UNIVERSIDADE DE ÉVORA

Sistemas Operativos 1-Trabalho Final

Primeira Parte

Criado por:

Carlos Valente-33298

Daniel Soares -34222

Sistemas Operativos 1-Trabalho Final

Primeira Parte

Introdução- O trabalho consiste em criar um escalonador Round Robin , tendo em conta uma arquitetura com o modelo de 5 estados e várias características descritas no enunciado .Neste relatório iremos abordar como encarámos os problemas e como os solucionámos bem como explicar algumas das funções chaves do trabalho .

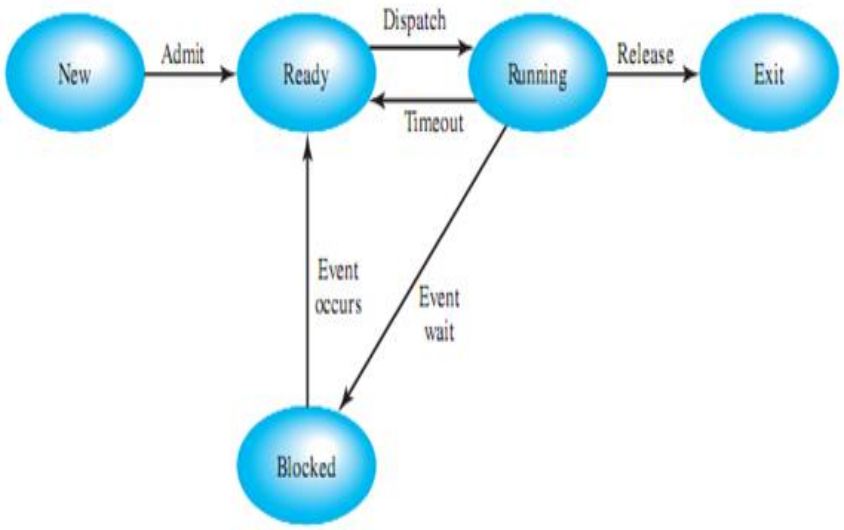
Typedef struct prog- Começámos por perceber que teríamos que criar uma estrutura com todas as informações necessárias do programa , sendo que no inicio apenas tínhamos o Id (correspondente ao número do programa , sendo que o primeiro é 0 o segundo 1 e assim consecutivamente) o Program Counter para no momento em que as instruções fossem lidas ser o indicador de qual a instrução a ser executada , o char Inst[100] (que recebe as instruções do ficheiro de texto que vão ser lidas no runnig) e o int Estado (o 0 corresponde ao "New" , 1 ao "Ready" , 2 ao "Running" , 3 ao "Exit" e o 4 ao "Blocked") , só mais tarde é que percebemos que iriamos precisar de mais duas variáveis na estrutura do programa , o blocked_counter para quando o programa é transferido do Running para o Blocked contar 3 ciclos que é o tempo que deve lá permanecer antes de voltar ao Ready .Também só mais tarde percebemos que teríamos que criar o Go_to_begin para contar os 10 loops antes de mudar a instrução (3) que corresponde ao go_to_begin para (1).

Typedef struct Node- Uma node contém o elemento que neste contexto é um programa e um Node *next que aponta para o próximo nó .

typedef struct Queue-A queue é constituída por nós e usa a estrutura de dados FIFO (first in first out) estrutura vantajosa pois é assim que os nós devem ser inseridos e removidos . A queue tem a o nó da cabeça , o int size que contém o tamanho actual da queue.

void Enqueue(Queue *list, Node *value)- A função Enqueue recebe como argumentos a Queue e o nó que vai ser inserido na respectiva Queue , se a Queue estiver cheia não pode dar Enqueue , se a cabeça estiver vazia significa que a Queue está vazia e o node inserido passa a ser a cabeça da Queue , se a Queue tiver elementos procura o nó em que o next é NULL coloca no nó seguinte o node*value e passa o NULL para esse nó.

Node *Dequeue(Queue *list)- A função Dequeue recebe como argumento a Queue que se pretende retirar a sua cabeça e retorna esse mesmo nó .



void node_prog(prog *P)-Nesta função estamos a introduzir em um nó um programa e damos Enqueue deles no NEW.

void Admit()- A função Admit transfere os programas do New para o Ready , se o ready tiver menos de 4 processos o Admit faz Dequeue(New) e faz enqueue para a queue Ready e o estado recebe o inteiro 1.

void Dispach()- A Função Dispach transfere os programas do Ready para o Running , se o Running estiver vazio e o Ready não , o processo passa para ready e o estado recebe o inteiro 2.

void Release()- O release envia o processo para a exit e o estado recebe o inteiro 3.

void run()- Nesta função vamos correr o array de instruções apenas se o programa estiver no running , se as instruções forem :

- Valor 0 chamamos o Release() que leva a o programa para o Exit.
- Valor 1 incrementamos o Program Counter e avançamos para a instrução seguinte.
- Valor 2 incrementamos o Program Counter para avançar para a próxima instrução na próxima vez que for lido ,colocamos o blocked_counter a 0 para que depois mais tarde vir a ser incrementado na função Blocked_counter_increment() (chamada no main) , assim que atingir os 3 ciclos chama o void Event_Occurs() que volta a colocar o programa no Ready.
- Valor 3 se o go_to_begin for menor que 10 o Program Counter volta a 0 para voltar a ler as instruções do inicio e incrementa o Programa Counter. Se o go_to_begin for igual a 10 a instrução correspondente ao Program Counter actual muda para 1 , o Program Counter volta a ser incrementado para ler o resto das instruções e o go_to_begin volta a 0.

void output()- O output cria um array onde cada estado do programa vai ficar alocada na posição prog->id do array e chama a função scheduler.

void Scheduler(int* Array)-Chama um array e substitui os valores inteiros por strings, onde as mesmas são os estados dos programas .

Conclusão-Através deste trabalho foi-nos possível consolidar alguns conhecimentos na área dos Sistemas Operativos. Ficámos de alguma forma satisfeitos por conseguirmos realizar muito dos objetivos propostos, apenas não conseguimos fazer correr mais que um programa ao mesmo tempo , mas com mais tempo seguramente que iremos resolver este problema.