Sistemas Operativos I

Departamento de Informática

Sistema Operativo

Ano Letivo 2018/2019

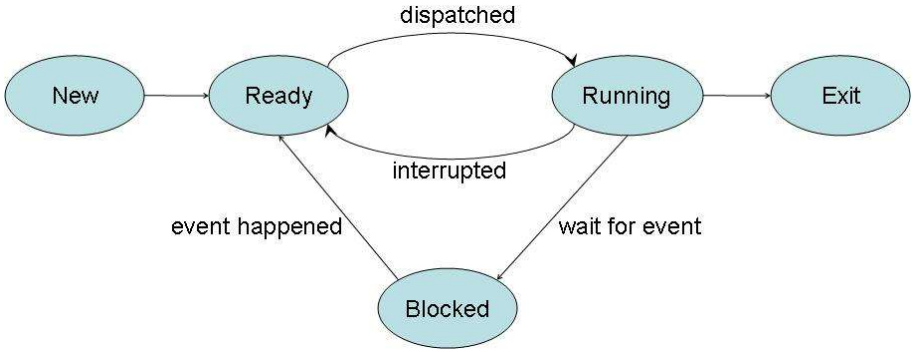
Este trabalho tem como finalidade a implementação de uma representação parcial e muito simplificada do funcionamento e complexidade de um Sistema Operativo que se conhece dos dias de hoje.

Ana Silvério nº 37561

João Queimado nº 38176

1. Introdução

Antes de qualquer implementação física do pretendido com a avaliação prática, é necessário ter presente algumas noções e conhecimentos adquiridos durante os momentos de avaliação mais teóricos. Para esta atividade, suponha-se que se tem uma arquitetura que recai/aplica o modelo de 5 estados. Quer isto dizer que, tem-se disponível o estado: NEW, READY/WAIT, RUN, EXIT e BLOKED. Como o representado na figura.



Esta arquitetura tem um consumo de programas constituídos por um conjunto de instruções. Instruções estas que são codificadas por um conjunto de 3 números inteiros. A sua codificação é nos fornecida sob a forma de uma tabela.

Codificação	Instruções	Significado
0, X1, X2	SET_X	Var X1 = var X2
1, X, N	SET_N	Var X = N
2, X, qq	INC_X	Var X = Var X + 1
3, X, qq	DEC_X	Var X = Var X - 1
4, N, qq	BACK_N	Salta para trás, PC -= N, em que N é logo o valor do salto, não se vai consultar o valor da variável
5, N, qq	FORW_N	Salta para a frente, PC += N, em que N é logo o valor do salto, não se vai consultar o valor da variável
6, X, N	IF_X_N	IF X ==0, salta N linhas para a frente PC+=N; ELSE vai para próxima linha PC++
7, X, qq	FORK X	X = Fork(), X é zero ser for o filho, ou o PID do processo criado se for o pai
8, X, qq	DISK_SAVE_X	Guarda a variável X no disco
9, X, qq	DISK_LOAD_X	Carrega a variável do disco para X
10, X, qq	PRINT_X	Imprime variável X
11, qq, qq,	EXIT	Termina

Exemplo:

```
X1 = 3
IF X1 ==0  SALTA 4
PRINT X1
X1=X1-1
BACK 3
EXIT
```

```
 1, 1, 3,
 6, 1, 4,
10, 1, 0,
 3, 1, 0,
 4, 3, 0,
11, 0, 0
```

Deste modo o programa seria representado por:
1 1 3 6 1 4 10 1 0 3 1 0 4 3 0 11 0 0

Deve ser apresentado e implementado um escalonamento *Round Robin* com um *Quantum* de 4 mas, pode ser configurável. Acerca dos processos recebidos, quando no mesmo instante, um processo vindo do NEW, do BLOCKED, e /ou do RUN querem entrar no READY, o vindo do BLOCKED tem prioridade, seguido do de RUN, e por fim o processo vindo de NEW. A gestão de memória deve ser implementada e definida com uma variável de modo a acomodar 2 modos dessa gestão: BEST FIT e NEXT FIT. O output ou resultado final desta avaliação prática são dois ficheiros de escalonamento. Um simples, que apenas contem os instantes e estados de cada processo, e um complexo, incorpora o simples e o print da memória sempre que um novo processo é colocado ou removido da memória.

```
. . .
9  | exit | exit | ready | block  |  run  | block | block | new   |
10 |      |      |  run  | ready  | exit  | block | block | ready |
11 |      |      | ready |  run   |       | block | block | ready | new
. . .
```

```
"instante 10:  READY |        | READY  | BLOCKED | RUN   | EXIT |       | 1234  "
```

2. Desenvolvimento

No desenvolvimento e implementação da 1ª parte deste trabalho muitos foram os erros cometidos e partes deixadas por desenvolver. Assim, uma parte desta 2ª parte, foi dedicada ao seu *upgrade* e melhoramento. Tanto funcional como estético.

1. Função read_file:

Os processos continuam a ser carregados para uma Queue. A função **read_file** vai colocar os instantes referentes a cada processo nessa estrutura de dados Queue (**Queue* pre_process**) já com a sua conversão feita. Isto é, lê os instantes do ficheiro **fname**, variável definida como global, que são do tipo string e converte-os para inteiros. Isto tudo apenas se o ficheiro de input não se encontrar vazio. Esta função tem como **Argumentos**: o nome do ficheiro de input com os instantes de cada processo (**fname**), definido como variável global dado que pode ser alterado, e a Queue para onde vão ser passados os processos (**Queue* pre_process**). É de mencionar ainda que a função é do tipo boolean, uma vez que o seu valor de retorno é **false**, se o ficheiro não contiver nenhum dado de input, e **true**, se o oposto ocorrer.

2. Função CPU:

Passando para a organização e implementação das instruções mencionadas na introdução, foram definidas variáveis globais para cada uma das instruções codificadas. A função **CPU** do tipo int recebe como **argumentos** a "Classe" onde os processos estão definidos (**Process * process**), o array designado para a representação da memória (**MEM**), e os *file pointers* dos ficheiros de escalonamento de output (**simple_file** e **comp_file**). A cada chamada da função 3 variáveis são inicializadas, uma vez que as instruções são codificadas por conjuntos de 3 em 3 números. Quer isto dizer, de 3 em 3 instantes. As instruções foram implementadas de acordo com as especificações estipuladas pela tabela mencionada na introdução. Existe uma condição para cada instrução e o valor de retorno é um código representado por um número inteiro. Código esse que representa ações.

3. Função mem_printer:

A função **mem_printer**, tal como o nome indica, vai imprimir os valores dos instantes que se encontram guardados em memória num ficheiro. Ficheiro este que será, nada mais nada menos que o ficheiro correspondente ao escalonamento de output complexo. Daí a função ser do tipo void. Os seus argumentos são o *file pointer* desse ficheiro (**comp_file**) e um array representativo da memória, **Memory**.

4. Função gc:

Outra função foi criada, e esta tem como objetivo a "limpeza" da memoria. A função **gc**, diminutivo para *garbage colector*, vai "limpar" a memória de processos considerados desnecessários. Isto é, processos que passaram para o estado BLOCKED porque, necessitavam da ocorrência de um evento ou I/O que nunca chegam. Após terminado o ciclo de tempo de espera, é tempo de se dar *unload* desses processos da memória.

5. Função main:

Como conclusão das funções utilizadas na realização deste trabalho, tem-se a função **main** onde tudo se vai processar. É aqui que, os ficheiros são acedidos e escritos para dentro destes. É aqui que se encontra a implementação do escalonamento Round Robin pedida na 1ª parte, em que foi definida uma variável global QUANTUM para o valor do quantum do escalonamento Round Robin visto este valor ser variável. É no corpo desta função que é aplicada as transições de estados bem como o seu desenvolvimento destes. Os processos recebidos andam a transitar de estados para estados até se dar o caso de terminarem e saírem pela transição RUN -> EXIT. É também aqui que se dá o "arranjo" da memória de acordo com mencionado na Introdução. Cada transição está identificação e assinalada. Tal como todos os acessos ao disco e á memoria. Na gestão da memória, foi criada uma variável global que tem uma codificação: 0 representa o BEST FIT e 1 representa o NEXT FIT. No final, é feito o print diretamente para os ficheiros de output pedidos respetivamente e identificados e libertado o espaço ocupado pelos acessos a estes e aos estados que entram em contacto com o disco e memória.

3. Outras Implementações

1. BEST FIT e NEXT FIT: Gestão da Memória

Foram implementadas Frames para cada um dos modos de gestão de memória pedidos. A frame do BEST FIT vai escolher o bloco de memória disponível que é mais semelhante, em tamanho ao tamanho do processo que se pretende armazenar em memória. Tentando, assim, achar um espaço mais próximo do espaço que o processo vai ter que ocupar. Conseguindo, assim, que o espaço que sobra entre o espaço que é utilizado pelo processo e o espaço que este tem disponível para a sua utilização seja o mais pequeno possível.

Seguindo um rumo diferente de implementação, a frame do NEXT FIT tem um apontador que vai servir que indicar qual o último espaço de memória utilizado para o armazenamento de um processo e vai iniciar a procura por um novo espaço livre desse apontador para cima. Ou seja, o apontador inicia no zero a partir dessa frame acedida anteriormente. A frame vai tornar os processos mais espaçosos entre si na memoria, criando uma memória uniforme em termos de espaço.

2. Disk

Neste ficheiro tem-se a implementação e inicialização do Disco, bem como das funções de **set_disk** e **get_disk** para a guardar e aceder a valores guardados em disco.

3. Pre_PRocess e Process

Os processos antes serem processos e irem para a Queue para se dar início ao seu escalonamento, têm de ser tratados e averiguar os seus tempos de chegada. Guardando sempre estes últimos. Depois de serem bem identificados e de os seus tempos de chegada coincidirem, passam a ser Processos.

É depois em Processos que se vai buscar todos os instantes de serviço de cada processo respetivamente e estabelecer/set os estados de cada processo. Os processos estão prontos a serem carregados para o disco, para a memória ou serem executados.

4. Queue

Na Queue é onde se encontram os processos para as transições de estados durante a execução do escalonamento Round Robin. Sabendo sempre quais os precessos a retirar da queue, uma vez que os processos entram na fila pela ondem dos seus tempos de chegada.