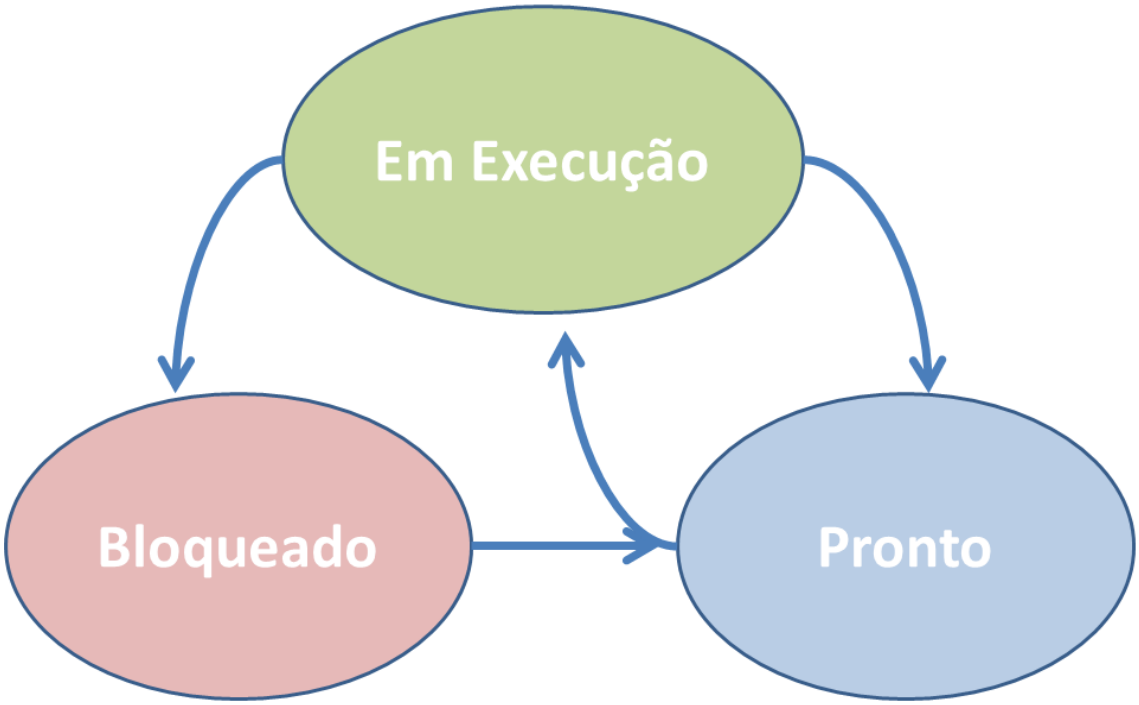
Universidade de Évora

Sistemas Operativos I

Escalonamento



Discente:
Filipe Alfaiate nº 43315

Docente:
Luís Rato

Índice

Introdução

Todos os processos que executam no computador são organizados em vários processos sequenciais. Um processo é apenas um programa em execução acompanhado dos valores atuais do contador de programa, dos registradores, e das variáveis. Introdução a Processos Conceitualmente, cada processo tem o seu próprio CPU virtual, mas, na realidade, o CPU troca, a todo o momento, de um processo para outro. Esses conceitos de pseudo paralelismo é conhecido como multiprogramação.

Descrição das funções

queue_t *queue_new (int size): constructor da queue;

void enqueue (queue_t *queue_v, int elemento): coloca elementos dentro da queue;

int dequeue (queue_t *queue_v): remove elementos da queue;

bool empty (queue_t *queue_v): retorna um booleano com o estado da queue, *true* cajo esteja vazia, *false* caso exista algum elemento;

int top (queue_t *queue_v): devolve o ultimo valor inserido na queue;

bool full (queue_t *queue_v): retorna um booleno com o estado da queue, *true* cajo esteja cheio, *false* caso contrário;

void printAll (queue_t *queue_v): mostra no ecrã todos os valores que existem na queue;

int posição (queue_t *queue_v, int pos): remove um elemento a meio da queue;

processo_t *processo_new (int size): constructor do processo;

processo_t *input_processo (int arr[ ], int ini, int fim): organiza o processo;

int posicao_PID (int PID, processo_t *processo_v[ ], int n_processo): devolve a posição em que o PID se encontra no array de processos;

void atualizar_processo_blocked (int n_processo, processo_t *processo_v[], int size): move cada posição do array blocked do processo (n_processo) uma posição para trás;

void atualizar_processo_run (int n_processo, processo_t *processo_v[], int size): move cada posição do array run do processo (n_processo) uma posição para trás;

void fcfs (int n_processo, processo_t *arr_processos[]): faz o escalonamento First Come First Served;

void rr (int n_processo, processo_t *arr_processos[]): faz o escalonamento Round Robin;

Pensamento

Face ao trabalho proposto, o inicio foi pensar como deveria abordar o problema, à priori desenhar em papel como os processos deviam de saltar entre estados. Após chegar à conclusão de que iriamos precisar de implementar uma struct de queues para cada estado foi dado início à escrita de código.

Na implementação das queues, criamos alguns métodos principais: queue_new, enqueue, dequeue, empty, full, top e por fim, printAll. Depois de verificar que todos os métodos estavam a funcionar de forma correta, foi passado para o próximo passo, ler os inputs de um ficheiro. Aqui foram encontradas algumas dificuldades, pois não se sabia como se podia guardar todos os inputs para que quando fossem ser utilizados, fosse uma busca fácil de se fazer.

Após investigar diversas formas com arrays, como, colocar cada processo e a sua informação num array, colocar os PID's, o tempo de inicio, os tempos de run e os tempos de blocked cada coisa em arrays separados, mas nenhum desses métodos de pensamento era uma forma fácil de utilizar. Por fim, a imaginação brotou que podia criar outra struct com tudo o que precisava de cada processo e guardar então a struct de processos num array. Na progressão da implementação da struct de processos inicializou-se por criar 2 métodos fundamentais, processo_new e input_processo.

Depois de conseguir guardar cada processo e as suas informações individualmente de outro processo, fui capaz então criar a função que irá realizar os saltos do escalonamento. Durante esse desenvolvimento, foi deparado alguns problemas que resultaram na necessidade da criação de mais métodos nas structs. Então, paramos de desenvolver a função e demos um passo atrás para puder remendar o código corretamente. Na struct de queue foi acrescentada o método posição e na struct de processos foi acrescentado mais 3 métodos: posicao_PID, atualizar_processo_blocked e atualizar_processo_run.

Por fim, voltando à criação da função, fui testando à medida que ia criando, e corrigindo os bugs e as imperfeiçoes da função.

Funcionamento

O programa de escalonamento funciona num ciclo infinito, que termina quando o utilizador insirir um valor diferente do qual é pedido. As funções principais do programa (fcfs e rr), são também funções de ciclo mas terminam quando todos os processos terminarem.

Existem 2 estruturas de dados, queue e processo. A função da sruct de queue é guardar o PID consoante o seu estado, já a struct de processo tem como função, organizar os processos e atualizar os processo.

Outputs

Input1 com o escalonamento FCFS:

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:

Conclusão

Durante a execução do trabalho, não houve grandes dúvidas ou dificuldades, apenas houve um obstáculo maior de ultrapassar, que foi de como obter a informação do input e guarda-la. Mas, mesmo assim, este foi ultrapassado com esforço e bastante investimento próprio, criando uma struct de processos para solucionar o dilema.

Com este trabalho absorvemos mais conhecimento e prática na escrita de código e até mesmo diferentes formas de como solucionar os problemas que vão surgindo no decorrer do trabalho.

Por fim, consideramos que este trabalho se encontra bem sintetizado, organizado e simples, demonstrando objetivamente os conceitos e conteúdos interiorizados no decorrer das aulas.