Visão Geral

Este código implementa um simulador de escalonamento de processos usando o algoritmo Round Robin com múltiplas filas de prioridade. Este sistema cria processos com valores aleatórios, gerencia relações de parentesco entre processos, e simula a execução de cada um, considerando diferentes prioridades e operações de I/O.

Número de processos: variável (na faixa de 1-10)

Fatia de tempo depende do tipo de mídia usada. Fita magnética, Disco ou Impressora.

Burst time total de 1.2 segundos ou 1200 ms.

Estrutura Principal

Inicialização e Parâmetros:

```
int PLim = 10; // Limite de criação de processos;
int PReq;
int BTLim = 1200; //Limite de Burst Time de 1.2 segundos
```

É definido limites máximos para:

- Número de processos igual a 10 - PLim.
- Tempo de execução (Burst Time) de 1200 ms ou 1.2 segundos - BTLim.

Geração de PID (Identificadores de Processo)

```
short PIDInit = (short) System.nanoTime();
int PIDRand = (PIDInit * 1103515245 + 12345) & Integer.MAX_VALUE;
short PIDGen = (short) (PIDRand % (BTLim + 1)*2);
```

Utiliza o tempo do sistema como seed para gerar PIDs através de um algoritmo pseudo-randômico.

Coleta de Input do Usuário

```
Scanner PReqInput = new Scanner(System.in);
System.out.print(String.format("Quantos processos gostaria de criar? (Max:
%d)\n#> ",PLim));
PReq = PReqInput.nextInt();
```

Solicita ao usuário quantos processos deseja criar, validando se o número está dentro dos limites permitidos.

Geração de Processos

Características de Processos

Para cada processo, são gerados aleatoriamente:

- Burst Time (tempo de execução)
- Tipo de dispositivo I/O (Disco, Fita magnética ou Impressora)
- Relacionamento de parentesco com outros processos (PPID)

Algoritmo de Geração de Números Aleatórios

```
long X = System.nanoTime();
long m = ((BTRVal * 20) * 2003515245 + 401245991) & Integer.MAX_VALUE;
long c = ((m*5) * 510351524 + 901245991) & Integer.MAX_VALUE;
long a = ((m*9) * 903515245 + 6956825) & Integer.MAX_VALUE;
int LCG = (int) ((a * X + c) % m);
/*
 * X -> Números pseudo-randômicos (inicializador)
 * m -> Módulo
 * a -> Multiplicador
 * c -> Incremento
 */
```

Como não era permitido o uso orientação a objeto, foi utilizado Linear Congruential Generator (LCG) para criar valores aleatórios que determinam as características dos processos.

Armazenamento de Informações

```
int[] GuardaInOut = new int[PReq];
int[] GuardaBT = new int[PReq];
int[][] GuardaRelacaoPPID = new int[PReq][];
int[] GuardaPID = new int[PReq];
String[] GuardaProcesso = new String[PReq];
```

Arrays são utilizados para armazenar todas as informações dos processos, como:

- Tempos de I/O
- Burst Times
- Relações de parentesco (PPID)
- PIDs
- Tipo de mídia (Disco, Fita, Impressora)
- Velocidade de mídia
- Prioridade

Sistema de Prioridades

Definição de Prioridades

```
switch(Integer.parseInt(TIPO)){
    case 0:
        PRIO="Baixa";
```

```
        break;
    case 1:
    case 2:
        PRIO="Alta";
        break;
}
```

Prioridade determinada pelo tipo de dispositivo I/O:

- Disco → Prioridade baixa (TQ = 30)
- Fita magnética → Prioridade alta (TQ = 80)
- Impressora → Prioridade alta (TQ = 120)

Organização em Filas

```
P1 = new String[P1Len]; // Fila de alta prioridade
P2 = new String[P2Len]; // Fila de baixa prioridade
int[] FilaIO = new int[GuardaProcesso.length]; // Fila de I/O
```

Os processos são organizados em 3 filas distintas baseadas em prioridade e processos finalizados.

A terceira, FilaIO, é para caso o processo precise de I/O, ficando ali sem usar a CPU até recebê-lo.

Simulação de Execução

Ciclo Principal de Execução

```
while (!todosTerminados) {
    System.out.printf("\n--- Ciclo %d ---\n",ciclo);
    // <Processamento das filas>
    ciclo++;
}
```

O sistema executa em ciclos, processando primeiro todos os processos da fila de alta prioridade (P1) e depois os da fila de baixa prioridade (P2) e por fim a fila de I/Os.

Processamento Individual de Processos

```
for(int j = 0; j < IOTempo; j++) {
    Preempt++;
    BT--;

    if(BT == 0) {
        ProcessosP1[6] = "2"; // Terminou
        FilaIO[FilaIOIDX++] = Integer.parseInt(ProcessosP1[0]);
        break;
    }
```

```
    if(j == IOTempo - 1 && BT > 0) {
        ProcessosP1[3] = String.valueOf(BT);
    }
}
```

Cada processo consome tempo de I/O, dedicado a eles no primeiro loop, a cada ciclo, reduzindo seu Burst Time restante. Quando o Burst Time chega a zero, o processo é marcado como concluído e movido para a fila de I/O.

Relatórios e Métricas

Log de Execução

```
System.out.printf("%-10s %-15s %-15s %-15s %-15s %-15s %-15s %-15s %-15s
%-15s %-15s%n",
                                    "PID",          "Processo          X",
"Tipo","BT","I/O","Parentes","Estado","Prioridade","Execução","BT
Restante","Varreduras");
```

No processo de execução, os processos são todos postos em um loop e iterados, diminuindo o BT do processo e aumentando o tempo de execução.

Caso o BT do processo acabe antes do Tempo Quantum esgotar, ele é marcado como terminado e não será mais executado.

Agora se o BT não acabe antes do Tempo Quantum, o processo vai ser lançado para a fila de IO e dar espaço para outros processos de menor prioridade.

Cálculo de Métricas de Desempenho

```
int tempoTotal = ciclo - 1;
double waitTimeTotal = 0;
double tatTotal = 0;
int processosCompletos = 0;

// Cálculos para waitTime e TAT
double  waitTimeMedio  =  (processosCompletos  >  0)  ?  waitTimeTotal  /
processosCompletos : 0;
double tatMedio = (processosCompletos > 0) ? tatTotal / processosCompletos
: 0;
```

Ao final da execução, o sistema calcula o:

- Tempo total de processamento
- Wait Time médio, ou seja, quanto tempo em média cada processo levou esperando para ser executado.
- Turn Around Time (TAT) médio (tempo total desde a criação até conclusão)