

# Sistemas Operativos

Relatório do 2º trabalho prático - Simulador de SO com modelo de 5 estados

2022/2023

Tomás Antunes (nº48511), Luís Simões (nº48726)



# Índice

|  |          |
|--|----------|
| <b>Índice.....</b>                       | <b>2</b> |
| <b>Introdução.....</b>                   | <b>3</b> |
| <b>Características do Simulador.....</b> | <b>3</b> |
| <b>Desenvolvimento do Simulador.....</b> | <b>4</b> |
| Estruturas.....                          | 4        |
| Estados.....                             | 4        |
| Filas.....                               | 4        |
| Mudanças de Estado.....                  | 4        |
| Criação dos Programas.....               | 4        |
| Execução dos programas.....              | 5        |
| Main.....                                | 5        |
| • Variáveis.....                         | 5        |
| • Funções.....                           | 5        |
| <b>Exemplo de Input/Output.....</b>      | <b>6</b> |
| Input.....                               | 6        |
| Output.....                              | 6        |
| <b>Conclusão.....</b>                    | <b>7</b> |

# Introdução

Neste trabalho, pretendemos implementar, usando a linguagem C, um simulador de um Sistema Operativo considerando um modelo de 5 estados que inclui os estados READY, RUNNING, BLOCKED, e ainda os estados NEW e EXIT.

Diferente do trabalho anterior, neste trabalho adicionamos também a instrução UNBLOCK. A instrução UNBLOCK, quando existir, corre sempre em paralelo com o primeiro instante em RUNNING do processo que dá a ordem de desbloqueio. Quando um processo recebe uma ordem para ser desbloqueado, este deve ser desbloqueado no próximo instante de tempo, mesmo que ainda tenha instantes no estado BLOCKED.

## Características do Simulador

A implementação do simulador de processos deve ter em consideração as seguintes características:

1. Admite-se que a mudança entre estados é infinitamente rápida; depois os processos ficam nos diversos estados 1 instante de tempo, seguindo-se outra mudança de estados, e assim sucessivamente.
2. Os processos quando são criados passam para o estado NEW e permanecem nesse estado 1 instante de tempo. Note que mais do que um processo iniciar no mesmo instante, ficarão vários processos no estado NEW, durante um instante de tempo, e todos eles passarão para o estado READY no instante seguinte.
3. Os processos quando saem de NEW passam para o estado READY, respeitando a ordem pela qual entraram em NEW. No entanto, se a fila de READY estiver vazia, e o CPU (estado RUNNING) não estiver ocupado um processo pode passar de NEW para READY e deste para RUNNING/CPU “instantaneamente”
4. Os instantes na fila BLOCKED, para cada processo, contam assim que o processo chega à fila.
5. Os processos quando terminam, passam do CPU (estado RUNNING) para o EXIT, onde permanecem 1 instante de tempo.
6. Os processos depois de estarem 1 instante de tempo em EXIT desaparecem do sistema.
7. O escalonamento a implementar deve ser o algoritmo FCFS (first-in first-out).
8. Os processos que saem de BLOCKED, passam para READY.
9. O número máximo de programas a dar entrada no sistema é de 8, e cada programa tem a dimensão máxima de 8 (1 instante inicial, mais 7 instruções alternadas de CPU e BLOCK).
10. Se no mesmo instante puderem entrar vários processos na fila de READY vindos de RUNNING, BLOCKED, e NEW, estes entrarão na fila de READY pela seguinte ordem: 1º os de BLOCKED; 2ª o de RUNNING/CPU; e por último os de NEW.
11. O programa deve ter como output, o estado de cada processo em cada instante.
12. Um processo no estado RUNNING, poderá desbloquear outro processo, se este se encontrar no estado BLOCKED, passando no próximo instante para o estado READY.

# Desenvolvimento do Simulador

## Estruturas

É utilizada uma estrutura chamada *Program*, que representa cada programa a ser executado pelo simulador. Esta estrutura é constituída por:

- *int start* → instante em que o programa é iniciado.
- *int exec* → processo a ser executado.
- *int state* → estado do programa.
- *int\* time* → arrays dos processos do programa.
- *int unblockTime* → inteiro que detêm o número de instantes que o processo esteve no estado UNBLOCKED

## Estados

Os estados são representados por inteiros:

- 1 → Processo não criado
- 0 → NEW
- 1 → READY
- 2 → RUN
- 3 → BLOCKED
- 4 → EXIT
- 5 → Processo terminado
- 6 → UNBLOCK

## Filas

São utilizadas três filas do tipo FIFO, para representar a fila READY, a fila BLOCKED, e a fila UNBLOCKED:

- Queue R → fila dos processos no estado READY
- Queue B → fila dos processos no estado BLOCKED
- Queue UBL → fila dos processos no estado UNBLOCKED

Cada programa nestas filas é representado por um inteiro, correspondendo ao índice da linha da matriz de programas fornecida.

## Mudanças de Estado

As mudanças de estado ocorrem na função *changeState*. Esta função irá verificar quais os programas que estão prontos para mudar de estado, fazendo assim, a troca do estado desses programas. Também controla as entradas e as saídas dos programas nas filas de READY, BLOCKED e UNBLOCKED.

## Criação dos Programas

A criação dos programas ocorre na função *createPrograms* que recebe uma matriz de programas como argumento. Esta função irá inicializar as filas de READY e BLOCKED, e de seguida irá processar a matriz fornecida, de modo a criar uma instância da estrutura *Program* para cada programa. Todos os programas irão ser inicializados com o estado -1 que significa que os programas ainda não foram criados.

## Execução dos programas

A execução dos programas ocorre na função **run**. É nesta função que se encontra o loop principal que irá executar todos os programas, mostrando o seu estado em cada instante.

### Main

- **Variáveis**

As variáveis utilizadas no nosso simulador são as seguintes:

- **programs[NUMPROGRAMS + 1]** → array de estruturas do tipo Program, que representa os programas/processos a serem simulados, contendo informações sobre o tempo de execução de cada processo e o seu estado atual
- **R** → uma fila para os processos READY, que guarda os programas que estão prontos a serem executados
- **B** → uma fila para os processos BLOCKED, que guarda os programas bloqueados e que aguardam algum evento para poderem ser executados
- **UBL** → uma fila para os processos que irão passar para o estado UNBLOCKED.
- **instant** → contador de tempo que representa a unidade de tempo atual da simulação
- **running** → inteiro que representa o ID do programa em execução, ou o valor **'NONE'** se não houver nenhum em execução

- **Funções**

As funções utilizadas no nosso simulador são as seguintes:

- **notFinished()** → função que verifica se ainda existem programas que não foram finalizados, ou seja, que têm estado de execução menor que 4 (EXIT). Retorna 1 se houver programas não finalizados e 0 caso contrário;
- **changeState(int pid)** → função que muda o estado de um programa com o ID **'pid'**, de acordo com as regras de escalonamento implementadas. O estado do programa pode ser alterado de NOTCREATED para NEW, de NEW para READY, de READY para RUN, de RUN para BLOCKED, de BLOCKED para READY e de EXIT para FINISHED, dependendo de várias condições;
- **unlockPro()** → função que muda o estado para UNBLOCKED dos processos que se encontrarem na Queue UBL (UNBLOCKED), se estes estiverem atualmente no estado BLOCKED.
- **showState(int pid)** → função que mostra o estado atual de um programa com o ID **'pid'**, de acordo com o estado definido na enumeração de estados;
- **run()** → função principal que simula o escalonamento dos processos. Ela chama as funções **'changeState()'** e **'showState()'** para cada programa em cada unidade de tempo, atualiza o contador de tempo, exhibe o estado dos programas e verifica se algum programa foi finalizado;
- **createPrograms(int p[NUMPROGRAMS][NUMPROCESS])** → função que cria os programas a serem simulados, inicializando a estrutura de dados **'programs'** com os tempos de execução de cada processo de cada programa, de acordo com a matriz **'p'** fornecida como argumento;
- **main()** → função principal do programa, que serve como ponto de entrada e saída para a execução do código e onde é feita a leitura do input de um ficheiro;

## Exemplo de Input/Output

### Input

```
int programas[3][10] = {
    {0, 3, -1, 2, 2, 3, 4, 2, 2, 0},
    {1, 4, 1, 4, 5, 1, 4, 3, 3, 0},
    {2, 2, 2, 6, 2, -1, 4, 2, 0, 0}
};
```

### Output

| i     | p01 | p02 | p03 |
|-------|-----|-----|-----|
| 0     | NEW |     |     |
| 1     | RUN | NEW |     |
| 2     | RUN | RDY | NEW |
| 3     | RUN | RDY | RDY |
| 4     | BLK | RUN | RDY |
| 5     | UBL | RUN | RDY |
| 6     | RDY | RUN | RDY |
| 7     | RDY | RUN | RDY |
| 8     | RDY | BLK | RUN |
| 9     | RDY | UBL | RUN |
| 10    | RUN | RDY | BLK |
| 11    | RUN | RDY | UBL |
| 12    | BLK | RUN | RDY |
| 13    | UBL | RUN | RDY |
| 14    | RDY | RUN | RDY |
| 15    | RDY | RUN | RDY |
| 16    | RDY | RUN | RDY |
| 17    | RDY | BLK | RUN |
| 18    | RDY | BLK | RUN |
| 19    | RUN | BLK | BLK |
| 20    | RUN | UBL | BLK |
| 21    | EXI | RUN | BLK |
| 22    |     | RUN | UBL |
| 23    |     | RUN | RDY |
| 24    |     | EXI | RUN |
| 25    |     |     | RUN |
| 26    |     |     | EXI |
| 27    |     |     |     |
| DONE. |     |     |     |

## **Conclusão**

A resolução deste trabalho permitiu-nos solidificar melhor a forma como um modelo de 5 estados de um SO funciona e desafiar-nos a implementar uma nova instrução no modelo concebido para o primeiro trabalho. Além disso, ajudou-nos a desenvolver continuamente as nossas capacidades de resolução de problemas em C.