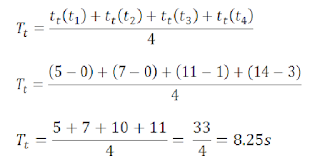
**Algoritmos de escalonamento de processos**

O escalonador é responsável por decidir a ordem de execução dos processos prontos, ou seja, que escalona os processos. O escalonamento de processos é realizado por um algoritmo que visa tratar de forma eficiente e rápida os processos a serem tratados. Vários critérios podem ser definidos para a avaliação dos escalonadores. Os mais frequentes são:

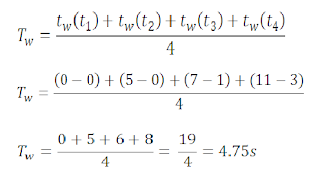
* *tempo de execução (turnaround)* → mede o tempo decorrido entre a criação e o encerramento da tarefa, computando todos os tempos de processamento e de espera.
* *tempo de espera (waiting time)* → tempo total perdido pela tarefa na fila de prontos, aguardando o processador.
* *tempo de resposta* → tempo decorrido entre a chegada de um evento ao sistema e o resultado imediato de seu processamento.
* *Eficiência* → Indica o grau de utilização do processador na execução das tarefas do usuário.
* *Justiça* → Distribuição do processador entre as tarefas prontas.

**Cálculo do tempo médio de execução**

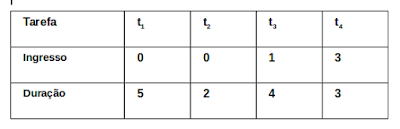
[](http://1.bp.blogspot.com/-pQ3pks-hciw/VmznLFm9M8I/AAAAAAAAAbY/v9xfgTwAgzc/s1600/imagem1.png)

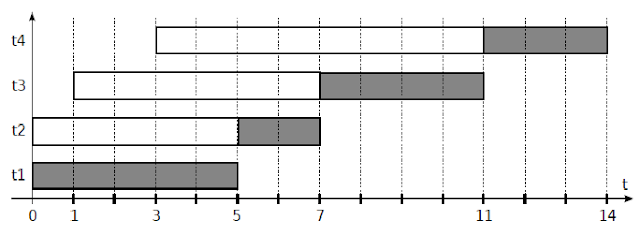
Ou seja, a soma dos tempos de execução de cada processo (tempo final – tempo de ingresso) dividido pela quantidade de processos.

**Cálculo do tempo médio de espera**

[](http://2.bp.blogspot.com/-AsOSskfG63k/VmznNCEcAtI/AAAAAAAAAbg/NuvXpf0KWZQ/s1600/imagem2.png)

Ou seja, a soma dos tempos de espera de cada processo (tempo de inicio da execução – tempo de ingresso) dividida pela quantidade de processos.  
  
  
**Preempção**  
  
O escalonamento de um SO pode ser preemptivo ou não preemptivo.  
Em escalonadores preemptivos, uma tarefa pode perder o processador caso acabe seu quantum, execute uma chamada de sistema ou ocorra uma interrupção que acorde uma tarefa mais prioritária.  
Em escalonadores não-preemptivos, a tarefa permanece no processador tanto quanto possível, só abandonando caso termine de executar, solicite uma operação de E/S ou libere o processador.  
  
  
**Escalonamento FCFC(First-Come, First Served)**  
  
É a forma mais elementar de escalonamento. Utiliza um algoritmo simples que atende as tarefas em sequência assim que ficam prontas. Ou seja, de acordo com sua chegada na fila de prontos (FIFO). Não utiliza preempção.  
 **Exemplo:**

[](http://1.bp.blogspot.com/-mg_fiuyozTA/Vmzn6lPislI/AAAAAAAAAbo/WPRv4qtfeOU/s1600/imagem3.png)

[](http://3.bp.blogspot.com/-cb6hEOpRMZQ/VmzofMGJsJI/AAAAAAAAAb0/07i-gp4-57U/s1600/imagem4.png)

**Implementação em Java**

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Scanner;

public class fcfs {

public static void main(String[] args) {

// declaracao de variaveis

Scanner scanner = new Scanner(System.in);

int N, entrada, tempoAtual;

double tempoExecucao, tempoEspera;

ArrayList processos, ingressos, duracoes, temposFinais, temposIniciais;

int contTeste = 0;

N = scanner.nextInt();

String formato, saida;

DecimalFormat nf = new DecimalFormat("0.00");

while (N != 0) {

contTeste++;

// inicializacao de variaveis

processos = new ArrayList();

ingressos = new ArrayList();

duracoes = new ArrayList();

temposFinais = new ArrayList();

temposIniciais = new ArrayList();

tempoEspera = 0;

tempoExecucao = 0;

for (int i = 1; i <= N; i++) {

// adiciona processo a lista de processos

processos.add(i);

// le e adiciona ingresso no processo i

entrada = scanner.nextInt();

ingressos.add(entrada);

// le e adiciona duracao do processo i

entrada = scanner.nextInt();

duracoes.add(entrada);

}

// tempo inicial = tempo de ingresso do primeiro processo

tempoAtual = ingressos.get(0);

// adicionando tempos de inicio e termino dos processos

for (int i = 0; i < N; i++) {

if (ingressos.get(i) > tempoAtual) {

tempoAtual = ingressos.get(i);

}

temposIniciais.add(tempoAtual);

tempoAtual += duracoes.get(i);

temposFinais.add(tempoAtual);

}

// calculo dos tempos medios de espera e execucao

for (int i = 0; i < N; i++) {

tempoExecucao += temposFinais.get(i) - ingressos.get(i);

tempoEspera += temposIniciais.get(i) - ingressos.get(i);

}

tempoExecucao = tempoExecucao / N;

tempoEspera = tempoEspera / N;

System.out.println("Teste " + contTeste);

formato = nf.format(tempoExecucao);

saida = "Tempo medio de execucao: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

formato = nf.format(tempoEspera);

saida = "Tempo medio de espera: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

// ordem dos processos = em tempo de entrada

for (int i = 0; i < N; i++) {

System.out.print("P" + processos.get(i) + " ");

}

System.out.println();

System.out.println();

N = scanner.nextInt();

}

}

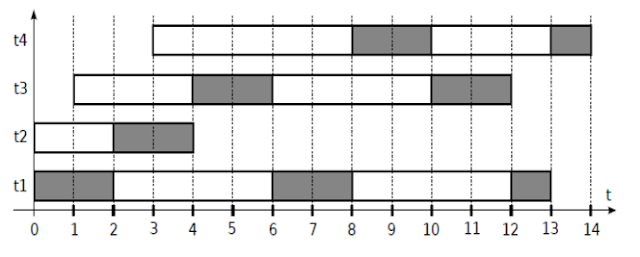
}

**Escalonamento Round-Robin**

O FCFS não leva em conta a importância das tarefas nem seu comportamento em relação aos recursos. É o algoritmo FCFS com a adição de preempção por tempo ao escalonamento. Define um tempo de quantum.

**Exemplo:**

*tempo de quantum = 2*

[](http://2.bp.blogspot.com/-qpxzlc1Jm0I/Vm19fguIVpI/AAAAAAAAAco/9ipZ-lnV7BM/s1600/imagem5.png)

**Implementação em Java**

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Scanner;

public class round {

@SuppressWarnings("unchecked")

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// declaracao de variaveis

int quantum, N, entrada, tempoAtual, execucao, q, temposFinais[], qteprocessos, novaDuracao, temposExecucao[];

ArrayList ingressos, duracoes, processos, cpingressos, cpduracoes;

String ordem;

double tempoMedioExecucao, tempoMedioEspera;

int contTeste = 0;

String formato, saida;

DecimalFormat nf = new DecimalFormat("0.00");

quantum = scanner.nextInt();

N = scanner.nextInt();

while (N != 0) {

contTeste++;

processos = new ArrayList();

ingressos = new ArrayList();

duracoes = new ArrayList();

ordem = "";

qteprocessos = N;

temposFinais = new int[N];

temposExecucao = new int[N];

// lendo os processos

for (int i = 0; i < N; i++) {

entrada = scanner.nextInt();

ingressos.add(entrada);

entrada = scanner.nextInt();

duracoes.add(entrada);

}

cpingressos = (ArrayList) ingressos.clone();

cpduracoes = (ArrayList) duracoes.clone();

// int tmpInicial = ingressos.get(0);

tempoAtual = ingressos.get(0);

processos = new ArrayList();

processos = new ArrayList();

while (qteprocessos > 0) {

// verifica antes de iniciar a execucao de um processo

for (int i = 0; i < N; i++) {

if (ingressos.get(i) != -1

&& ingressos.get(i) <= tempoAtual) {

processos.add(i);

ingressos.set(i, -1);

}

}

if (processos.isEmpty()) {

tempoAtual++;

} else {

execucao = processos.remove(0);

ordem += "P" + (execucao + 1) + " ";

q = quantum;

while (q > 0 && duracoes.get(execucao) > 0) {

tempoAtual++;

q--;

novaDuracao = duracoes.get(execucao) - 1;

duracoes.set(execucao, novaDuracao);

}

if (duracoes.get(execucao) > 0) {

// verificar primeiramente se algum processo entrou

// durante

// o

// tempo de execucao

for (int i = 0; i < N; i++) {

if (ingressos.get(i) != -1

&& ingressos.get(i) <= tempoAtual) {

processos.add(i);

ingressos.set(i, -1);

}

}

processos.add(execucao);

} else {

// processo acabou

temposFinais[execucao] = tempoAtual;

qteprocessos--;

}

}

}

// calculo de tempo medio de espera e execucao;

tempoMedioExecucao = 0;

tempoMedioEspera = 0;

for (int i = 0; i < N; i++) {

temposExecucao[i] = temposFinais[i] - cpingressos.get(i);

tempoMedioExecucao += temposExecucao[i];

tempoMedioEspera += temposExecucao[i] - cpduracoes.get(i);

}

tempoMedioExecucao = tempoMedioExecucao / N;

tempoMedioEspera = tempoMedioEspera / N;

System.out.println("Teste " + contTeste);

formato = nf.format(tempoMedioExecucao);

saida = "Tempo medio de execucao: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

formato = nf.format(tempoMedioEspera);

saida = "Tempo medio de espera: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

System.out.println(ordem);

System.out.println();

N = scanner.nextInt();

}

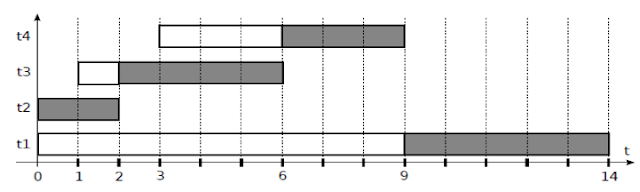
}

}

**Escalonamento SJF**

O algoritmo de escalonamento que proporciona os menores tempos médios de execução e de espera é conhecido como menor tarefa primeiro, ou SJF (Shortest Job First). Consiste em atribuir o processador à menor (mais curta) tarefa da fila de tarefas prontas.

**Exemplo:**

[](http://4.bp.blogspot.com/-hzHCUY5w318/Vm19wIEPyKI/AAAAAAAAAcw/fyCZjwDbQiU/s1600/imagem6.png)

**Implementação em Java**

import java.text.DecimalFormat;

import java.text.NumberFormat;

import java.util.ArrayList;

import java.util.Locale;

import java.util.Scanner;

public class sjf {

// versao 2

@SuppressWarnings("unchecked")

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// declaracao de variaveis

int N, entrada;

ArrayList processos, ingressos, cpingressos = new ArrayList(), duracoes;

int[] temposFinais = new int[1], temposIniciais = new int[1];

int idProcessoAtual;

String ordemExecucao = "", formato, saida;

double tempoEspera, tempoExecucao;

int contTeste = 0;

DecimalFormat nf = new DecimalFormat("0.00");

N = scanner.nextInt();

while (N != 0) {

contTeste++;

ordemExecucao = "";

processos = new ArrayList();

ingressos = new ArrayList();

duracoes = new ArrayList();

temposFinais = new int[N];

temposIniciais = new int[N];

for (int i = 0; i < N; i++) {

// le e adiciona tempo de ingresso do processo

entrada = scanner.nextInt();

ingressos.add(entrada);

// le e adiciona tempo de duracao do processo

entrada = scanner.nextInt();

duracoes.add(entrada);

}

// cria copia da lista de tempos de ingressos devido a modificacoes

cpingressos = (ArrayList) ingressos.clone();

int execucao;

int qteprocessos = N;

// tempo inicial = primeiro tempo da lista de ingressos

int tempoAtual = ingressos.get(0);

while (qteprocessos > 0) {

// percorre ingressos para achar processos que ingressam nesse

// tempo

processos = new ArrayList();

for (int i = 0; i < N; i++) {

if (ingressos.get(i) != -1

&& ingressos.get(i) <= tempoAtual) {

// adicionar na lista de processos

processos.add(i);

}

}

// assumindo que o primeiro da lista eh o de menor duracao

if (processos.isEmpty()) {

tempoAtual++;

} else {

execucao = processos.get(0);

for (int i = 0; i < processos.size(); i++) {

idProcessoAtual = processos.get(i);

// se a duracao do processo atual for menor do que a

// menor

// duracao

// ja encontrada

if (duracoes.get(idProcessoAtual) < duracoes

.get(execucao)) {

// entao alteramos o processo que vai executar

execucao = processos.get(i);

}

}

temposIniciais[execucao] = tempoAtual;

tempoAtual += duracoes.get(execucao);

temposFinais[execucao] = tempoAtual;

ingressos.set(execucao, -1);

// define ordem de execucao

ordemExecucao += "P" + (execucao + 1) + " ";

qteprocessos--;

}

}

// calculo tempo de execucao e tempo de espera

tempoExecucao = 0;

tempoEspera = 0;

for (int i = 0; i < N; i++) {

tempoExecucao += temposFinais[i] - cpingressos.get(i);

tempoEspera += temposIniciais[i] - cpingressos.get(i);

}

tempoExecucao = tempoExecucao / N;

tempoEspera = tempoEspera / N;

// DecimalFormat f = (DecimalFormat) DecimalFormat

// .getInstance(new Locale("pt", "BR"));

// String saida = "Tempo medio de execucao: "

// + f.format(tempoExecucao) + "s";

System.out.println("Teste " + contTeste);

formato = nf.format(tempoExecucao);

saida = "Tempo medio de execucao: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

formato = nf.format(tempoEspera);

saida = "Tempo medio de espera: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

System.out.println(ordemExecucao);

System.out.println();

N = scanner.nextInt();

}

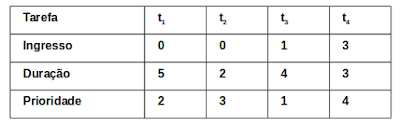
}

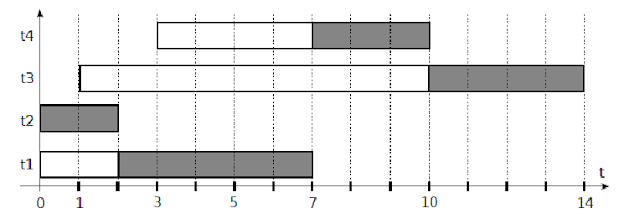
}

**Escalonamento por Prioridades**

No escalonamento por prioridades, a cada tarefa é associada uma prioridade (número inteiro) usada para escolher a próxima tarefa a receber o processador, a cada troca de contexto.

**Exemplo:**

[](http://2.bp.blogspot.com/-SSaofv2R9BY/Vm1-Y-BBATI/AAAAAAAAAc4/4QsJKVdejnI/s1600/imagem7.png)

[](http://1.bp.blogspot.com/-JjqEhv3mqUU/Vm1-Y1M7IHI/AAAAAAAAAc8/Z5m6TxQ-ZX0/s1600/imagem8.png)

**Implementação em Java**

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Scanner;

public class prio {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// declaracao de variaveis

int N, entrada, tempoAtual, execucao, idProcessoAtual, qteprocessos;

ArrayList ingressos, duracoes, prioridades, processos, cpingressos;

int[] temposFinais, temposIniciais;

String ordemExecucao;

int contTeste = 0;

String formato, saida;

DecimalFormat nf = new DecimalFormat("0.00");

N = scanner.nextInt();

while (N != 0) {

contTeste++;

processos = new ArrayList();

ingressos = new ArrayList();

duracoes = new ArrayList();

prioridades = new ArrayList();

// lendo os processos

for (int i = 0; i < N; i++) {

// le e adiciona dados dos processos em suas respectivas listas

entrada = scanner.nextInt();

ingressos.add(entrada);

entrada = scanner.nextInt();

duracoes.add(entrada);

entrada = scanner.nextInt();

prioridades.add(entrada);

}

temposIniciais = new int[N];

temposFinais = new int[N];

// cria copia da lista de tempos de ingressos devido a modificacoes

cpingressos = (ArrayList) ingressos.clone();

ordemExecucao = "";

// tempo comeca do primeiro processo a ingressar

tempoAtual = ingressos.get(0);

qteprocessos = N;

while (qteprocessos > 0) {

// percorrendo ingressos para descobrir processos que entram no

// tempo

// atual

processos = new ArrayList();

for (int i = 0; i < N; i++) {

if (ingressos.get(i) != -1

&& ingressos.get(i) <= tempoAtual) {

// adicionar na lista de processos

processos.add(i);

}

}

if (processos.isEmpty()) {

tempoAtual++;

} else {

// assumindo que o primeiro da lista eh o de menor

// prioridade

execucao = processos.get(0);

for (int i = 0; i < processos.size(); i++) {

idProcessoAtual = processos.get(i);

// se a prioridade do processo atual for menor do que a

// menor

// prioridade

// ja encontrada

if (prioridades.get(idProcessoAtual) < prioridades

.get(execucao)) {

// entao alteramos o processo que vai executar

execucao = processos.get(i);

}

}

// System.out.println("vou executar o P" + (execucao + 1) +

// " de prioridade " + prioridades.get(execucao));

// tempo que o processo comeca a executar

temposIniciais[execucao] = tempoAtual;

tempoAtual += duracoes.get(execucao);

// tempo que o processo termina de executar

temposFinais[execucao] = tempoAtual;

ingressos.set(execucao, -1);

// define ordem de execucao

ordemExecucao += "P" + (execucao + 1) + " ";

qteprocessos--;

}

}

// calculo tempo de execucao e tempo de espera

double tempoExecucao = 0, tempoEspera = 0;

for (int i = 0; i < N; i++) {

tempoExecucao += temposFinais[i] - cpingressos.get(i);

tempoEspera += temposIniciais[i] - cpingressos.get(i);

}

tempoExecucao = tempoExecucao / N;

tempoEspera = tempoEspera / N;

System.out.println("Teste " + contTeste);

formato = nf.format(tempoExecucao);

saida = "Tempo medio de execucao: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

formato = nf.format(tempoEspera);

saida = "Tempo medio de espera: " + formato + "s";

saida = saida.replace(".", ",");

System.out.println(saida);

System.out.println(ordemExecucao);

System.out.println();

N = scanner.nextInt();

}

}

}

## Tipos básicos

## Definição

## Algoritmos de escalonamento

## Escalonamento em Sistemas em Batch

## Estados de processos

## Distribuição de Prioridades

## Trocas de contexto

## Threads