3 a 13 ut, 33 a 45 ut

- Os endereços lógicos e físicos de todas as instruções do tipo 1 e 2, mas apenas o endereço inicial dessas instruções nas páginas lógicas e físicas da tarefa. Usar o formato abaixo para exibir os endereços. Os valores de página lógica, página física e deslocamento devem ser exibidos em decimal.

página : deslocamento

- A tabela de páginas que possui apenas os endereços das páginas físicas.

Figura 3 - Diagrama de Gantt



- Critérios de avaliação

- O trabalho será avaliado considerando:
 - A validação dos dados fornecidos pelo usuário.
 - A lógica empregada na solução do problema.
 - O funcionamento do programa.
 - O conhecimento da linguagem de programação C.
 - O uso do princípio do menor privilégio². Veja um exemplo abaixo.
 - Código fonte sem erros e sem advertências do compilador.
 - Código fonte legível, indentado, organizado e comentado.
 - Identificadores significativos para aprimorar a inteligibilidade do código fonte.
- O programa deve ser desenvolvido integralmente usando apenas a linguagem C padrão ISO³ e a *C Standard Library*. Programas desenvolvidos em outras linguagens, mesmo que parcialmente, receberão nota zero.
- 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.
- Trabalhos com plágio, ou seja, programas com código fonte copiados de outra pessoa (cópia integral ou parcial) receberão nota zero.

² O **princípio do menor privilégio** declara que deve ser concedido ao código somente a quantidade de privilégio e acesso de que ele precisa para realizar sua tarefa designada, não mais que isso.

³ A Linguagem C padrão ISO pode ser consultada em <https://en.cppreference.com/w/c>.

5. O desenvolvimento do trabalho é individual.
6. Escrever funções e métodos específicos, ou seja, com atribuição clara e objetiva.

Exemplo: Pesquisa um nome em um vetor de *strings*. Retorna a posição do nome no vetor se ele for encontrado ou -1 caso contrário.

```
int pesquisarNome(const char* vetor[], const char* nome);
```

A descrição dessa função deixa claro que não é atribuição dela ler o nome via algum dispositivo de E/S e nem exibir o resultado da consulta, somente realizar a pesquisa do nome no vetor e devolver o resultado.

O uso do *const* no protótipo de função acima é um exemplo do princípio do menor privilégio, porque se a função não precisa alterar os argumentos usados na sua chamada, ela não pode ter parâmetros formais com esse poder ou privilégio. O uso do *const* assegura que apesar da função receber a referência dos argumentos ela não poderá modificá-los.

7. Não escrever código redundante.
8. O programa deve ser composto de um único arquivo de cabeçalho (tsmm.h) e um único código-fonte (tsmm.c).

- Instruções para entrega do trabalho

1. Antes de enviar sua solução via SIGAA, compacte os dois arquivos (tsmm.h e tsmm.cpp) para criar um arquivo 7z com o seu nome e sobrenome, por exemplo: AyrtonSenna.7z.
2. Use o software livre de código aberto 7-Zip, que está disponível em <https://www.7zip.org/download.html>.

- Data de entrega

Quarta-feira, 24 de março de 2021.

- Valor do trabalho

10,0 pontos.

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