Relatório do Trabalho-2 de Sistemas Operativos 1

Trabalho Realizado por:

Leonardo Catarro nº43025

Diogo Solipa nº43071

Índice

Introdução

O programa consiste na implementação de um simulador de Sistema Operativo com um modelo de 3 estados que consome programas constituídos por um conjunto de instruções seguindo determinadas especificações.

Estruturas utilizadas e Funções das mesmas

Neste trabalho foram implementadas duas estruturas de dados, um Queue e uma Linked List.

A Queue é de inteiros, guardando, em cada posição o PID do processo, e tinha como objetivo simular o funcionamento do READY e do RUN. O facto de se ter utilizado uma Queue de PIDs (inteiros) e não de Processos (estavam a criar problemas relativos ao C). Levou a que se tivesse também de criar uma função que comparava um PID com os PIDs da struct Processo.

Para o Blocked foi usado Linked Lists. Estas sim são de Processos e a facilidade de atualizar nodes de Processos ajudou muito.

Em relação às structs, ignorando as das implementações, usei apenas uma, struct Program que contém um array mem[] , um arr[] e uma struct do tipo Process.

Arquitetura do Programa

O programa é fundamentalmente cíclico, inicialmente no input e posteriormente na implementação do algoritmo, não foram usadas funções recursivas.

O programa inicia-se com o preenchimento do array input[] e de um array temp[].

De seguida, abrimos o ficheiro com o input para leitura, calculando com a função nLines() , o numero de instruções INI existentes, correspondendo ao numero de programas existentes.

Após isso, passamos a introduzir nesse array de inteiros temp[] os códigos das instruções de cada programa.

Por fim temos um ciclo que começa com a chamada da função updateProgram() que é responsável pela verificação e execução das operações de acordo com os códigos de instruções existentes. Fazendo uso da função updateScheduler() responsável pelo update do escalonador a cada instrução executada.

Descrição das Funções Usadas

Vou excluir as funções das implementações, pois estas são fundamentalmente semelhantes para todos os programas das mesmas, Queue do tipo FIFO e Linked List com inserção à cabeça.

Função nLines():

Função que recebe um ficheiro e calcula o de programas existentes.

Função moveBlocked():
Insere na linkedList BLOCKED um Processo.

Função blockedReady():

Percorre a lista BLOCKED e verifica se já terminou o tempo de permanência do processo no estado BLOCKED, através do auxilio da função instantIo().

Esta função recebe o tempo de IO do processo, para este trabalho ele corresponde sempre a 5 instantes de CPU, instantes resultantes da instrução DSK e verifica se esse tempo de permanência já terminou. Ao terminar, remove o processo do BLOCKED, inserindo-o no READY.

Função compare():

Função que compara um PID com o PID de um Processo no array de processos.

Função print():
Print na consola do output com o formato previamente exigido.

Função instantCpu():
Calcula os instantes de CPU a cada iteração do programa e decide se deve sair do RUN para o READY.

Função getPid() e Função getEntry():

Retorna o PID de um processo e o tempo de entrada.

Função getRunProcess():

Retorna o Processo que se encontra no RUN.

Função process_new():

Cria um novo processo.

Função updateScheduler():

Função utilizada para atualizar o escalonador conforme as instruções vão sendo executadas.

Função updateProgram():

Função usada para o programa saber o que deve executar a cada instrução existente. Instrução passada apartir do código referente a essa instrução(códigos fornecidos no enunciado do trabalho).

Outras funções já do C utilizadas foram o atoi(string) para passar valores do input de string para inteiros e o memcpy() que copia x bytes de um array para outro , strcmp() e strcpy() para ajudar a trabalhar as strings do input.

O resto do algoritmo foi implementado na main com chamadas de funções pelo meio.

Conclusão

O trabalho serviu muito bem para não só melhorarmos as nossas capacidades de programação como de entendimento do funcionamento do escalonamento com o modelo de 3 estados, embora tenhamos enfrentado inúmeras dificuldades. Dificuldades essas que não permitiram o funcionamento do trabalho na integra. Foi certamente uma tarefa difícil e que deu trabalho e o facto de estarmos em casa não ajudou nada. O C faz com que uma grande parte do processo (implementação das estruturas com pointers) seja dolorosa pois temos demasiados erros que nada tem haver com a lógica do algoritmo, mas sim de má alocação de memória, basicamente debugging de C. Ainda assim uma experiência muito positiva.