UNIVERSIDADE DE ÉVORA

Trabalho prático de Sistema Operativos 1

Yaroslav Kolodiy nº 39859 Eduardo Medeiros nº 39873

Ano Letivo 2018/2019

Sistemas Operativos 1, Docente: Luís Rato

1 Introdução

É pretendida a implementação de um sistema operativo com uma arquitetura de 5 estados cujas instruções são codificadas com 3 números inteiros.

O sistema operativo deve executar as seguintes instruções.

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

2 Breve Resumo

O sistema operativo é constituído por uma memória com 300 posições onde vão estar presentes o código dos processos e as suas variáveis.

Para executar as instruções pedidas o sistemas irá aceder à memória para as ler e executá-las.

3 Funcionamento

O sistema operativo é constituído por uma memória com 300 posições. Para cada processo introduzido, se possível, será introduzido na memória o seu código complementado com mais 10 espaços da mesma que vão ser usados para armazenar variáveis deste.

O sistema guarda a posição da memória onde o processo foi guardado em memória. Quando for o instante de este ser executado acederá à memória nessa posição ou na posição onde o processo interrompeu a sua execução previamente. Caso o processo não tenha mais nada para correr é removido do sistema.

Os processos mudam entre os 5 estados implementados. Caso haja espaço suficiente na fila, o precesso passará de *NEW* para *WAIT* onde apenas permanecerá caso haja um processo em *RUN*, caso contrário muda. Se não forem executadas instruções de *BLOCK* ou de *EXIT*, o processo passará 4 instantes em *RUN*. Se for chamada uma instrução que mude o processo para *BLOCK* este ficará no memso 3 instantes; por outro lado se for chamada a instrução que o muda para *EXIT* este ficará um instante em *EXIT* antes de ser removido do sistema.

4 Descrição das Funções

novoPcb: Esta função cria um novo PCB para o pid pretendido.

novoProcesso: Esta função cria um novo Processo com o PCB do pid pretendido.

obterPosicao: Dependendo do FIT pretendido esta função calcula a posição mais adquada a partir da qual será escrito o processo e as suas variáveis.

copiarParaMemoria: Esta função copia o processo pretendido para a posição de memória pretendida, calculada em obterPosicao(). Ou copia o processo pai para posição de memória do processo caso tenha sido chamada através da instrução *fork*.

printMemoria: Esta função escreve no ficheiro 'scheduler_complexo.out' a ocupação da memória no instante em que é chamada.

limparExit: Esta função verifica se o processo que esta em *EXIT* esta lá há um ciclo, caso esteja, remove-o (passando para estado -2) e remove-o da memória.

percorrerBlock: Esta função percorre todos os processos que estão em *BLOCK* para verificar se algum deles se encontra no último instante que precisa de estar em *BLOCK*, caso sim, este escreve ou lê do disco, consoante a operação pretendida, e é colocado de novo em *WAIT*, caso não seja o último instante, o tempo que precisa de estar em *BLOCK* é decrementado em uma unidade.

blockParaWait: Esta função chama a função percorreBlock() dependendo dos elementos que existam na fila *BLOCK*.

runParaBlock: Esta função é executada caso o processo que esteja em *RUN* executa uma instrução que vai ler/escrever do/no disco. Deste modo passa o processo para *BLOCK*.

runParaExit: Esta instrução é executada caso se pretenda colocar um processo em *EXIT* ou matar um processo, sempre que este tenta aceder a memória incorretamente.

runParaWait: Esta função é executada caso o processo que se encontra em *RUN* atinja o seu quantum máximo. Nesse caso o processo é colocado de novo na fila de *WAIT*.

newParaWait: Caso haja espaço suficiente na fila do estado *WAIT*, na memória, e caso existam processos em *NEW*, esta função acrescenta-os à fila do *WAIT*, copia o código dos mesmos para memória e altera-lhes o estado.

waitParaRun: Se não existir nenhum processo em *RUN* e se houver um processo na fila do estado *WAIT*, este é removido da fila, é-lhe atualizado o estado para *RUN* e é também atualizado o seu Quantum.

receberParaNew: É percorrida a lista de todos os processos, caso o instante em que tem que ser introduzido seja igual ao instante atual e ainda não tenha sido introduzido no sitema, esse processo passa para *NEW*.

printEstados: Esta função percorre todos os processos e consoante o estado em que estão coloca no repetivo ficheiro de output "|ESTADO|". Caso exista algo para dar print (devido a uma instrução) ou caso um *fork* falhe, é também adicionada ao output essa informção.

debugPrint: Função a qual serve para dar um print com alguma informação sobre os processos introduzidos. Chamada só caso seja necessário.

violacaoMemoria: Esta função é chamada caso alguma instrução, ou fluxo do processo, acedam a locais inválidos de memória.

main: Esta função é o fluxo principal do programa, nela são inicializadas variávies que irão servir de controlo para algumas funções e um array onde vão ser guardados todos os processos lidos do input. São também inicializados os ficheiros de output.

A memória é inicializada toda com o valor -1 (que significa "lixo"). De seguida é feita uma leitura dos processos que irão ser corridos e são adicionados ao array previamente criada. São também criadas as filas dos estados *WAIT* e *BLOCK*.

É começado um ciclo que apenas irá terminar quando todos os processos tiverem terminado de correr na sua totalidade. Cada iteração é considerada um instante (começando em 0).

Para cada instante, é chamada a função limparExit(). Tendo em conta as prioridades pedidas, as funções são chamadas pela seguinte ordem: blockParaWait(), caso exista um processo em *RUN*, é corrida uma instrução. De seguida são chamadas as funções newParaWait(), waitParaRun(), receberParaNew(), printEstados(), respetivamente. O instante é incrementado em uma unidade.

Casos seja executada sem erros, o valor retornado é 0.

5 Observações sobre a implementação

Existe uma anomalia na implementação do nosso trabalho. Todos os testes disponibilizados pelo professor correm excepto o Teste6 que, na nossa implementação, entra em loop infinito.

Segundo a nossa interpretação do código deve de acontecer exactamente isso visto que existe um *fork* seguido de um salto para trás nas instruções.