Sistemas Operativos

Licenciatura em Engenharia Informática

Trabalho Prático 1

Simulador Modelo de 5 Estados

UNIVERSIDADE DE ÉVORA

Miguel Pombeiro, 57829 | Miguel Rocha, 58501

Departamento de Informática
Universidade de Évora
Abril 2025

Índice

1 Introdução

Este relatório descreve a implementação, em linguagem C, de um programa que simula o funcionamento de um Sistema Operativo que utiliza o modelo de cinco estados: `NEW`, `READY`, `RUNNING`, `BLOCKED`, `EXIT`. Este simulador recebe como *input* uma matriz de valores inteiros, que representam pseudo-instruções, organizadas em diferentes programas. O simulador implementa um escalonador com algoritmo *Round Robin*, com um *time quantum* de 3 instantes. Esta simulação corre durante 100 instantes de tempo, guardando o estado de cada processo ativo por cada instante, de forma a seguir o seu desenvolvimento.

2 Estrutura do Programa

Para que este simulador funcione de forma semelhante ao que acontece nos Sistemas Operativos reais, é necessário identificar os componentes essenciais que é necessário implementar. Assim, foi implementado um escalonador, que permite escalonar os vários processos que estão em execução e do qual fazem parte várias estruturas de dados como filas e listas, de forma a representar a estado atual de cada processo. Os processos, estão representados através de alguns elementos que fariam parte do seu PCB (*Process Control Block*) e que serão descritos nesta secção.

2.1 Estruturas

Cada processo é representado por uma estrutura que inclui:

- **ID**: identificador único, atribuído incrementalmente a partir de 1;
- **Estado**: o estado atual do processo;
- **currentInstruction**: índice da instrução atual;
- **Programa**: vetor de instruções do programa a que corresponde o processo;
- **nInstructions**: número de instruções do programa;
- **time**: contador de instantes, para gerir a permanência nos estados `NEW`, `RUNNING`, `BLOCKED` e `EXIT`.

O escalonador (*scheduler*) é representado pela seguinte estrutura:

- **Lista NEW**: Representa o estado `NEW`. Quando os processos são criados devem estar nesta lista dois instantes de CPU;
- **Fila READY**: Representa o estado `READY`. Contém os processos que estão à espera para serem executados;

- **runningProcess**: Processo que está no estado `RUN`;
- **Lista BLOCKED**: Representa o estado `BLOCKED`. O tempo de permanência de processo nesta lista corresponde ao módulo do valor da instrução I/O executada;
- **Lista EXIT**: Lista na qual os processos passam um instante de CPU antes de serem terminados. Apesar de só existir um processo a sair por instante, esta implementação foi necessária devido à possível entrada de um processo no estado `EXIT` antes de um processo anterior a este ter saído deste estado;
- **numProcesses**: Número de processos atuais do *scheduler*;
- **processes**: *Array* que contém todos os processos;
- **instructions**: Matriz que contém todas as instruções por programa, dados no *input*;
- **nInstructionsProgram**: Número de instruções por programa;
- **nPrograms**: Número de programas dados no *input*;

2.2 Estados

Foram implementados 6 estados diferentes, cada um representado por um inteiro de 0-5 sendo estes, respetivamente:

- **NEW**;
- **READY**;
- **RUN**;
- **BLOCKED**;
- **EXIT**;
- **TERMINATED**;

Apesar de o modelo apenas ter 5 estados, foi adicionado o estado `TERMINATED` para esta implementação em C, devido à necessidade de manter os processos guardados após o final da execução para realizar a impressão correta dos resultados.

2.3 Funções

Tendo, o simulador, sido dividido em duas componentes principais, foram implementadas funções que as permitem manipular. Ao nível do escalonador, as funções podem ser divididas em dois tipos principais diferentes:

- Funções que permitem manipular os estados dos processos em execução e alterar estrutura de dados do escalonador, em que estão representados. Os nomes destas funções são representativos das várias operações do modelo de cinco estados, *e.g.* `schedulerNew`, `schedulerBlock`, `schedulerRelease`.

- Funções que permitem decrementar, para todos os processos num determinado estado, o tempo que estes ainda lá devem permanecer, *e.g.* `timeDecrementRunning`, `timeDecrementNew`.

Ao nível dos processos, podem ser destacadas duas funções principais:

- `fetchNextInstruction`, que permite ler a próxima instrução a executar.

- `changeCurrentInstruction`, que permite alterar o *program counter* do processo (neste caso, o índice da instrução atual no vetor de instruções), de forma a implementar as instruções do tipo `JUMP`.

Por fim, foram ainda implementadas algumas funções que simulam a interpretação e execução das instruções dos programas, de acordo com as orientações do enunciado, respetivamente `getInstructionType` e `executeInstruction`.

2.4 Execução

O programa começa por ler um *input* a partir dos argumentos da linha de comandos (`argv[1]`), que indica qual das matrizes de programas, presentes no ficheiro de *inputs* fornecido, deverá ser utilizada. Após escolhida a matriz de programas, que contém as instruções, esta é processada para que cada programa fique representado numa linha (e não numa coluna) da matriz, de forma a tornar os posteriores acessos mais rápidos.

Cada tipo de instrução tem uma função diferente, podendo esta fazer um salto de `n` instruções para trás (`JUMP`), bloquear um processo por n instantes (`IO`), lançar um novo processo que executa um certo programa (`EXEC`) ou até terminar a execução do processo (`HALT`).

Antes de se iniciar a simulação da execução, é inicializado o *scheduler* e todas as estruturas de dados a ele associadas. Esta simulação irá correr pelo máximo de 100 instantes. Por cada instante de tempo, são realizados os seguintes passos:

- ***Dispatch*** - Se não existe nenhum programa no estado `RUN` e a fila `READY` não está vazia, então um processo é movido do topo da fila e começa a correr.

- **Execução da Instrução** - Se um processo está a correr, a próxima instrução é lida do vetor de instruções e executada de acordo com o seu tipo (`HALT`, `JUMP`, `EXEC`, `I/O`, `ANY` - instrução normal).
- **Decrementar os tempos** - Uma vez que os processos devem passar um número de instantes especifico no estado em que se encontram, com a exceção o estado `READY`, então, após a execução da instrução, todos os processos nos estados `NEW`, `RUN`, `BLOCKED` e `EXIT` têm os seus tempos de permanência decrementados de um instante. Quando o tempo de um processo chega a 0, este passa para o respetivo próximo estado, de acordo com o modelo de cinco estados.
- **Impressão do estado** - Todos os processos em execução têm o seu estado impresso para o *output*.

A simulação da execução do programa é realizada pela função `simulate`, que começa pela atribuição do programa 1 ao primeiro processo, no instante 1. Nessa atribuição é feita a entrada desse processo na lista `NEW`, onde deverá passar dois instantes. Depois de passar dois instantes, é então possível a leitura das instruções. Essas instruções são executadas de acordo com o seu tipo, durante um instante.

A simulação termina quando transcorridos o número máximo de 100 instantes, sendo tanto o *scheduler* e respetivas estruturas de dados, bem como a lista de programas gerada, destruídos.

3 Comentários

O programa deve ser executado de acordo com o `README` que acompanha o código fonte, nomeadamente através do `makefile` fornecido, de forma a que os ficheiros de *output* tenham a denominação e o formato pedidos.