

ALGORITMOS DE ESCALONAMENTO

Introdução à informática
Professor Hugo Pinheiro

INTEGRANTES DO GRUPO 7



FERNANDA



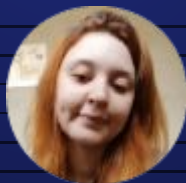
RODRIGO



KENNETH



LUANDERSON



AMANDA



JADE

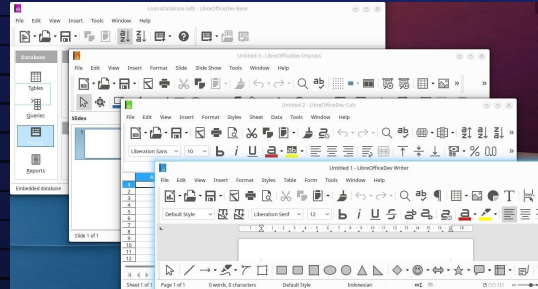
O QUE VOCÊ VERÁ POR AQUI...

- 1) O Algoritmo de Escalonamento
- 2) Tipos de Algoritmos
- 3) Curiosidades



OS PROCESSOS

Um processo é um programa em execução
junto ao ambiente associado (registros,
variáveis, entre outros)



O ESCALONADOR

- O que dita as normas
- Existem diversas técnicas/algoritmos para o escalonamento de processos



Na fila do banco...

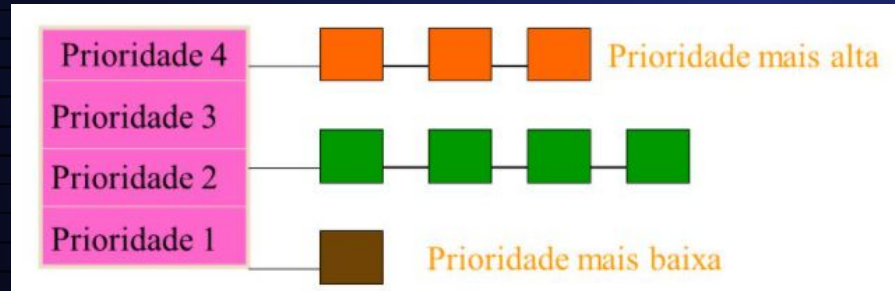
O que são esses Algoritmos?

Uma divisão de tempo em que cada processo utilizará a CPU



Para que servem?

- Evitar ociosidade na CPU
- Lentidão no computador
- Dar a impressão, para o usuário, que executa mais de uma tarefa ao mesmo tempo



O que eles envolvem?

- Processador
- Núcleos



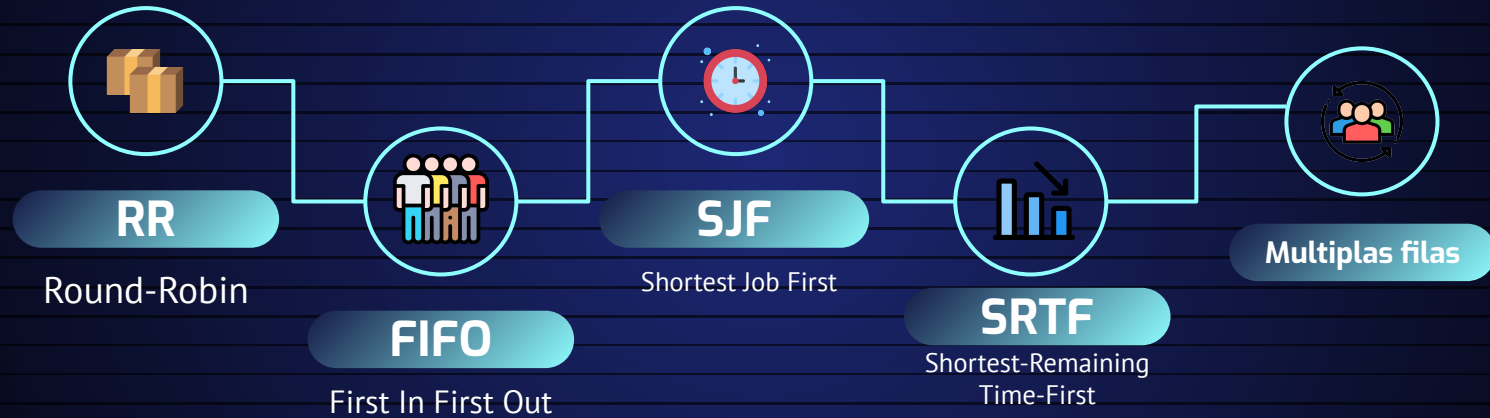


TIPOS DE ALGORITMOS

Quais são os principais tipos de algoritmo de escalonamento que foram ou são utilizados.



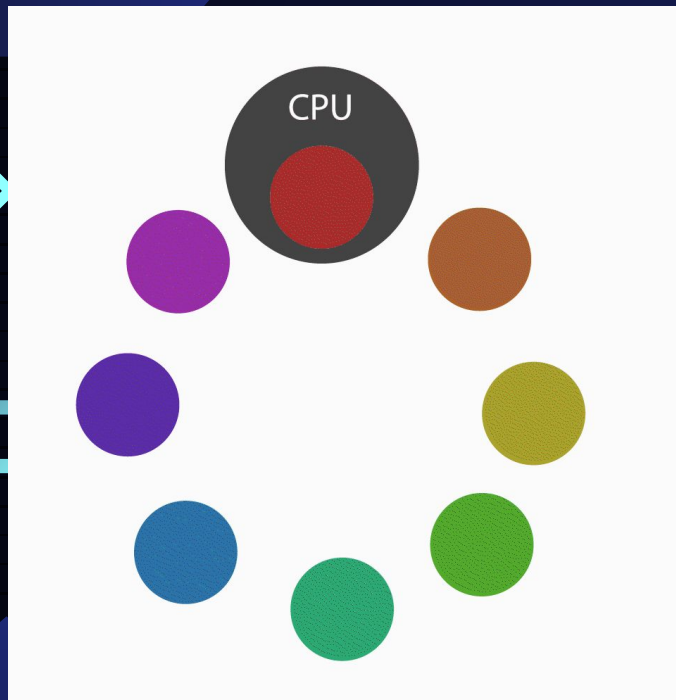
ALGORITMOS





RR

Round-Robin



O que é o Round Robin?

O RR é um dos algoritmos mais antigos e é usado em sistemas operacionais multitarefas. Ele funciona utilizando preempção.

O que é preempção?

É a interrupção abrupta/forçada de processos para que outro seja executado.

E porque o Round Robin leva esse nome?

O RR tem seus processos organizados em fila circular, explicando o nome de Round Robin (que traduzido significa rodízio do inglês).

Exemplo de funcionamento:

- Se o quantum é 100 milisegundos e a tarefa leva 250 milisegundos para completar, o agendamento round-robin suspenderá a tarefa após os primeiros 100 milisegundos e dará a outra tarefa da fila, o mesmo tempo. Essa tarefa será executada portanto após 3 agendamentos a saber (100 ms + 100 ms + 50 ms). A interrupção da tarefa é conhecida como preempção.
- Tarefa 1 = Tempo de execução igual a 250 ms (quantum 100 ms).
- 1. Primeiro agendamento = executa tarefa durante 100 ms.
- 2. Segundo agendamento = mais 100 ms de execução da tarefa.
- 3. Terceiro agendamento = 100 ms, mas a tarefa termina após os primeiros 50 ms.

Vantagens X Desvantagens

Vantagens

- Simples de implementar
- Todos os trabalhos recebem uma alocação justa de CPU.
- Permite o melhor desempenho em termos de tempo médio de resposta.

Desvantagens

- Se houver processos longos ativos, tarefas de curta duração podem ter seu tempo prejudicado.
- O desempenho do processador depende muito do quantum de tempo.

The background is a dark blue gradient with various geometric shapes and patterns. There are several light blue arrows pointing in different directions (up, down, left, right). There are also some light blue lines and dots scattered around. The text 'FIFO' is in a large, bold, light blue font, and 'FIRST IN FIRST OUT' is in a smaller, bold, white font below it.

FIFO

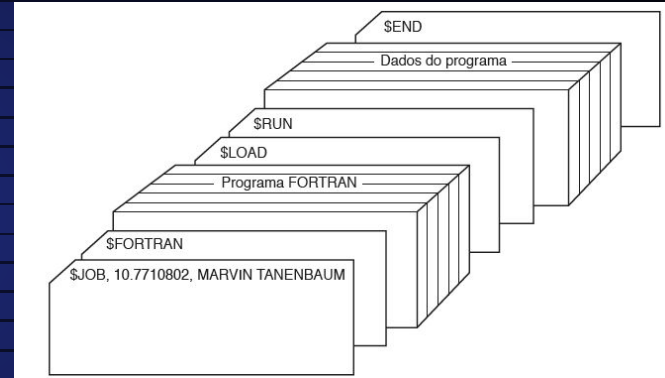
FIRST IN FIRST OUT

- FIFO significa primeiro a chegar, primeiro a sair.



Contexto Histórico

- Segunda Geração - Sistema em Lote (batch) - Década de 60.
- Conjunto de JOBS (tarefas) colocados de uma vez, e executados um após o outro.

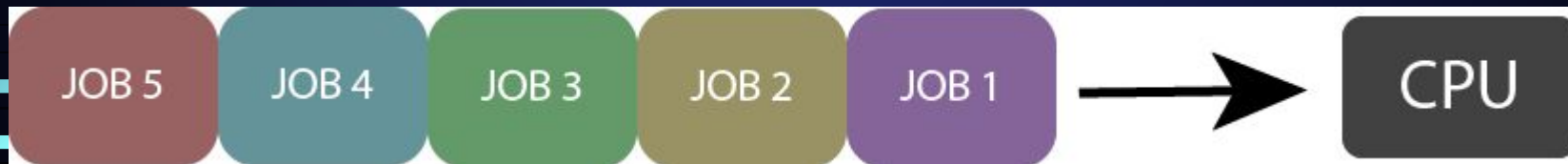


Estrutura de um job do Sistema Operacional FMS.

- Não exige a interação do usuário com a aplicação.

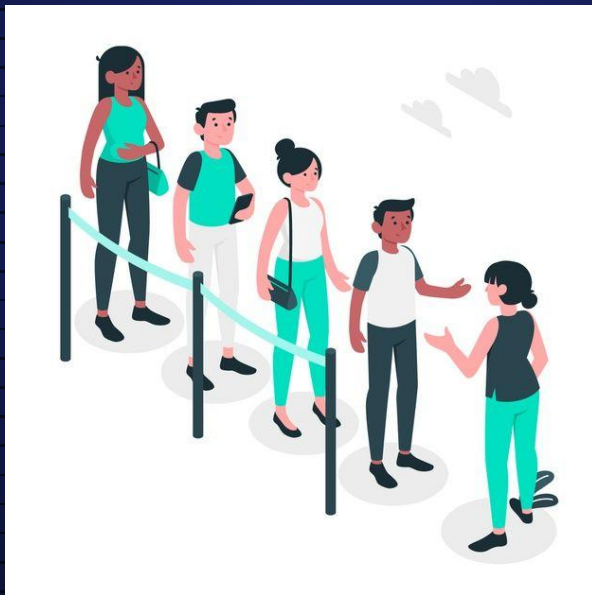
Exemplos de programas:

- cálculos numéricos
- compilações
- ordenações
- backups.



Analogia: FILA.

- FIFO não prioriza o tempo de execução, mas sim a ordem que os processos chegam à CPU.



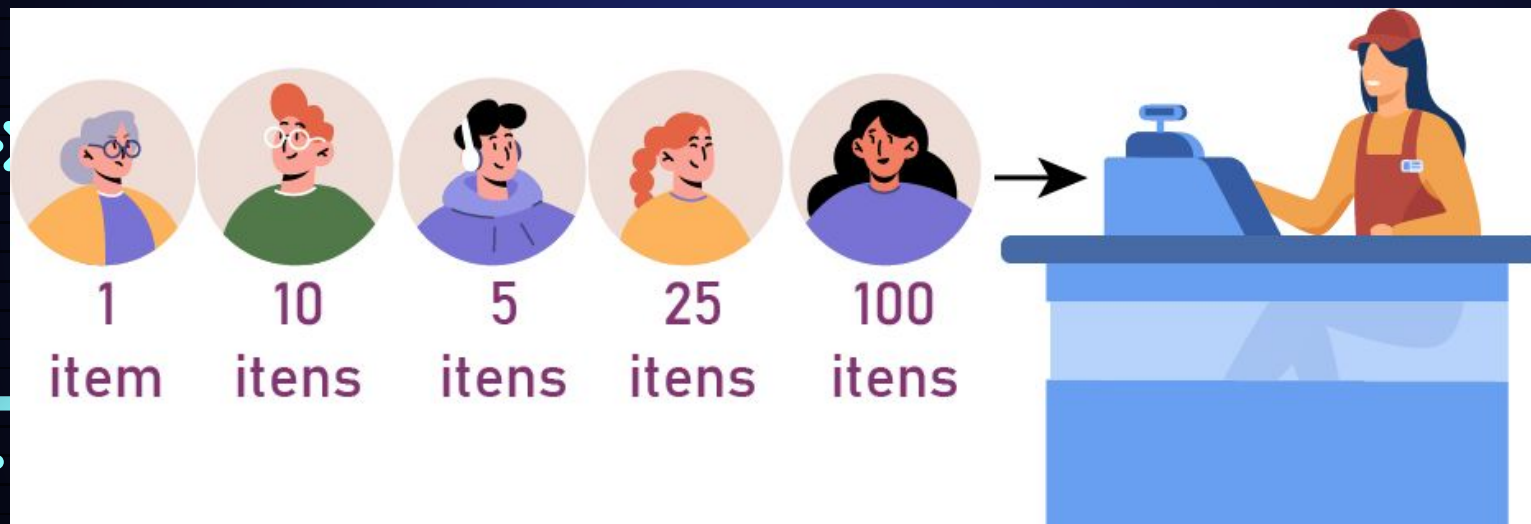
Analogia com o dia a dia: fila de supermercado


- Vantagem: simples!
- O que importa é a ordem de chegada (nada de furar fila).



Desvantagem

- Maior tempo de espera por processos curtos
- Não existem processos preferenciais (não preemptivo).





SJF

SHORTEST JOB FIRST

Como funciona o SJF ?

- O SJF age da seguinte maneira, ele dá prioridade a processos com o menor tempo de execução em fila.

TABLE

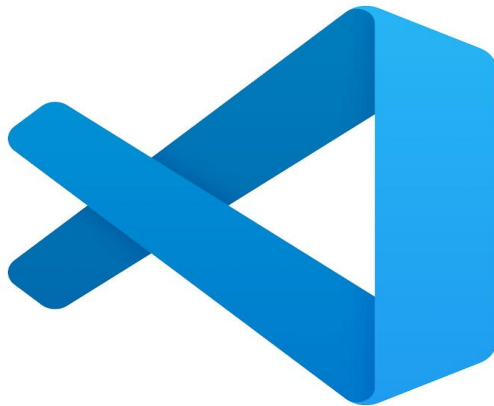
Process	Burst Time
P1	21
P2	3
P3	6
P4	2

**Gantt
Chart**



O SJF é usado comumente ?

- O uso do SJF, é muito comum de ser usado porém não perceptivo. como assim não perceptivo ?, quando executamos um programa no nosso sistema operacional, o uso de escalonamento é ativado pela CPU, referente a esse programa para a realização da execução.



Exemplo SJF no cotidiano.

ATENDIMENTO PRIORITÁRIO



DEFICIENTES

IDOSOS

GESTANTES

PESSOAS COM
CRIANÇA DE COLO

AUTISTAS



Curiosidades sobre o SJF:

- Em caso de empate, usa-se o FIFO para desempate.
- Quando a informação sobre o tempo de execução(CPU burst) não se encontra disponível, o ciclo deve ser estimado ou usa-se o anterior.
- Associa a cada processo a duração de seu próximo tempo de execução.
- A CPU é atribuída ao processo que tem o menor tempo de execução.



The background is a dark blue gradient with various light blue and white decorative elements. These include circuit-like lines with small circles at the ends, several arrows pointing in different directions (some solid, some dotted), and horizontal lines of dots. The overall aesthetic is technical and digital.

SRTF

Shortest Remaining Time First

A sigla em inglês SRTF significa “Shortest Remaining Time First”
- Menor tempo restante primeiro.







Algoritmo de agendamento de primeiro tempo restante mais curto.



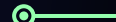


A versão preemptiva do agendamento Shortest Job First (SJF) é conhecida como Shortest Remaining Time First (SRTF). Com a ajuda do algoritmo SRTF, o processo com a menor quantidade de tempo restante até a conclusão é selecionado primeiro para ser executado.


Então basicamente no SRTF, os processos são escalonados de acordo com o menor tempo restante.



No entanto, o algoritmo SRTF envolve mais overheads do que o escalonamento do Shortest job first (SJF), porque no SRTF OS é necessário frequentemente para monitorar o tempo de CPU dos jobs na fila READY e para realizar a troca de contexto.



No algoritmo de escalonamento SRTF, a execução de qualquer processo pode ser interrompida após um determinado período de tempo. Na chegada de cada processo, o escalonador de curto prazo agenda os processos da lista de processos disponíveis e processos em execução que têm o menor tempo de intermitência restante.



Vantagens

- A principal vantagem do algoritmo SRTF é que ele torna o processamento dos trabalhos mais rápido que o algoritmo SJN, já que seus custos indiretos não são contabilizados;
- Vantagem sobrepor processamento com operações E/S (multiprogramação);
- Busca minimizar tempo de espera;

Desvantagens

- No SRTF, a troca de contexto é feita muito mais vezes do que no SJN devido ao maior consumo do valioso tempo de processamento da CPU. O tempo consumido da CPU então se soma ao seu tempo de processamento e isso diminui a vantagem do processamento rápido desse algoritmo.

EXEMPLOS

Explicação

Na 0ª unidade da CPU, há apenas um processo que é P1, então P1 é executado para a 1 unidade de tempo.

Na 1ª unidade da CPU chega o Processo P2. Agora, o P1 precisa de mais 6 unidades a mais para ser executado, e o P2 precisa de apenas 3 unidades. Então, P2 é executado primeiro pela preempção de P1.

Na 3ª unidade de tempo, o processo P3 chega, e o tempo de rajada de P3 é de 4 unidades, que é maior que o tempo de conclusão de P2 que é de 1 unidade, então P2 continua sua execução.

EXEMPLOS

Process	Arrival Time	Burst Time	Completion time	Turn around Time Turn Around Time = Completion Time – Arrival Time	Waiting Time Waiting Time = Turn Around Time – Burst Time
P1	0	7	14	$14-0=14$	$14-7=7$
P2	1	3	4	$4-1=3$	$3-3=0$
P3	3	4	8	$8-3=5$	$5-4=1$

Implementação

```
#include <iostream> • Untitled-1 - dsmovie2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
styles.css .../Pagination #include <iostream> Untitled-1 index.tsx .../Pagination styles.css .../MovieStars

1  #include <iostream>
2  #include <algorithm>
3  #include <iomanip>
4  #include <string.h>
5  using namespace std;
6  struct process {
7      int pid;
8      int arrival_time;
9      int burst_time;
10     int start_time;
11     int completion_time;
12     int turnaround_time;
13     int waiting_time;
14     int response_time;
15 };
16 int main() {
17     int x;
18     struct process p[100];
19     float avg_turnaround_time;
20     float avg_waiting_time;
21     float avg_response_time;
22     float cpu_utilization;
23     int total_turnaround_time = 0;
24     int total_waiting_time = 0;
25     int total_response_time = 0;
26     int total_idle_time = 0;
27     float throughput;
28     int burst_remaining[100];
29     int is_completed[100];
30     memset(is_completed, 0, sizeof(is_completed));
31
32     cout << setprecision(2) << fixed;
33
34     cout<<"Enter the number of processes: ";
35     cin>>x;
36
37     for(int i = 0; i < x; i++) {
38         cout<<"Enter arrival time of the process "<<i+1<<" : ";
39         cin>>p[i].arrival_time;
40         cout<<"Enter burst time of the process "<<i+1<<" : ";
41         cin>>p[i].burst_time;
42         p[i].pid = i+1;
43         burst_remaining[i] = p[i].burst_time;
44         cout<<endl;
45     }
46
47     int current_time = 0;
48     int completed = 0;
49 }
```

Implementação

```
#include <iostream> • Untitled-1 - dsmovie2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
styles.css .../Pagination C++ #include <iostream> Untitled-1 index.tsx .../Pagination styles.css .../MovieStars
50 int prev = 0;
51
52 while(completed != x) {
53     int idx = -1;
54     int mn = 10000000;
55     for(int i = 0; i < x; i++) {
56         if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
57             if(burst_remaining[i] < mn) {
58                 mn = burst_remaining[i];
59                 idx = i;
60             }
61             if(burst_remaining[i] == mn) {
62                 if(p[i].arrival_time < p[idx].arrival_time) {
63                     mn = burst_remaining[i];
64                     idx = i;
65                 }
66             }
67         }
68     }
69
70     if(idx != -1) {
71         if(burst_remaining[idx] == p[idx].burst_time) {
72             p[idx].start_time = current_time;
73             total_idle_time += p[idx].start_time - prev;
74         }
75         burst_remaining[idx] -= 1;
76         current_time++;
77         prev = current_time;
78
79         if(burst_remaining[idx] == 0) {
80             p[idx].completion_time = current_time;
81             p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
82             p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
83             p[idx].response_time = p[idx].start_time - p[idx].arrival_time;
84
85             total_turnaround_time += p[idx].turnaround_time;
86             total_waiting_time += p[idx].waiting_time;
87             total_response_time += p[idx].response_time;
88
89             is_completed[idx] = 1;
90             completed++;
91         }
92     }
93     else {
94         current_time++;
95     }
96 }
97
98 int min_arrival_time = 10000000;
```

Implementação

```
int min_arrival_time = 100000000;
int max_completion_time = -1;
for(int i = 0; i < x; i++) {
    min_arrival_time = min(min_arrival_time, p[i].arrival_time);
    max_completion_time = max(max_completion_time, p[i].completion_time);
}

avg_turnaround_time = (float) total_turnaround_time / x;
avg_waiting_time = (float) total_waiting_time / x;
avg_response_time = (float) total_response_time / x;
cpu_utilization = ((max_completion_time - total_idle_time) /
(float) max_completion_time ) * 100;
throughput = float(x) / (max_completion_time - min_arrival_time);

cout<<endl<<endl;

cout<<"Process\t"<<"Arrival Time\t"<<"Burst Time\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT
\t"<<"\n"<<endl;

for(int i = 0; i < x; i++) {
    cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time
<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"
\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
}

cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
cout<<"Average Response Time = "<<avg_response_time<<endl;
cout<<"CPU Utilization = "<<cpu_utilization<<"%"<<endl;
cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;
```

Multiplas Filas

Permite que os processos sejam agrupados e atribuídos a filas diferentes, com escalonamentos diferentes



Vantagens

- Permite determinar a importância de cada processo e a preferência de execução
- Outros algoritmos podem favorecer uma situação e desfavorecer outra

Desvantagens

Pode causar overhead
(sobrecarga)



Múltiplas filas com realimentação

Permite que os processos mudem de fila ou sejam reescalados na própria fila.

É usado:

- FIFO
- SJF
- RR

Exemplo do cotidiano



OBRIGADA!

