

# Primeiro Trabalho Prático

Este trabalho tem por objetivo o melhor entendimento sobre gerenciamento de processos em um sistema operacional.

## 1 O Trabalho

A sua tarefa é criar um programa para simular 5 funções de gerenciamento de processos: criação de processo, substituição do processo atual por outro processo, transição de estados de um processo, escalonamento e troca de contexto.

Nós iremos utilizar chamadas de sistema do linux como `fork()`, `wait()`, `pipe()` e `sleep()`. Leio o manual dessas chamadas de sistema para mais detalhes.

O simulador consiste em 3 tipos de processos: *commander*, *process manager* e *reporter*. Existe um processo *commander* (esse processo inicia a simulação), um processo *process manager* que é criado pelo *commander* e vários processos do tipo *reporter* que são criados pelo *process manager*, quando necessário.

## 2 O processo *commander*

O processo *commander* primeiramente cria um *pipe* e depois um processo do tipo *process manager*. Ele então lê comandos repetidamente (1 por segundo) da entrada padrão e os passa para *process manager* através do *pipe*. Existem 4 tipos de comandos:

1. **Q**: Fim de uma unidade de tempo.
2. **U**: Desbloqueie o primeiro processo simulado que está na fila de bloqueados.
3. **P**: Imprima o estado atual do sistema.
4. **T**: Imprima o tempo de retorno médio e finalize o simulador.

O comando **T** aparece apenas uma vez, como sendo o último comando.

## 3 O processo simulado

A simulação de gerenciamento de processo gerencia a execução de **processos simulados**. Cada processo simulado consiste em um programa que manipula o valor de um única variável inteira. Sendo assim, o estado de um processo simulado em um instante de tempo consiste no valor da sua variável inteira e o valor de seu contador de programa. Um processo simulado é formado por uma sequência de instruções. Existem 7 tipos de instruções:

1. **S** *n*: Atualiza o valor da variável inteira para *n*.
2. **A** *n*: Soma *n* na variável inteira.
3. **D** *n*: Subtrai *n* na variável inteira.
4. **B**: Bloqueia o processo simulado.
5. **E**: Termina o processo simulado.
6. **F** *n*: Cria um novo processo simulado. O novo processo é uma cópia exata do pai. O novo processo executa da instrução imediatamente após a instrução **F**, enquanto o pai continua *n* instruções após **F**.
7. **R** nome do arquivo: Substitui o programa do processo simulado com o programa no arquivo nome do arquivo, e atualiza o valor do contador de programa para a primeira instrução do novo programa.

Um exemplo de um programa para um processo simulado segue abaixo:

```
S 1000
A 19
A 20
D 23
A 55
F 1
R file_a
F 1
R file_b
F 1
R file_c
F 1
R file_d
F 1
R file_e
E
```

Você pode armazenar o programa de um processo simulado em um vetor, com uma instrução para cada posição.

## 4 O processo *Process Manager* (PM)

O processo *Process Manager* simula 5 funções de gerenciamento de processos: criação de um novo processo, substituição de um processo em execução por outro (troca da imagem) , gerenciamento de transições de estado, escalonamento de processo e troca de contexto. Além disso, ele cria um processo *Reporter* sempre que precisa imprimir o estado do sistema.

O *Process Manager* cria o primeiro processo simulado (id = 0). O programa para esse processo é lido de um arquivo com o nome *init*. Esse é o único processo criado por conta própria pelo *Process Manager*. Todos os outros processos são criados em resposta a execução da instrução **F**.

## 5 *Process Manager* estruturas de dados

O *Process Manager* mantém 6 estruturas de dados: Tempo, CPU, TabelaPcb, EstadoPronto, EstadoBloqueado e EstadoExecutando. Tempo é um inteiro inicialização com 0. CPU é usado para simular a execução de um processo simulado que está no estado executando. Essa estrutura tem que conter dados como: ponteiro para o array do programa, valor atual do contador de programa, valor inteiro e fatia de tempo do processo simulado. Também deve guardar o número de unidades de tempo usadas até agora na fatia de tempo atual.

TabelaPcb é um array com uma entrada para cada processo que ainda não terminou sua execução. Cada entrada deve incluir dados como: id do processo, id do processo pai, um ponteiro para o contador de programa, o valor inteiro, prioridade, estado, tempo de início e CPU utilizada até agora.

EstadoPronto contém os índices dos processos na TabelaPcb que estão prontos para serem executados. A mesma lógica seque para EstadoBloqueado e EstadoExecutando.

## 6 *Process Manager*: processando os comandos de entrada

Após criar o primeiro processo e inicializar todas as suas estruturas de dados, o *Process Manager* recebe e processa uma comando por vez recebido do processo *commander*. Ao receber um **Q**, o processo executa a próxima instrução do processo que está sendo executado, incrementa o valor do contador de programa, incrementa Tempo e realiza o escalonamento.

Ao receber um comando **U**, o PM move o primeiro processo bloqueado para a fila de prontos. AO receber um **P**, o PM cria um processo *reporter*. Ao receber um comando **T**, o PM cria um processo *reporter* e finaliza após o fim do *reporter*.

## 7 *Process Manager*: simulando processos

O PM executa a próxima instrução do processo atualmente em execução após receber um comando **Q**. Note que a execução é confinada a estrutura de dados CPU.

As instruções **S**, **A** e **D** atualizam o valor inteiro armazenado em CPU. A instrução **B** move o processo atual para o estado bloqueado e começa a executar um novo processo. Isso irá resultar em uma troca de contexto. A instrução **E** termina o processo atual, desaloca a memória e atualiza a TabelaPcb. Um novo processo precisa então ser executado.

A instrução **F** resulta na criação de um novo processo simulado. Uma nova entrada é criada na TabelaPcb. Um novo id é dado a esse processo e o id do processo pai também é armazenado. O tempo de início é inicializado com o tempo atual e o tempo de CPU é inicializado com 0. O programa simulado é uma cópia do programa pai. O novo programa simulado é criado com estado pronto.

Finalmente a instrução **R** resulta na troca da imagem do processo atual. Seu array de programa é sobrescrito pelo código em nome de arquivo, o contador de programa é colocado em 0 e o valor da variável inteira é indefinida. Essa mudanças são realizadas somente na estrutura CPU.

## 8 *Process Manager*: escalonamento

O PM também implementa uma estratégia de escalonamento. Várias políticas de escalonamento podem ser implementadas.

## 9 *Process Manager*: troca de contexto

A troca de contexto envolve a copia do estado do programa atual da CPU para a TabelaPcb, e a cópia do novo processo criado da TabelaPcb para a CPU.

## 10 *Processo Reporter*

O processo *Reporter* imprime o estado atual do sistema:

```
*****
Estado do sistema:
*****\\

TEMPO ATUAL: tempo
PROCESSO EXECUTANDO:
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
BLOQUEADO:
Fila processos bloqueados:
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
...
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
PROCESSOS PRONTOS:

pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
*****
```

Observações sobre a entrega:

- (a) O programa deve estar bem indentado e comentado
- (b) O programa não deve fazer uso de comando goto
- (c) Caso apareçam números fixos no código, estes devem ser definidos como constantes.
- (d) Trabalhos copiados serão penalizados conforme informado em sala.

- (e) O trabalho pode ser feito em grupos de, **no máximo**, três pessoas.
- (f) A parte de implementação do trabalho deverá ser entregue em um único arquivo compactado, com o nome dos integrantes (por exemplo, Fulano\_de\_Tal\_Beltrano\_de\_Qual.zip). Explique, em um arquivo “leiam.txt”, as instruções para compilação (todos os trabalhos deverão ser desenvolvidos utilizando o Sistema Operacional Linux)
- (g) Nesse zip não deve haver arquivos executáveis.
- (h) Incluir pdf da parte escrita no zip.
- (i) A entrega dos arquivos deverá ser feita via portal didático e a fórmula para desconto por atraso na entrega é  $\frac{2^d-1}{0,32}\%$ , onde  $d$  é o atraso em dias. Note que após 6 dias, o trabalho não pode ser mais entregue. Ao final da descrição do trabalho, há outras informações disponíveis sobre a entrega.
- (j) Valor: 10 pontos

**O que deve ser entregue:**

- Documentação do trabalho: Em entre outras coisas, a documentação deve conter:
  1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. Implementação: descrição sobre a implementação do programa.
  3. Resultados e Discussões: dados obtidos das execuções dos algoritmos de ordenação sobre cada um dos conjuntos de dados (incluindo gráficos e tabelas); análise crítica dos dados obtidos em comparação com o esperado.
  4. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet, se for o caso.
- Além disso, neste trabalho deve ser enviado ao professor o arquivo fonte. A entrega deverá ser feita via moodle, seguindo as diretrizes informadas no início da descrição deste trabalho.

**Comentários Gerais:**

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
- Clareza, indentação e comentários no programa também vão valer pontos.