Sistemas Operacionais – Trabalho Prático 1

Prof. Juliano Henrique Foleiss, M. Sc.

1 Introdução

A simulação é uma ferramenta extremamente importante no desenvolvimento de novas técnicas e tecnologias em todas as áreas da ciência e engenharia. Na computação a simulação é utilizada para ajudar a determinar a eficiência de novas técnicas em um ambiente controlado para prever a eficiência de algoritmos e procedimentos computacionais aplicados a resolver determinados problemas. A vantagem está no fato que a implementação de um simulador é geralmente mais simples que a implementação da técnica na prática, mas seus resultados permitem avaliar se o ganho esperado com a técnica supera seus esforços de implementação. Além disto, como a principal funcionalidade da simulação é coletar dados para a análise posterior, sua implementação não requer tanto cuidado com o desempenho da máquina quanto uma implementação real. Portanto a coleta de dados pode ser realizada de forma mais agressiva, revelando informações que não poderiam ser facilmente determinadas a partir de implementações reais pela interferência no desempenho.

No contexto de sistemas operacionais, a implementação de um escalonador de processos é uma tarefa complexa que exige a compreensão de muitos componentes do sistema e suas interações. Se o objetivo da implementação for compreender a viabilidade e o comportamento do escalonador em várias situações, a utilização de um simulador é mais adequada uma vez que muitos detalhes podem ser omitidos em uma simulação, como, por exemplo, as interfaces entre o sistema operacional e os subsistemas de E/S e gerenciamento de memória. Ao omitir estes detalhes, a implementação do simulador é muito mais rápida e simples (barata!), ao mesmo tempo que permite o estudo dos parâmetros de interesse.

O objetivo principal deste trabalho é a implementação e a análise de comportamento de algoritmos de escalonamento em um ambiente simulado uniprocessado. A seguir a especificação do simulador é apresentada, juntamente com as regras de entrega e apresentação. Um simulador básico funcional é fornecido como ponto de partida para o seu trabalho.

2 O Simulador

O principal objetivo do simulador a ser desenvolvido é computar algumas estatísticas para determinar a eficiência dos algoritmos de escalonamento estudados em ambientes em lote e interativos. Estas estatísticas estão explicadas na Seção 2.3. Lembre-se que o seu simulador deve simular todos os componentes pertinentes e coletar todas as informações necessárias para a computação destas estatísticas durante a simulação.

Para simplificar o trabalho de correção, os arquivos de entrada e saída do simulador devem estar formatados de acordo com esta especificação. Caso não estejam, a nota do trabalho corre o risco de ser anulada. Portanto, preste bastante atenção nos formatos que serão apresentados, juntamente com os exemplos anexos a este documento. Serão fornecidos arquivos de teste, no entanto não disponibilizarei os arquivos que serão utilizados na correção. Seja meticuloso! Esta é uma característica que todo cientista da computação deve prezar!

2.1 Entrada

Os experimentos que serão executados pelo simulador estão divididos em dois tipos de arquivos: **Arquivo de Experimento** e o **Arquivo de Processos**. O Arquivo de Experimento é o único argumento para o simulador. Este arquivo contém: qual algoritmo deve ser executado no experimento, quais são os parametros para o algoritmo (por exemplo, no algoritmo `Round-Robin` existe o *quantum*) e qual arquivo de processos deve ser utilizado no experimento. O Arquivo de Processos, por outro lado, contém informações pertinentes à execução dos processos como: a quantidade de processos, o tempo que cada processo é criado, o tempo em que acontecem bloqueios e desbloqueios, a prioridade de cada processo e o tempo que o processo é terminado. A seguir são apresentados os formatos destes arquivos.

Formato do Arquivo de Experimento

```
NOME_EXP
ARQ_PROCESSOS
ARQ_SAIDA
fcfs|sjf|rand|rr|fp
PARAM
```

Figura 1: Formato do Arquivo de Experimento

- `NOME_EXP` é uma string com o nome do experimento
- `ARQ_PROCESSOS` é o nome do arquivo de processos a ser utilizado neste experimento
- `ARQ_SAIDA` é o nome do arquivo que conterá a saída da simulação
- `fcfs|sjf|rand|rr|fp` é o algoritmo que deve ser usado
- `PARAM` são os parametros do algoritmo, que serão descritos a seguir

Para os algoritmos `fcfs`, `sjf` e `rand` não existem parametros para os algoritmos de escalonamento. Portanto, nestes casos, a linha `PARAM` deve conter a string "`NENHUM`". Para o algoritmo Round-Robin, o formato é apresentado a seguir.

```
QUANTUM
```

Figura 2: Parametros para o algoritmo `rr`

- `QUANTUM` indica quantos tempos dura o *quantum* de cada processo

Para o algoritmo de filas de prioridades (`fp`)o formato é apresentado a seguir:

```
NUM_FILAS
fcfs|sjf|rand|rr(quantum)
fcfs|sjf|rand|rr(quantum)
...
fcfs|sjf|rand|rr(quantum)
```

Figura 3: Parametros para o algoritmo `fp`

- `NUM_FILAS` indica quantas filas de prioridade existem. Este parametro é obrigatório mas nos experimentos que utilizaremos ele sempre vai ser 40.
- Cada linha depois da primeira indica qual algoritmo de escalonamento deve ser usado em cada uma das filas de prioridade correspondentes. Ou seja, o algoritmo selecionado na primeira linha é o algoritmo que vai ser usado no escalonamento dos processos da primeira fila de prioridades, e assim por diante. Note que o único que possui parâmetros é o `rr`, que, neste caso é a duração do *quantum* e que deve ser passado diretamente nesta linha.

A seguir segue um exemplo de um Arquivo de Experimento para o algoritmo `sjf`.

```
ex01
processos1.proc
sjf
NENHUM
```

Figura 4: Exemplo de Arquivo de Experimento para `sjf`

A seguir um exemplo de um Arquivo de Experimento para o algoritmo `rr`.

```
ex02
processos1.proc
rr
20
```

Figura 5: Exemplo de Arquivo de Experimento para `rr`

A seguir um exemplo de um Arquivo de Experimento para o algoritmo `fp` (encurtado por razões de espaço).

```
ex03
processos1.proc
fp
40
rr(50)
rr(50)
rr(50)
sjf
sjf
sjf
fcfs
fcfs
fcfs
rand
rand
rand
...
rand
```

Figura 6: Exemplo de Arquivo de Experimento para `fp`

Formato do Arquivo de Processos

Como explicado anteriormente, o Arquivo de Processos contém a descrição dos processos que serão escalonados, juntamente com uma fila de eventos que deverá ser processada de forma adequada.

```
NUM_PROCESSOS
FAIXA_PRIORIDADE
PID
PRIORIDADE
N_EVENTOS
T_ENTRADA
T1 EVENTO
T2 EVENTO
T3 EVENTO
...
TN TERMINO
PID
PRIORIDADE
N_EVENTOS
T_ENTRADA
T1 EVENTO
T2 EVENTO
T3 EVENTO
...
TN TERMINO
...
```

Figura 7: Formato do Arquivo de Processos

- `NUM_PROCESSOS` indica a quantidade de processos que estão descritos neste arquivo
- `FAIXA_PRIORIDADE` está relacionada a qual é a faixa de prioridades que podem ser atribuídas aos processos. Para este trabalho, a faixa de prioridades será sempre 1-40, tal que 1 é a maior prioridade e 40 a menor.
- `PID` é o id do processo. Note que a descrição de um processo sempre começa com uma linha PID e termina com um evento `TERMINO`.
- `PRIORIDADE` é a prioridade do processo. Este campo só é utilizado efetivamente quando o algoritmo `fp` é usado. No entanto, para fins de simplificação da especificação este campo está sempre presente neste arquivo.
- `N_EVENTOS` indica quantos eventos este processo terá. Este número inclui a quantidade de linhas inclusive e entre `T_ENTRADA` e `TN TERMINO`.
- `T_ENTRADA` é o tempo (real) que este processo é inserido no sistema, ou seja, este é o tempo, em relação ao relógio do simulador, que este processo é colocado pela primeira vez na lista de prontos.
- `T1 EVENTO ... TN TERMINO` denotam quais eventos acontecerão durante a execução deste processo. Estes eventos podem (e devem) ser:
  - `BLOQUEIO`: indica que este processo não deve ser escolhido para execução, ou seja, é colocado na lista de "bloqueados". Neste caso, o tempo é relativo a quantos "tempos" o processo já ocupou a CPU e indica quando este evento aconteceu.
  - `DESBLOQUEIO`: indica que este processo pode voltar a ser escolhido para execução, ou seja, é colocado na lista de "prontos". Neste caso, o tempo indica quantos "tempos" do relógio do simulador este processo deve ficar bloqueado. Este tempo NÃO INDICA em qual tempo do relógio este processo é desbloqueado! (isto nem faz sentido, certo?)
  - `TERMINO`: indica que este processo deve ser removido do sistema. O tempo mostrado é relativo a quantos "tempos" o processo já ocupou a CPU e indica quando este evento aconteceu.

A seguir segue um exemplo de um Arquivo de Processos com 2 processos.

```
2
1-40
555
1
6
0
50 BLOQUEIO
175 DESBLOQUEIO
490 BLOQUEIO
300 DESBLOQUEIO
5000 TERMINO
556
1
4
420
170 BLOQUEIO
120 DESBLOQUEIO
350 TERMINO
```

Figura 8: Exemplo de um Arquivo de Processos

2.2 Processamento

Conforme discutimos em sala, a função do escalonador de processos é escolher a sequencia que os processos prontos para a execução serão eleitos para ocuparem a CPU. De forma geral, em um determinado momento, um processo pode estar em um de três estados: pronto (aguardando ser escolhido para execução), bloqueado (aguardando o atendimento de uma requisição pelo SO) e executando (ocupando a CPU). A Figura 9 mostra as possíveis transições de estado que os processos podem realizar.

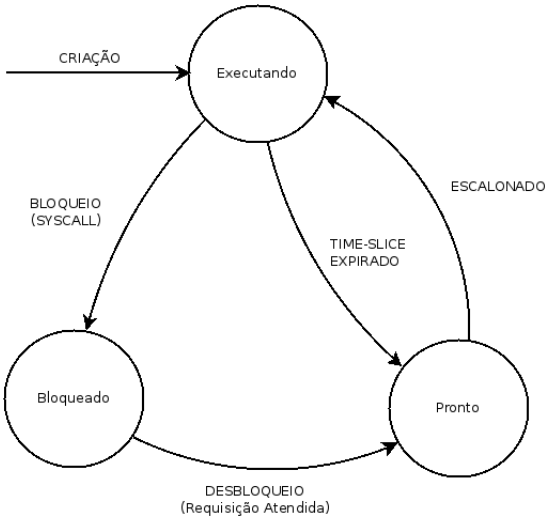

Figura 9: Estados de um Processo

O papel do escalonador é escolher um processo que está no estado `Pronto` e colocá-lo no estado `Executando`. Esta escolha é realizada com base em uma **política de escalonamento**. Estas políticas estão atreladas a **algoritmos de escalonamento** que podem utilizar diversos parâmetros para realizar a escolha de forma a otimizar alguma métrica de escalonamento de processos.

Ao invés de iterar por todos os descritores de processo, uma forma mais eficiente de realizar as computações da operação de escalonamento é a manutenção de listas que indicam quais processos estão em quais estados. Desta forma a implementação do escalonador é simplificada. Outro conceito importante nesta simulação é o conceito de tempo do simulador. Como referência, um relógio é mantido durante a simulação. Cada operação dura uma **unidade de**

tempo, ou, para encurtar, um **"tempo"**.

Para efeito de simplificação do simulador, e sem perda de detalhes significativa para efeitos de simulação, os processos não executam qualquer processamento. Desta forma, o **contexto** (conteúdo de registradores, estado da memória, etc) é simplificado e possui apenas um único dado: quantos "tempos" o processo já ocupou a CPU. Assim, quando um processo entra na CPU, o simulador deve contabilizar o tempo da ocupação e atualizar o BCP do processo com esta informação. Esta mesma informação pode ser usada para determinar se o processo atingiu algum evento de `BLOQUEIO` ou `TÉRMINO`.

No caso de eventos de `BLOQUEIO`, o processo é suspenso e colocado na lista de `BLOQUEADOS`. Imediatamente após um evento de bloqueio, sempre há um evento `DESBLOQUEIO` que indica quantos tempos de simulação o processo deve ocupar a lista de `BLOQUEADOS`. No caso de eventos de `TÉRMINO`, a execução do processo é suspenso e o mesmo é removido da lista de prontos. Para fins de simulação, a estrutura que representa um processo nestas listas é o equivalente ao BCP.

Em suma, para cada tempo da simulação, o simulador contabiliza o "tempo" como executado no processo atual, verifica se há eventos de `BLOQUEIO` a serem tratados. Após o tratamento de eventos de bloqueio, o simulador deve verificar se o `TIME-SLICE` (caso haja esta política no algoritmo atual) expirou. Além disto, o simulador também deve iterar pela lista de bloqueados para processar as requisições. Nota-se que, como este é um ambiente simulado, nenhuma requisição é de fato atendida. Neste caso, apenas é deduzido um tempo de cada uma das requisições pendentes. Caso algum tempo atinja 0, o processo é colocado novamente na lista de prontos. A escolha de um novo processo para ocupar a CPU deve acontecer toda vez que um processo executa um evento de `BLOQUEIO` ou `TÉRMINO` ou quando o `TIME-SLICE` do processo é esgotado.

Devem ser implementados os seguintes algoritmos de escalonamento de processos: `FCFS` (*First Come First Serve*), `SJF` (*Shortest Job First*), `Random`, `Round-Robin` e `Múltiplas Filas` (de prioridades). O funcionamento destes algoritmos deve ser realizado conforme apresentado em sala de aula, segundo o livro do Tanenbaum. A seguir são apresentadas algumas variações que devem ser incorporadas em cada um destes algoritmos.

FCFS

- No bloqueio há o chaveamento de processos, caso hajam processos prontos.
- Em um desbloqueio o processo mais antigo retoma a execução.

SJF

- No bloqueio há o chaveamento de processos, caso hajam processos prontos.
- O tempo restante é usado como parâmetro de escalonamento.
- No desbloqueio o SJF não necessariamente retoma a execução de outro processo, a não ser que o tempo restante do processo recém-desbloqueado é menor que o tempo restante do processo atual.

Random

O algoritmo `Random` não foi estudado em sala. No entanto, seu funcionamento é simples: um processo aleatório é escolhido entre os processos prontos pra serem executados. No bloqueio, o processo é colocado na lista de `BLOQUEADOS` e um outro processo aleatório é executado. No desbloqueio, o processo é simplesmente colocado na lista de processos prontos. Esta implementação não utiliza o conceito de `TIME-SLICE`.

Round-Robin

- O *quantum* é selecionável no Arquivo de Experimento.

Multiplas Filas

- Existem 40 classes de prioridade 1-40, onde 1 tem a maior prioridade e 40, a menor.
- Cada classe de prioridade é caracterizada por uma lista de processos com a mesma prioridade que devem ser escalonados de acordo com uma política de escalonamento selecionável no Arquivo de Experimento.
- Não há remanejamento de processos entre as filas.

2.3 Saída

A saída da simulação de cada experimento deve conter as seguintes informações:

- Número de chaveamentos de contexto realizados
- Tempo médio de espera. Calcula-se da mesma forma que calculamos em sala de aula.
- Tempo médio de retorno. É a média aritmética da diferença entre o tempo que a última e a primeira instrução foi executada de cada processo.
- Vazão. Quantidade de processos concluídos a cada 1000 unidades de tempo.
- Sequência de término dos processos.
- Diagrama de Execução.

A saída deve ser escrita em um arquivo (cujo nome foi passado como parâmetro no Arquivo de Experimento) e deve estar no formato especificado a seguir.

```
CHAVEAMENTOS: ZZZ
TME: ZZZ
TMR: ZZZ
VAZAO: ZZZ
TERMINO: PID PID PID PID PID PID
DIAGRAMA_DE_EXEC
```

Figura 10: Formato de Saída da Simulação

O Diagrama de Execução deve ter o seguinte formato:

```
DIAGRAMA DE EVENTOS
TTT PID EVENTO
TTT PID EVENTO
TTT PID EVENTO
...
TTT PID EVENTO
```

Figura 11: Formato do Diagrama de Execução

- `TTT` é o tempo, relativo ao tempo total da simulação (relógio da simulação) que o evento aconteceu
- `PID` é o `ID` do processo ao qual o evento está relacionado
- `EVENTO` é um dos eventos a seguir:
  - `CRIAÇÃO`: indica a criação (introdução) do processo no sistema.
  - `TERMINO`: indica o término do processo.
  - `BLOQUEIO`: indica o bloqueio do processo no sistema.
  - `DESBLOQUEIO`: indica o desbloqueio do processo.
  - `QUANTUM_EX`: indica o momento que o quantum do processo expirou.
  - `EXEC`: indica que o processo foi colocado em execução (foi escalonado naquele momento)

A seguir é apresentada a saída da simulação do exemplo da execução do algoritmo `Round-Robin` que consta no material da disciplina.

```
CHAVEAMENTOS: 8
TME: 11
TMR: 17
VAZAO: 3
TERMINO: 3 2 1
DIAGRAMA DE EVENTOS
0 1 CRIAÇÃO
0 1 EXEC
4 1 QUANTUM_EX
4 2 CRIAÇÃO
4 2 EXEC
8 2 QUANTUM_EX
8 3 CRIAÇÃO
8 3 EXEC
11 3 TERMINO
11 1 EXEC
15 1 QUANTUM_EX
15 2 EXEC
18 2 TERMINO
18 1 EXEC
22 1 QUANTUM_E
22 1 EXEC
26 1 QUANTUM_EX
26 1 EXEC
30 1 QUANTUM_EX
30 1 EXEC
34 1 QUANTUM_EX
34 1 TERMINO
```

Figura 12: Exemplo de Saída

3 Regras de Execução / Entrega

Simulador Baseline

Para que o trabalho possa se concentrar nas partes relacionadas às políticas de escalonamento, um simulador funcional (que já possui a política round-robin implementada) é oferecido. Algumas dicas sobre o simulador:

- Já estão implementadas rotinas para ler os arquivos de experimento e os arquivos de processo.
- O arquivo `simulador.c` contém o núcleo da simulação, que é o mesmo utilizado para todas as políticas. As rotinas `escalonar`, `tick`, `novoProcesso`, `fimProcesso` e `desbloqueado` são conhecidas como *callbacks*. Estas rotinas representam pontos no programa onde as políticas se diferenciam umas das outras.
- Caso haja necessidade de usar outros callbacks, fique à vontade para criá-los.
- O arquivo `politicas.c` contém as implementações destes callbacks. Estude o código para descobrir como inserir outras políticas.
- O arquivo `politicas.h` contém a definição das estruturas das políticas. Estude bem este arquivo para entender como manipular as estruturas relativas às políticas de escalonamento corretamente.
- Os demais arquivos possuem código de apoio, que não necessitam ser estudados em tanto detalhe. No entanto, estudar o programa todo serve como um bom ponto de partida para entender como modularizar e como gerenciar programas maiores escritos na linguagem C. Não percam a oportunidade de amadurecer-se como desenvolvedores!
- O simulador baseline ainda não oferece suporte para escrever o arquivo de saída! Inclua esta funcionalidade na sua versão do programa.

- ESTE SIMULADOR PODE SER USADO E ALTERADO NO DESENVOLVIMENTO DO SEU TRABALHO.
- O simulador somente está disponível em C.

Entregas

Dois artefatos devem ser entregues:

- O código-fonte do simulador.
- Um relatório de implementação.

Um relatório de implementação? Sim! Este relatório deve ser um documento que explica como você realizou a implementação do simulador: como vocês dividiram o trabalho entre os dois integrantes da equipe? Quais foram os módulos que vocês criaram? Qual é a interface entre os módulos? Qual estratégia foi utilizada para evitar uma implementação diferente do simulador para cada algoritmo implementado? Utilize diagramas e explicações que respondam estes questionamentos. Seu relatório deve ter, no mínimo, 8 páginas e deve estar no formato de artigos da IEEE (disponível em: http://www.ieee.org/conferences_events/conferences/publishing/templates.html).

A implementação vale 50% da nota, o relatório vale 20% e a apresentação do Simulador vale 30%.

Regras

Respeite as regras a seguir sob pena de anulação da nota do trabalho.

- Este trabalho deve ser feito exclusivamente em duplas.
- O trabalho pode ser feito em `C`, `C++` ou `Java`.
- Trabalhos plagiados de outros alunos ou de outros trabalhos na internet receberão nota 0,0 (zero vírgula zero).
- O trabalho deve ser entregue, impreterivelmente, até dia 24/5 via moodle. Não serão aceitos trabalhos enviados após esta data.
- Somente o código-fonte do trabalho e o relatório de implementação devem ser entregues.
- É possível fazer este trabalho sem usar biblioteca não-padrão. Não utilize bibliotecas não-padrão.
- Em projetos `C`/`C++` utilize um Makefile.
- Utilize os formatos de arquivo de entrada e saída especificados neste documento! Caso contrário, a nota do simulador será ANULADA.

Dicas

- A utilização (correta) do paradigma de programação orientado a objetos facilita grandemente a implementação do simulador. Em específico, esta abordagem elimina a necessidade de implementar um núcleo do simulador diferente para cada algoritmo de escalonamento de processos que for implementado.
- Estude cautelosamente o simulador dado como ponto de partida (baseline). Este simulador possui um ótimo exemplo da dica acima. Note a forma como cada política pode ser implementada de forma independente das demais, com mínimas alterações ao núcleo do simulador.
- O programa não precisa de interface gráfica. É esperado que seja possível passar um Arquivo de Experimento para o simulador e que ele gere um Arquivo de Saída conforme especificado neste documento. A criação de uma interface gráfica apenas daria trabalho extra.