Universidade De Évora
Departamento de Informática
Sistemas Operativos 1
Ano letivo 2018 - 2019

Trabalho prático de Sistemas Operativos 1

Escalonador

Modelo de 5 estados

Alunos:
- Luís Ressonha - 35003
- Rúben Teimas - 39868
- Pedro Claudino - 39870

Docentes:
- Luís Rato
- Nuno Miranda

12 de Junho de 2019

Introdução

O objetivo deste projeto é implementar um simulador do comportamento de um escalonador tendo em conta o modelo de 5 estados.

Os programas são constituídos por um conjunto instruções, instruções estas que são codificadas por conjuntos de 3 números inteiros.

Na tabela seguinte encontram-se as instruções, a sua codificação e o significado de cada uma:

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

Figura 1. (instruções)

Descrição e Funcionamento do escalonador

O trabalho está dividido maioritariamente por quatro ficheiros principais denominados por *escalonador, lista, pageList, processo_ instruct.*

Começámos por modificar a primeira parte do trabalho de modo a implementar a gestão de memória com paginação.

Cada página do processo está guardada numa lista de páginas acessível pelo *id* da página.

Em relação aos processos gerados pela instrução *fork* optamos por incrementar uma posição em relação ao ultimo *id* do processo filho começando na posição trezentos e um.

Ficheiros

1 *escalonador.c*

Neste ficheiro encontra-se a nossa função *main()* assim como as funções de mudança de estado e a função utilizada para ler o ficheiro das instruções (*input.txt*).

Após o ficheiro ser lido, existe um ciclo que corre o mesmo numero de vezes que processos existentes, dentro desse ciclo existem condições para verificar os estados dos processos chamando funções para efetuar as devidas alterações no final do ciclo o tempo é incrementado em todos os estados.

1.1 *read_input*

Esta função começa por abrir um ficheiro de input, depois entra num ciclo que corre enquanto o ficheiro não chegar ao fim.

Durante este ciclo os processos são guardados numa lista, o tempo a que cada um é iniciado, a posição de cada instrução e o número de instruções por linha.

1.2 *new_*

Nesta função a lista de processos é percorrida e caso o tempo inicial do processo seja igual ao tempo atual o processo passa para a lista *new*.

1.3 *new_ready*

Efetua a troca do processo do estado *new* para o estado *ready*. Percorre também a lista de processos no estado *ready* pois pode haver mais do que um processo no *ready*.

Dentro desse ciclo ocorre outro ciclo que corre enquanto não tiver alocado todas as instruções, de seguida verifica se a página está vazia e sem elementos alocados caso esteja guarda a página das variáveis e aloca o correspondente espaço, deixando assim a página ocupada e passando à próxima.

Caso encontre apenas uma página vazia percorre o tamanho da página retirando as instruções do array do processo e colocando-as na memória quando a página estiver cheia é colocada como ocupada e passa á página seguinte, por fim se não existir nenhuma página disponível é apenas incrementada uma próxima.

1.4 *ready_run*

Efetua a troca do processo do estado *ready* para o estado *run*.

1.5 *run_ready*

Efetua a troca do processo do estado *run* para o estado *ready*.

1.6 *blocked_ready*

Nesta função se a o estado *blocked* não estiver vazio e o processo estiver no último tempo do block efetua uma das operações possíveis no disco e passa ao estado *ready*.

1.7 *run_blocked*

Efetua a troca do processo do estado *run* para o estado *blocked*.

1.8 *exit_*

Efetua a *saída* do processo do estado *run*.

1.9 *copy_pai_filho*

Esta função guarda numa variável chamada *id* do próximo filho, e passa do processo pai para o processo filho o número de instruções e as instruções, o número de páginas, o *program counter* do processo, coloca também o *id* do processo filho com o valor da variável *id* próximo filho.

1.10 *run_*

Nesta função é onde são feitas as chamadas de função referentes às instruções mencionadas anteriormente na Figura 1. (instruções) da introdução.

1.11 *clear_process*

É criado um ciclo que percorre os processos e verifica se o estado atual desse processo é o *exit*, caso seja dá o processo como terminado e remove-o da memoria.

2 *lista.c*

Este ficheiro contém todas a funções relacionadas com ações sobre as listas, contém também o construtor do nó e o *print* da interface.

2.1 *node_new*

O construtor do nó guarda um valor num elemento e aponta para o próximo nó inicializado a zero.

2.2 *list_new*

Cria uma nova lista.

2.3 *list_empty*

Verifica apenas se a lista está vazia ou não.

2.4 *list_insert*

Insere elementos na lista.

2.5 *list_size*

Retorna o tamanho da lista.

2.6 *list_find*

Verifica se um dado elemento existe na lista.

2.7 *list_remove*

Remove elementos da lista.

2.8 *list_destroy*

Apaga uma lista.

2.9 *print_todos*

Dá *print* de todos os processos do escalonador a cada instante.

2.10 *mais_tempo*

Incrementa a variável do tempo no estado em que o processo está.

3 *pageList.c*

Este ficheiro contém todas a funções relacionadas com ações sobre a paginação.

3.1 *new_pageSet*

Inicializa a tabela completa da paginação.

3.2 *new_page*

Cria uma nova página.

3.3 *new_pageList*

Cria uma nova lista.

3.4 *pageList_insert*

Insere uma nova página na lista, guarda o valor na página no *node* a seguir e o *node* passa a ser a nova *tail*.

3.5 *pageList_remove*

Remove elementos da lista.

3.6 *pageList_Next*

Passa á próxima página.

3.7 *pageList_Previous*

Recua para a página anterior.

4 *processo_instruct.c*

Neste ficheiro é contida a função *new_PCB* e são também implementadas todas instruções base do escalonador.

4.1 *new_PCB*

Nesta função são guardados todos os dados referentes a cada processo.

4.2 *destroy_PCB*

Serve unicamente para apagar um processo.

4.3 *get_instruction*

São efetuadas verificações para localizar todos os algarismos da instrução na respetiva página.

4.4 *Instruções base*

As restantes funções é onde são implementadas todas as instruções mencionadas anteriormente na Figura 1. (instruções) da introdução.

- set_x
- set_n
- inc_x
- dec_x
- back_n
- back_n
- forwd_n
- if_x_n
- disk_op
- print_var
- process_exit

Conclusão

Após a realização deste trabalho podemos afirmar que o objetivo inicial foi atingido, dado que conseguimos, implementar o escalonador tendo em conta o modelo de cinco estados completando todos os testes à exceção do sexto teste ao qual obtemos o erro de *segmentation fault* sem sucesso na resolução deste erro.