Arquitetura de Computadores e Sistemas Operacionais

SIMULAÇÃO DE ESCALONAMENTO DE PROCESSOS

Gustavo de Mendonça Freire — 123102270

Vítor Amaral Bedin — 123199405

Yuri Rocha de Albuquerque — 123166143

Sumário

Capítulo 1

Objetivos

O objetivo principal do trabalho é implementar um simulador de um escalonador de processos na linguagem de programação *C* que utilize a estratégia de seleção *Round Robin* com *feedback*, de modo que o próprio usuário do programa possa escolher quais processos serão escalonados e, se desejar, outros parâmetros do escalonador como tempo de preempção e tempo de *I/O* de cada dispositivo.

Além disso, o simulador deve mostrar ao usuário, a cada instante de tempo da simulação, uma *snapshot* contendo as principais informações do escalonador, como os processos de cada fila, que tipo de ações o escalonador está executando e os eventos sofridos pelos processos. Por fim, o programa deve gerar um texto que explica de forma sucinta tudo o que ocorreu durante o escalonamento (preempções, pedidos de *I/O* etc.)

Capítulo 2

Premissas

- **Limite máximo de processos criados**

  Como o simulador aloca memória dinamicamente sob demanda, não há um limite implementado para o número de processos criados.

- **Valor da fatia de tempo dada aos processos em execução**

  Escolhido pelo usuário na linha de comando da chamada do programa utilizando o argumento `--time` seguido do tamanho da fatia de tempo em unidades de tempo. Caso o usuário não explicite o tempo na chamada do programa, o valor padrão de 5 unidades de tempo é utilizado.

- **Tempo de duração de cada tipo de *I/O* (disco, fita magnética, impressora)**

  Escolhidos pelo usuário na linha de comando da chamada do programa utilizando os argumentos `--disk`, `--tape` e `--printer`, cada um seguido do tempo de *I/O* do disco, da fita magnética e da impressora, respectivamente. Caso o usuário não explicite algum desses valores, o programa utilizará o valor padrão correspondente (4 unidades de tempo para o disco, 7 unidades de tempo para a fita e 9 unidades de tempo para a impressora).

- **Gerência de processos**

  - Definição do *PID* de cada processo:

    As definições dos *PID*s de cada processo são feitas de acordo com a ordem em que eles aparecem no arquivo *CSV* começando pelo *PID* 1 e aumentando de 1 em 1.

  - Escalonador:

    Há duas filas implementadas para processos prontos, sendo uma de baixa prioridade, para a qual voltam os processos que estavam realizando *I/O* no disco, e uma de alta prioridade, para a qual voltam os processos que estavam realizando *I/O* na fita magnética e na impressora. Além disso, para o nosso escalonador, consideramos que o sistema possui recursos infinitos, ou seja, não existe uma fila na qual os processos que pedem *I/O* entram, eles apenas adquirem o recurso assim que o solicitam.

- **Tipos de *I/O***

  - DISK (disco) — baixa prioridade
  - PRINTER (impressora) — alta prioridade
  - TAPE (fita magnética) — alta prioridade

- **Ordem de entrada na fila de prontos**

  A ordem de entrada na fila de prontos é feita da seguinte forma:

  Processos novos entram na fila de alta prioridade, processos que sofreram preempção entram na fila de baixa prioridade e processos que terminaram *I/O* entram na fila correspondente ao tipo de *I/O* realizado (especificado anteriormente). Se dois processos de tipos diferentes tentarem entrar na mesma fila ao mesmo tempo, processos novos têm prioridade sobre os outros e processos retornando de operações de *I/O* têm prioridade sobre processos que sofreram preempção. Em caso de empate de processos novos ou empate de processos que sofreram preempção, aquele com o menor *PID* entra primeiro na fila. Por fim, se dois processos voltando de operações de *I/O* tentam entrar numa mesma fila ao mesmo tempo, o que só pode ocorrer se ambos forem do mesmo tipo ou se um deles for do tipo TAPE e o outro do tipo PRINTER, vence aquele que tiver menor *PID* no caso de os tipos de *I/O* serem os mesmos e o tipo TAPE é favorecido em detrimento do tipo PRINTER, caso contrário.

- **Formato esperado**

  – Arquivo *CSV*:

  A separação entre as descrições de cada processo é dada pela quebra de linha e o formato esperado para cada linha do arquivo *CSV* consiste em um instante de tempo, seguido de uma sequência de pares do tipo (tipo de *I/O*, tempo de *CPU*), em que o primeiro elemento do primeiro par e/ou o último elemento do último par podem não existir, seguindo cronologicamente a execução daquele processo, da seguinte forma:

  ```
  T: IO_1, tempo_1, IO_2, tempo_2, ...
  ```

  Em que `T` é um inteiro não negativo que representa o instante do tempo de chegada do processo (considerando que o escalonamento começa no instante de tempo 0), `tempo_i` é um inteiro não negativo que representa o tempo de execução entre o começo do programa e o primeiro *I/O* ou fim do programa, ou entre dois pedidos de *I/O* ou entre um pedido de *I/O* e o fim do programa. Dessa forma, a soma de todos os `tempo_i` é o tempo de *CPU* total do processo. E, por fim, `IO_i` é um tipo de *I/O* a ser realizado naquele instante do processo (DISK, PRINTER ou TAPE).

  Por exemplo, a descrição *"O processo P1, que possui tempo de* CPU *igual a 10 segundos e pede um* I/O *do tipo Impressora no instante 6 de sua execução e um do tipo disco no instante 9, chega no instante de tempo 3."* geraria a seguinte linha no arquivo:

  ```
  3:  6, PRINTER, 3, DISK, 1
  ```

  – Comando de chamada do programa:

  O programa é chamado no terminal através do seu nome seguido do nome do arquivo *CSV* com as descrições dos processos e, opcionalmente, argumentos para determinar os tempos de cada tipo de operação de *I/O* e o *quantum* (fatia de tempo dada para cada processo na *CPU* antes da preempção) em qualquer ordem. Para selecionar cada tempo de cada tipo de *I/O*, deve-se utilizar:

  * `--disk` seguido de um inteiro não negativo para selecionar o tempo de execução do *I/O* do tipo disco;
  * `--tape` seguido de um inteiro não negativo para selecionar o tempo de execução do *I/O* do tipo fita magnética;
  * `--printer` seguido de um inteiro não negativo para selecionar o tempo de execução do *I/O* do tipo impressora;

* `--time` seguido de um inteiro não negativo para selecionar o *quantum* do escalonador;
* `--steps` caso o usuário deseje ver, a cada instante de tempo, o estado do escalonador (processos em execução, filas de espera e processos realizando I/O).

Por exemplo:
O comando

```
./nome_programa.out processos.csv --disk X --tape Y --printer Z --time W
```

chama o programa referenciando o arquivo `processos.csv` com tempo de disco de `X` unidades, tempo de fita magnética de `Y` unidades, tempo de impressora de `Z` unidades e *quantum* de `W` unidades de tempo.

Capítulo 3

Descrição dos Módulos

- `event.h` / `event.c`

  Módulos próprios que contêm as funções e o enum utilizados pelo escalonador para lidar com cada tipo de evento.

- `eventqueue.h` / `eventqueue.c`

  Módulos próprios criados que contém as estruturas de dados e funções que lidam com a implementação da fila de eventos (inserção de elementos, inicialização e tratamento de leitura de linhas de um arquivo).

- `main.c`

  Módulo principal, contendo a lógica para a execução do programa e união de todos os módulos.

- `options.h` / `options.c`

  Módulos próprios com funções e estruturas de dados relacionadas à leitura de opções de usuário na linha de comando da chamada do programa (tempo de cada tipo de I/O e tempo de preempção do escalonador, ambos em unidades de tempo)

- `process.h` / `process.c`

  Módulos contendo as estruturas de dados que compõem um processo e uma fila de processos, juntamente com funções para criar, manipular e testar essas estruturas.

- `scheduler.h` / `scheduler.c`

  Módulos próprios responsáveis por implementar toda a lógica de escalonamento de processos utilizando as opções determinadas pelo usuário para a tomada de decisões e todos os outros módulos do programa. É o coração do simulador.

- `display.h` / `display.c`

  Módulos próprios responsáveis por implementar toda a lógica da saída do programa, lidando com a formatação dos prints (escolha de cor para cada print, a informação dos processos que serão exibidas, etc).

Capítulo 4

Funcionamento do Escalonador

O escalonador, implementado no módulo `scheduler.c`, é orientado às filas de eventos, implementadas no módulo `eventqueue.c`. A cada instante de tempo, o escalonador verifica se há algum evento para ocorrer e, se há, que tipos de eventos devem ocorrer e quais processos eles afetam. Esses eventos podem ser: chegada de um novo processo, retorno da operação de *I/O*, ou a preempção de um processo. Ao serem executados, esses eventos podem engatilhar outros eventos, que serão adicionados à fila de eventos na posição correspondente ao tempo em que devem ocorrer.

Capítulo 5

Saídas Esperadas

Todas as saídas esperadas para este programa serão demonstradas através de um exemplo. Dito isso, utilizaremos o exemplo vindo de um arquivo *csv* contendo as seguintes linhas:

```
0: 4, TAPE, 9

4: 2, PRINTER, 4, TAPE, 5

5: 3, DISK, 1

10: 2, TAPE, 5, PRINTER
```

A primeira forma de saída para este exemplo descreve, em cada linha, o evento sofrido pelo processo e algum identificador do processo. Para isso, menciona o evento em si, pode mencionar o tipo de *I/O* (se tiver pedido em algum momento), pode mencionar o *PID* do processo e em qual fila o processo entrou após ter finalizado algum evento.

```
New process with PID 1
Process 1 entered the high priority queue
Process 1 has started executing
Process 1 has triggered TAPE IO
New process with PID 2
Process 2 entered the high priority queue
Process 2 has started executing
New process with PID 3
Process 2 has triggered PRINTER IO
Process 3 entered the high priority queue
Process 3 has started executing
Process 3 has triggered DISK IO
New process with PID 4
Process 4 entered the high priority queue
Process 4 has started executing
Process 1 finished its IO
Process 4 has triggered TAPE IO
Process 1 entered the high priority queue
Process 1 has started executing
Process 3 finished its IO
```

```
Process 2 finished its IO
Preempted process 1
Process 2 entered the high priority queue
Process 2 has started executing
Process 4 finished its IO
Process 2 has triggered TAPE IO
Process 4 entered the high priority queue
Process 4 has started executing
Process 4 has triggered PRINTER IO
Process 3 entered the low priority queue
Process 3 has started executing
Process 3 has finished execution
Process 1 entered the low priority queue
Process 1 has started executing
Process 2 finished its IO
Process 1 has finished execution
Process 2 entered the high priority queue
Process 2 has started executing
Process 4 has finished execution
Process 2 has finished execution
```

A segunda forma de saída desse programa é através de uma *snapshot*, que detalha exatamente que tipo de processo cada estrutura do escalonador contém e onde estão os processos a cada intervalo de tempo. Para utilizar essa saída, deve-se adicionar o parâmetro `--steps` após a chamada do programa, depois do nome do arquivo *CSV*. A seguir, encontram-se 3 *snapshots* que descrevem três momentos do programa: início, meio e fim:

Início:

```
==================================================

  Time: 0

--------------------------------------------------

  Executing:
  P[1]: [(4, TAPE)(9, NONE)]

--------------------------------------------------

  High priority queue:
  ---

--------------------------------------------------

  Low priority queue:
  ---

--------------------------------------------------
```

```
  Doing I/O:

  Disk:
  ---

  Tape:
  ---

  Printer:
  ---

================================================

  [Press Enter to continue]
```

Meio:

```
================================================

  Time: 22

------------------------------------------------

  Executing:
  P[4]: [(4, PRINTER)]

------------------------------------------------

  High priority queue:
  ---

------------------------------------------------

  Low priority queue:

  (0).

  P[3]: [(1, NONE)]

  (1).

  P[1]: [(4, NONE)]

------------------------------------------------

  Doing I/O:

  Disk:
  ---

  Tape:
```

```
  P[2]: [(5, NONE)]     Remaining time: 6

  Printer:
  ---

================================================

  [Press Enter to continue]
```

Fim:

```
================================================

  Time: 37

------------------------------------------------

  Executing:
  ---

------------------------------------------------

  High priority queue:
  ---

------------------------------------------------

  Low priority queue:
  ---

------------------------------------------------

  Doing I/O:

  Disk:
  ---

  Tape:
  ---

  Printer:
  ---

================================================

  [Press Enter to continue]
```

Bibliografia

[1] Gustavo de Mendonça Freire, Vítor Amaral Bedin e Yuri Rocha de Albuquerque. *Process Scheduler*. GitHub repository. URL: `https://github.com/GustavoMF31/ProcessScheduler`.