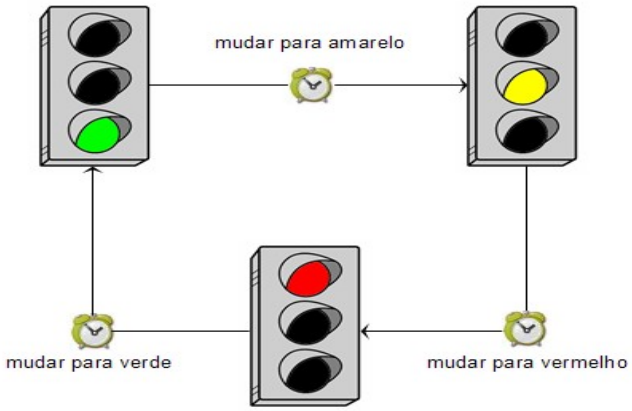
UNIVERSIDADE DE ÉVORA

# Relatório dos trabalhos práticos
# de
# Sistemas Operativos I




Docentes:

Luis Rato

Nuno Miranda

Ricardo Fusco, 29263

Bruno Santos, 29451

**Engenharia Informática – 2012/2013**

# Introdução

Este trabalho tem como objectivo principal elaborar um simulador em Java para a gestão de processos de acordo com o modelo de cinco estados. Neste segundo trabalho o nosso objectivo é melhorar e acabar o primeiro trabalho e tentar fazer o segundo trabalho da melhor maneira possível, utilizado assim os conhecimentos adquiridos ao longo do semestre nas aulas teóricas e práticas.

Após fazermos o primeiro trabalho tentámos alterar a classe PCB e Pipeline para que se adequasse à utilização da memória necessária para este segundo trabalho.

# Desenvolvimento

# Primeiro Trabalho

## Classes Utilizadas:

Começámos por criar a seguinte classe:

PCB – Esta classe cotém as informações relativas a cada processo. Esta cria uma colecção (ArrayList) que contém todas as informações sobre um processo específico.

### Variáveis da Classe:

```
private int PID;
private int estado;
private int dispSeg;
private int tempRun;
private int tempBlock;
private int cicloActual;
private int cicloChegada;
public ArrayList <Integer> pcb = new ArrayList<Integer> ();
```

### Métodos da Classe:

**Public PCB(int pid, int estado) –** Construtor que recebe como argumentos o id do processo e o estado em que este se encontra para que possa ser criado o PCB com tais parâmetros.

**Public void setDispSeg(int n) –** Método modificador para alterar o valor do dispositivo seguinte.

**Public int getDispSeg() -** Método selector que retorna o valor do dispositivo

seguinte.

**Public void setProcid(int n) –** Método modificador para alterar o id do processo.

**Public int getProcid() -** Método selector que retorna o valor do id do processo.

**Public void setEstado(int e) –** Método modificador para alterar o estado do processo.

**Public int getEstado() -** Método selector que retorna o estado do processo.

**Public int getCicloActual() -** Método selector que retorna o valor do ciclo actual.

**Public void setCicloActual(int cicloActual) –** Método modificar para alterar o valor do ciclo actual.

**Public int getCicloChecada() -** Método selector que retorna o valor do ciclo de chegada do processo.

**Public void setCicloChecada(int cicloChegada) –** Método modificador que serve para alterar o ciclo de chegada do processo.

**Public int getTempRun() -** Método selector que retorna o tempo que o processo se fica no estado Run.

**Public void setTempRun(int tempRun) –** Método modificador que serve para alterar o valor do tempo em que o processo se encontra no estado Run, no caso do primeiro trabalho este tinha o valor de 4 ciclos, voltando ao estado Ready se ainda tivesse algum ciclo para ser executado.

**Public int getTempBlock() -** Método selector que retorna o valor do tempo que o processo permanece no estado Block.

**Public void setTempBlock(int tempBlock) –** Método modificador que serve para alterar o tempo em que o processo se encontra no estado Block.

**De seguida foi criada a classe para a transição de estados.**

**Pipeline –** Esta classe contém todos os estados transitórios que cada processo necessita para finalizar a sua execução.

**Variáveis da Classe:**

```
public static final int PRE_ESTADO = 0;
public static final int ESTADO_NEW = 1;
public static final int ESTADO_READY = 2;
public static final int ESTADO_RUN = 3;
public static final int ESTADO_BLOCK = 4;
public static final int ESTADO_EXIT = 5;
public int tempRUN = 4;
public int tempBLOCK = 3;
public int cycle;
public boolean running;
public static LinkedList<PCB>[] estado;
public int numOfProcess;
```

A variável cycle trata-se de uma variável incrementadora do loop principal.

A variável running (booleana) de condição do while, encontra-se na função onde todos os processos estão a correr (função Run). Quando for false significa que todos os processos foram terminados.

## Métodos da classe:

**Public Pipeline(int numP) -**

**Public void run() -** Este método faz com que todos os processos avancem um ciclo.

**public void preEstadoToNew() -** Método que evita que o processo avance para o proximo estado (New) antes da altura correcta.

**public void estadoNewToReady –** Método que evita que o processo avance para o proximo estado (Ready) antes da altura correcta.

**public void estadoReadyToRun –** Método que evita que o processo avance para o proximo estado (Run) antes da altura correcta.

**public void estadoRunToReady() -** Método que evita que o processo avance para o proximo estado (Ready) antes da altura correcta.

**public void estadoBlockToReady() -** Método que evita que o processo avance para o proximo estado (Ready) antes da altura correcta.

**public void estadoExit() -** Método que faz com que o processo saia para o estado Exit, terminando assim a execução deste.

**Por último foi criada uma outra classe denominada como ReadFile.**

A função Read recebe uma String como argumento, essa mesma String é o caminho do ficheiro. Com a ajuda do File e do BufferReader é recebido o input que se encontra no ficheiro de texto. Desse input vai ser verificado linha a linha e na primeira linha que encontrar cria um Array com as informações gerais acerca de todos os processos, para as restantes linhas é feito o split da String, ou seja, vai ver os números separados por espaços e coloca cada número num Array temporário de Strings. Depois é percorrida cada posição desse Array temporário e cada número que encontrar é convertido para inteiro e é colocado no caso da primeira linha no Array sobre os processos, nas restantes linhas é colocado num novo PCB correspondente a

cada processo que existir no ficheiro.

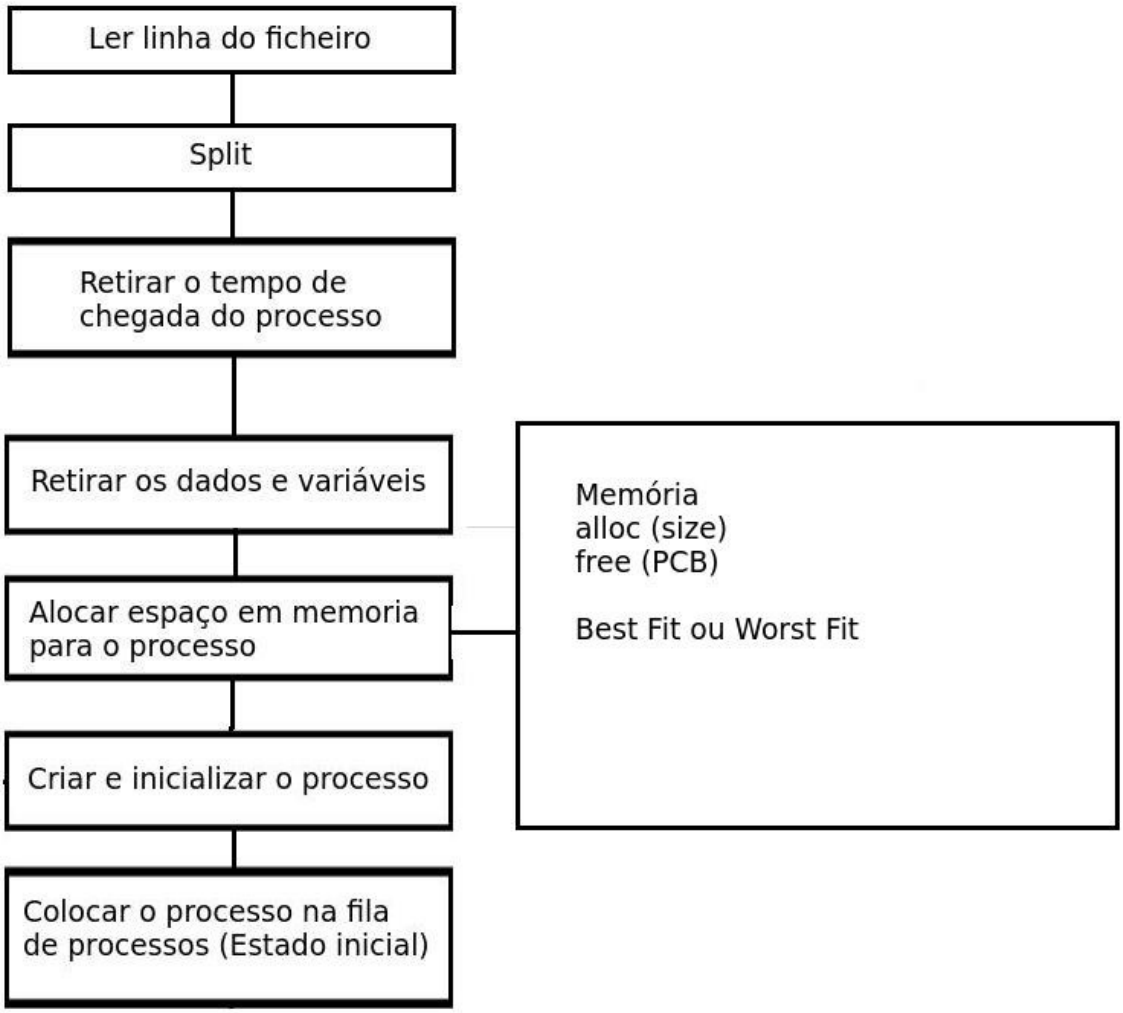
Cada processo criado (PCB) é colocado numa ArrayList chamada processos que irá conter todos os processos.

Caso o ficheiro não se encontre no caminho especificado é lançada uma excepção “FileNotFoundException”.

**public ArrayList<PCB> getListaProcessos() -** Método para retornar a lista dos processos.

**Segundo Trabalho**

## Esquema inicial




As classes PCB e Pipeline são praticamente iguais às classes do primeiro trabalho, tendo apenas mais umas variáveis extra e no caso do PCB não vai ser necessária uma estrutura para guardar informações de um processo visto que os dados vão ser armazenados em memória. Foram adicionadas três variáveis na classe PCB, uma para o Program Counter, outra para o tamanho que o processo irá ocupar em memória e onde vai estar em memória ou onde começa.

Além das classes PCB e Pipeline temos a classe onde se vão processar as trocas de dados chamada GestProg, na qual se encontra a função para ler o ficheiro juntamente

com a função main onde vão ser chamadas as funções do Pipeline.

# Conclusão

Concluímos com este trabalho que conseguimos aplicar alguns dos conhecimentos base necessários para efectuar este trabalho usando a linguagem Java e o IDE Eclipse.

A maior dificuldade que tivémos foi conseguir efectuar de forma satisfatória a transição correcta dos estados apesar de que, a lógica por trás da implementação que fizémos esteja correcta.

Tivémos também algumas dificuldades em conseguir avançar com o segundo trabalho visto que o esgotámos muito do nosso tempo a tentar colocar o primeiro trabalho a funcionar.