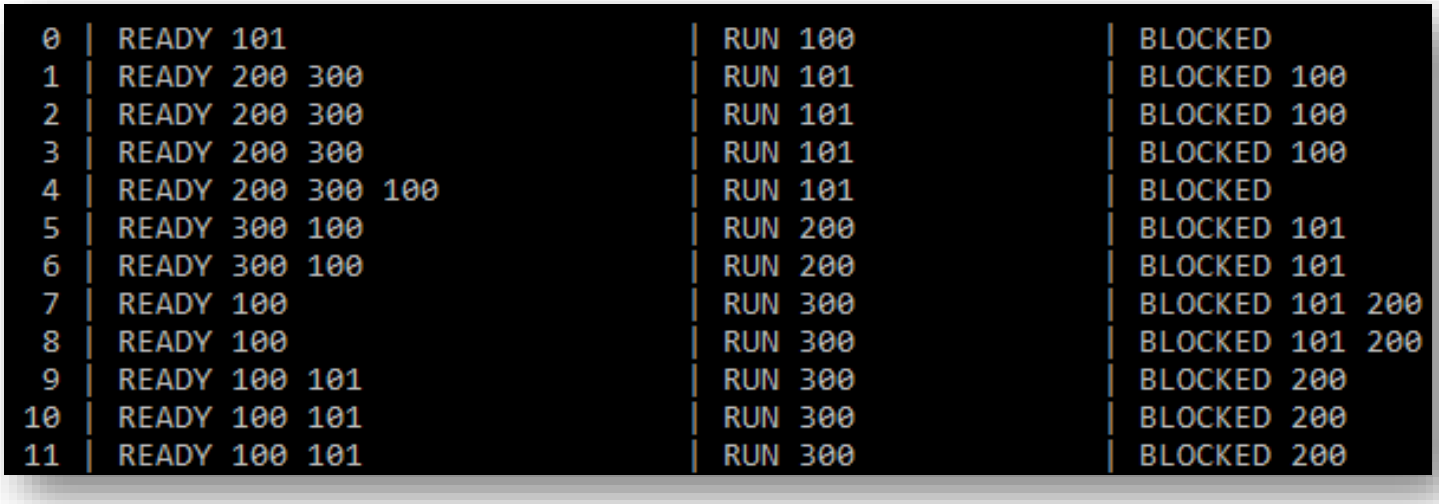
Relatório – Trabalho 1

Sistemas Operativos

Simulador de escalonamento de um sistema operativo



Trabalho realizado por:

Pedro Grilo, 43012

1. Introdução

Suponha uma arquitetura sobre o modelo de 3 estados que consome programas constituídos por um conjunto instruções. O objetivo deste trabalho é implementar um simulador de escalonamento dum sistema operativo. As instruções são codificadas por uma sequência de números inteiros representando alternadamente o tempo (burst) de CPU e um tipo de pedido I/O (por exemplo acesso ao disco), e.g

5 1 3 3 5

é equivalente à seguinte sequência

5 instantes no CPU

acesso a I/O com espera de 1 instante

3 instantes no CPU

acesso a I/O com espera de 3 instantes

5 instantes no CPU

Sendo que cada sequência de instruções tem sempre um número ímpar de elementos (e por isso termina sempre com um burst de cpu).

Os ficheiros de teste têm a indicação do PID (que será único para cada processo) e do instante de entrada, seguida da sequência de instruções, separadas por espaços.

O input será algo do género:

100 0 5 1 11 3 5

110 3 2 2 2 3 1

122 4 5 1 3 7 10

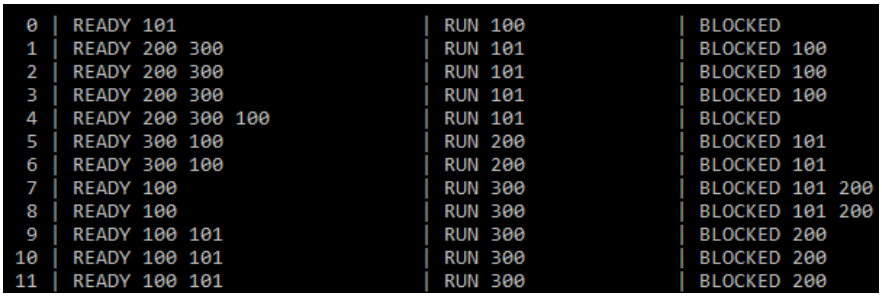
101 7 1 1 3 3 8

Onde os números > 100 representam o PID, o numero que os segue representa o tempo de chegada desse processo, seguidos da sequência de burst e i/o.

1.1 Introdução

Foi-nos proposto implementar um escalonamento FCFS e outro ROUND ROBIN, com Quantum = 3, com as seguintes prioridades: quando no mesmo instante, um processo novo ou vindo de BLOCKED, e /ou do RUN pretendem entrar na fila de READY, o vindo do BLOCKED tem prioridade, seguido do de RUN, e por fim o processo novo. O output deverá apresentar em cada instante a lista de processos (indicando os PIDs) em cada um dos estados: READY, RUN e BLOCKED, e.g.

O output será algo como: (imagem ilustrativa, não representa o input acima)



2. Decisões tomadas na realização do trabalho

Em primeiro lugar tentei perceber como funcionava melhor os escalonamentos que tinha que implementar.

O primeiro troço de código criado foi a implementação das filas que foram recomendadas pelo professor para a realização deste trabalho. Após alguma pesquisa e usando conhecimentos anteriores foi a parte mais fácil de se implementar.

De seguida, e a parte mais dificil do trabalho, foi a leitura do input e consequentemente a atribuição dos valores corretos às variáveis da struct processo corretas.

Depois disso foram então implementados os escalonamentos, sendo que primeiramente foi o FCFS.

3. Lista de funções utilizadas

- Funções frequentes da biblioteca stdio.h, como por exemplo("printf", "scanf").

- **Void criarFila(Fila_processo *f, int c)** – cria uma fila f, com a estrutura de uma fila processo, e com capacidade c. Como qualquer fila, serão lhe atribuídos valores para o primeiro e ultimo elemento da fila.

- **Void inserir(Fila_processo* f, Processo* processo)** – esta função insere numa fila f, um processo com a estrutura de Processo, definida anteriormente, sendo que por isso o tamanho da fila aumenta

- **Void remover(Fila_processo *f)** – função que remove o item que está no inicio da fila, diminuindo assim o seu tamanho.

- **Void mostrarFila(Fila_processo *f)** – mostra o PID de todos os processos que se encontram na fila de momento

- **Processo* novoProcesso()** – função que retorna um processo já com memória alocada, atribuindo o valor 0 aos dois atributos de processo auxiliar1 e auxiliar2 que representam o índice dos burst(se burst = 5 representa 5,4,3,2,1 por exemplo) e dos Ios, da mesma maneira.

- **Processos* organizar_processos(int array[], int inicio, int fim)** – função que tem como objetivo atribuir a um novo processo criado os valores corretos do pid, t_inicio e da sequência de burst e de IOS correspondente. Retorna essa novo processo criado.

- **Int dígitos(int numero)** – função que recebe um numero inteiro e retorna o numero de dígitos do mesmo. Por exemplo, numero = 100, retorna 3 digitos.

- **Int totalBurstTime1(int array[], int numero_total)** – função que vai retornar a soma de todos os burts presentes num array, que neste caso será um que conterá todos os números do input.

- **Int numero_processos(int array[], int numero_total)** – função que retorna o numero de processos existentes num array.

3.1 Lista de funções utilizadas

- **Void ready_check(Fila_processo* f, Processo* lista[], int instante)** – função que tem como objetivo inserir na lista f, que será a ready, todos os processos de Processo* lista[] que tenham um t_inicio igual ao do instante em que estará o ciclo na main.

- **int run(Fila_processo* f, Processo* processo, int auxiliar)** – função que retorna o PID do Processo *processo se o burst time do mesmo ainda não for 0, ou se o burs time[auxiliar1] do processo for igual a 0, remove o mesmo da fila run, e aumenta a variável auxiliar1 para o seguinte, que será o próximo valor de burst desse processo.

- **Int toBlocked(Fila_processo *f)** – função que tem como objetivo ver se o processo[primeiro] da fila f, ainda tem IO times para executar ou não. Decrementa o valor desse IO time especifico e se esse já estiver igual a 0, ou seja, acabou o tempo wait, vai dar à variável auxiliar o valor 1 para que na função fcfs(que agrega todas estas funções acima mencionadas) seja possível inserir esse processo na fila ready e retirar-se então esse processo da fila blocked após terminar o seu tempo de espera.

- **void fcfs(Processo *array_processos[], int total_burst, int numero_processos1, int total_numeros1, int array[])** – funcao que recebera uma lista de processos que será o array com size de números_processos1, que conterá em cada índice um processo criado pela função organizar_processos, já com valores atribuídos de maneira correta (vindos do input).

  Usando a função criar fila iremos criar filas para os 3 estados, blocked, ready e fila(que será do run), sendo que depois teremos um ciclo principal que correrá enquanto o instante for menor que total_burst calculado pela função totalBurstTime1. Dentro desse while iremos usar as funções acima para inserir, remover de cada fila os processos e dar os printfs necessários em cada ciclo e nos instantes corretos, com as propriedas corretas.

- **Void round_robin(Processo *array_processos[], int total_burst, int numero_processos1, int total_numeros1, int array[])** – funcao que implementa o algoritmo de round robin com quantum = 3.
  Devido a alguns problemas não percetiveis por mim não foi possível ser implementada a 100%, faltando assim algum troço de código importante na sua realização.

3.2 Lista de funções utilizadas

- **Int main()-** função que irá abrir o ficheiro, que o irá ler e colocar num vetor todos os números que ler do input. Para além disso irá contar o numero de números existentes nesse ficheiro. Terá opção para escolher que escalonamento quer executar. Se não inserir um numero correto não correrá nenhum dos escalonamentos implementados.

4. Estruturas de dados

- typedef struct {
  int PID;
  int t_inicio;
  int burstTime[BURST_MAX];
  int ioTime[IO_MAX];
  int tamanho_burst;
  int tamanho_io;
  int auxiliar1;
  int auxiliar2;

  }Processo;

  Estrutura para um processo. Com as variáveis PID, que representa unicamente o processo em si, t_inicio, instante em que ele chegue.
  Depois temos um array burstTime que terá tamanho máximo 4, pois o último será o '\0', que conterá todos os burstimes de um processo. Depois temos um array ioTime igual ao de burst mas para os tempos de wait do mesmo.
  A variável tamanho burst representa a quantidade de burst existentes, e o do tamanho io a quantidade de io existentes.
  A variável auxiliar1 reresenta o índice dentro de um instante de burst, e a variável auxiliar2 representa também um índice, mas dentro de um instante de wait.

4.1 Estruturas de dados

- typedef struct{
  int primeiro;
  int ultimo;
  int tamanho;
  int capacidade; //nItens
  Processo* processos;

  }Fila_processo;

  Estrutura para uma fila que conterá um Processo*processos, e que terá variável primeiro para o primeiro elemento da fila, um ultimo para o ultimo elemento da fila, uma para o tamanho da mesma e outra para a sua capacidade.

5. Demonstração de resultados

- Input1 com FCFS

```
- >> Numero de processos no ficheiro: 4
- >> Numero de bursts: 37

 0 | READY 101                | RUN 100          | BLOCKED
 1 | READY 200 300            | RUN 101          | BLOCKED 100
 2 | READY 200 300            | RUN 101          | BLOCKED 100
 3 | READY 200 300            | RUN 101          | BLOCKED 100
 4 | READY 200 300 100        | RUN 101          | BLOCKED
 5 | READY 300 100            | RUN 200          | BLOCKED 101
 6 | READY 300 100            | RUN 200          | BLOCKED 101
 7 | READY 100                | RUN 300          | BLOCKED 101 200
 8 | READY 100                | RUN 300          | BLOCKED 101 200
 9 | READY 100 101            | RUN 300          | BLOCKED 200
10 | READY 100 101            | RUN 300          | BLOCKED 200
11 | READY 100 101            | RUN 300          | BLOCKED 200
12 | READY 100 101 200        | RUN 300          | BLOCKED
13 | READY 100 101 200        | RUN 300          | BLOCKED
14 | READY 101 200            | RUN 100          | BLOCKED 300
15 | READY 101 200            | RUN 100          | BLOCKED 300
16 | READY 101 200            | RUN 100          | BLOCKED 300
17 | READY 101 200            | RUN 100          | BLOCKED 300
18 | READY 101 200            | RUN 100          | BLOCKED 300
19 | READY 101 200            | RUN 100          | BLOCKED 300
20 | READY 101 200 300        | RUN 100          | BLOCKED
21 | READY 101 200 300        | RUN 100          | BLOCKED
22 | READY 101 200 300        | RUN 100          | BLOCKED
23 | READY 101 200 300        | RUN 100          | BLOCKED
24 | READY 200 300            | RUN 101          | BLOCKED 100
25 | READY 200 300            | RUN 101          | BLOCKED 100
26 | READY 300                | RUN 200          | BLOCKED 100
27 | READY 100                | RUN 300          | BLOCKED 200
28 | READY                    | RUN 100          | BLOCKED 200
29 | READY 200                | RUN 100          | BLOCKED
30 | READY 200                | RUN 100          | BLOCKED
31 | READY 200                | RUN 100          | BLOCKED
32 | READY 200                | RUN 100          | BLOCKED
33 | READY 200                | RUN 100          | BLOCKED
34 | READY                    | RUN 200          | BLOCKED
35 | READY                    | RUN 200          | BLOCKED
36 | READY                    | RUN 200          | BLOCKED
```

5.1 Demonstração de resultados

- Input2 com FCFS

```
- >> Numero de processos no ficheiro: 4
- >> Numero de bursts: 53

 0 | READY 101                    | RUN 100              | BLOCKED
 1 | READY 101 200 300            | RUN 100              | BLOCKED
 2 | READY 101 200 300            | RUN 100              | BLOCKED
 3 | READY 101 200 300            | RUN 100              | BLOCKED
 4 | READY 200 300                | RUN 101              | BLOCKED 100
 5 | READY 200 300 100            | RUN 101              | BLOCKED
 6 | READY 200 300 100            | RUN 101              | BLOCKED
 7 | READY 300 100                | RUN 200              | BLOCKED 101
 8 | READY 300 100 101            | RUN 200              | BLOCKED
 9 | READY 300 100 101            | RUN 200              | BLOCKED
10 | READY 300 100 101            | RUN 200              | BLOCKED
11 | READY 300 100 101            | RUN 200              | BLOCKED
12 | READY 100 101                | RUN 300              | BLOCKED 200
13 | READY 100 101                | RUN 300              | BLOCKED 200
14 | READY 100 101 200            | RUN 300              | BLOCKED
15 | READY 100 101 200            | RUN 300              | BLOCKED
16 | READY 100 101 200            | RUN 300              | BLOCKED
17 | READY 100 101 200            | RUN 300              | BLOCKED
18 | READY 101 200                | RUN 100              | BLOCKED 300
19 | READY 101 200 300            | RUN 100              | BLOCKED
20 | READY 101 200 300            | RUN 100              | BLOCKED
21 | READY 101 200 300            | RUN 100              | BLOCKED
22 | READY 101 200 300            | RUN 100              | BLOCKED
23 | READY 101 200 300            | RUN 100              | BLOCKED
24 | READY 200 300                | RUN 101              | BLOCKED 100
25 | READY 200 300                | RUN 101              | BLOCKED 100
26 | READY 200 300 100            | RUN 101              | BLOCKED
27 | READY 200 300 100            | RUN 101              | BLOCKED
28 | READY 200 300 100            | RUN 101              | BLOCKED
29 | READY 200 300 100            | RUN 101              | BLOCKED
30 | READY 200 300 100            | RUN 101              | BLOCKED
31 | READY 300 100                | RUN 200              | BLOCKED
32 | READY 300 100                | RUN 200              | BLOCKED
33 | READY 300 100                | RUN 200              | BLOCKED
34 | READY 300 100                | RUN 200              | BLOCKED
35 | READY 300 100                | RUN 200              | BLOCKED
36 | READY 300 100                | RUN 200              | BLOCKED
37 | READY 100                    | RUN 300              | BLOCKED 200
38 | READY 100                    | RUN 300              | BLOCKED 200
39 | READY 100 200                | RUN 300              | BLOCKED
40 | READY 100 200                | RUN 300              | BLOCKED
41 | READY 100 200                | RUN 300              | BLOCKED
42 | READY 200                    | RUN 100              | BLOCKED
43 | READY 200                    | RUN 100              | BLOCKED
44 | READY 200                    | RUN 100              | BLOCKED
45 | READY 200                    | RUN 100              | BLOCKED
46 | READY 200                    | RUN 100              | BLOCKED
47 | READY 200                    | RUN 100              | BLOCKED
48 | READY                        | RUN 200              | BLOCKED
49 | READY                        | RUN 200              | BLOCKED
50 | READY                        | RUN 200              | BLOCKED
51 | READY                        | RUN 200              | BLOCKED
52 | READY                        | RUN 200              | BLOCKED
```

6. Comentário critico

Após muito esforço e depois de várias horas durante o tempo da realização do trabalho apena foi possível realizar com sucesso o escalonamento FCFS, pelo que o ROUND ROBIN não ficou 100% bem implementado, tendo alguns problemas dos quais eu não consegui resolver.