Operating Systems

Laboratory Task No. 2

Note:

- **This lab is to be completed in one week. You are required to submit (upload on moodle) the lab report before or on April 27, 2012 23:59.**
- **This lab is to be performed individually or in a group of 2-3 students. All students should submit their group or individual work.**
- **Please be prepared well before you come to the lab and exercise sessions!! If you have questions, please bring them to the lab session!**
- **There is a sample code at the end of this document. It contains some graphic routines to be discussed below. You may find it useful to extend this program for your own work. You can write your code in any way you want. However, during submission, you are required to put all your source code in one zip file, together with your report. The preferred format for the report is MS-Word.**

Introduction

In this lab, you are required to experiment and understand CPU scheduling. You shall also evaluate different scheduling algorithms based on the concepts you learned in the lectures. All information you need for the lab is found in Chapter 5 of the text book. As you may have read, scheduling is a very important component of the operating system. In particular multiprogramming/multi tasking operating systems cannot be realized without a scheduler.

There are several algorithms used in CPU scheduling. These algorithms have their own strengths and weaknesses in different scenarios. One of the tasks in this lab is to evaluate these algorithms. You are required to evaluate the following scheduling algorithms:

1) First come first served
2) Shortest job first
3) Shortest remaining job first (without pre-emption)
4) Round robin (without pre-emption)

If you run the attached program, you will see horizontal lines drawn on the screen. These lines show the submission (arrival) time of a process and its task duration. If you click on the 'Show Burst' button, you will see that the lines are divided into red and blue segments. The red shows CPU bursts, and the blue shows IO bursts. During the IO burst part of a job execution, the CPU becomes idle. It can therefore be assigned to run one or more of the tasks on the process queues. However, for the mandatory tasks of this lab, you ignore the two parts and treat the entire duration as the task length.

Preparation for the Lab
You should have read and understood about scheduling algorithms in your text book (Chapter 5) before starting the lab. Most of the information you need for this lab is available there. Additionally, you should run the attached code and try to get some idea about process arrival and execution times.

The Windows Task Manager gives you some useful hint about the workings of the Windows scheduler. In Linux there is a GUI based application with similar information. You can also use the **top** command and make observations about the output.

The process arrival is modeled using Poisson's distribution. If you are not familiar with probability distribution functions, just use the function **poissonDF** in the given code only. If you are interested, you can change the value of Lambda and get different process arrival distributions.

A. Task 2.1

1. Write a program to implement the first come first served (FCFS) and shortest job first (SJF) algorithms. Your program should show, the following for each process
   - the arrival time,
   - the execution start time
   - the wait time
   - the response time
   - the turn-around time

2. Your program should compute average values for the wait, response and turn- around times. Compare the results of the two algorithms and explain what you observe.

B. Task 2.2

1. Implement the shortest-remaining-job-first algorithm without pre-emption. Compare your result with the SJF algorithm you implemented earlier and comment on the differences you observe.

2. Implement the round robin (RR) scheduling algorithm. Evaluate the algorithm for different values of the CPU quantum. Use values such as 10, 20, 50, 100. Observe and explain the effect of different quantum durations.

3. Compute the average waiting time and response times for all algorithms.

C. Task 2.3

Implement the algorithms with pre-emption. For this purpose, you can introduce the concept of priorities. Assume that the arriving processes have different priority values (based on importance) (H=high, L=low) and accordingly placed in separate queues. Higher priority processes pre-empt the lower priority ones.

Optional Tasks

1. Observe the effect of context switching time. Take different values for context switching times (1,5, 10) and discuss the effect of this on the efficiency of the different scheduling algorithms.

2. Extend the scheduler for multi-processing systems. Assume that your computer has two cores. (This is common in modern computers, as multi-core processors have now become the standard configuration for most desktop and server systems)

3. Give a nice look to the program through a GUI. Try to animate the scheduling process, so that users can have better grasp of what is really happening.

Examination

1. This lab carries a maximum of 10 points. A minimum of 7 points is required to get a passing grade in this lab. Your work is graded as follows:

   - Task A (2.1) 4points
   - Task B (2.2) 4 points
   - Task C (2.3) 2 pints

2. In order to get the full points for each task,

   - You must have uploaded your program code and a written report on the course page on moodle;
   - Your program must be working correctly;
   - Your code is understandable (structured and clear);
   - You have documented your program in the report clearly.
   - You should be able to present your lab work answer questions orally.

3. The oral presentation is individual based and not in group. If you work on this lab in a group, each group member must understand the common work and be able to explain it independently. After you have uploaded your reports, the instructor will go through them and arrange a time for you to present.

Sample Program Code

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.math.*;

public class Scheduler {
/*
 * author Baran
 * GUI Based Scheduler
 */

 static class MainWindow extends JPanel implements ActionListener {
          JButton burstButton, exitButton;
          JLabel cmdLabel1;
          JFrame frame;
          Graphics G;
          long[] ptime = new long[100];
          int [] plen = new int[100];
          int choice = 0;
       final int jobSize = 20;
       int yscale = 1;

          public MainWindow(JFrame frame){
                   this.frame = frame;
                   burstButton = new JButton("     Show Burst");
                   exitButton  = new JButton("Exit Scheduler");
             this.add(burstButton);
                   burstButton.addActionListener(this);
                   this.add(exitButton);
                   exitButton.addActionListener(this);
             yscale = (int) (600/jobSize);
          }

          public void paintComponent(Graphics g) {
                G = g;
                if (choice == 0) drawProcess(g);
                if (choice == 1) drawBurst();
                choice = 2;
             }
          public void actionPerformed(ActionEvent e) {
                   String s = "";
                   if (e.getSource() == burstButton) {
```

```
                choice = 1;
                drawBurst();
            }
            if (e.getSource() == exitButton) {
                System.exit(0);
            }
}

private long poissonDF(double lambda){
        double L, p;
        long k=0;
        L = Math.exp(-lambda);
        p = 1;
             while (p > L) {
                    k = k + 1;
                    p = p *  Math.random();
        }
        return k;
}

private void drawBurst(){
        int[] cpuT =  new int[15];
        int [] ioT = new int[15];
        String s;
        int burstNo, burstSize, cpuBurst, x1, x2, y;
        G.setColor(Color.WHITE);
        G.fillRect(1, 1, 800, 800);
        G.setColor(Color.RED);

        //Draw Legend and axes
        x1 = 550;
        x2 = 600;
        y = 70;
        G.drawLine(x1, y, x2, y);
        G.drawLine(x1, y+1, x2, y+1);
        G.drawString("CPU Burst", x2+5, y+5);
        G.drawLine(700, 35, 750, 35);
        G.drawString("> Time", 750, 40);

        //Draw time axis
        G.setColor(Color.RED);
        G.drawLine(10, 60, 800, 60);
        G.setColor(Color.BLACK);
        for (int i=0; i<10; i++)
            G.drawLine(i*50+10, 55, i*50+10, 800);

        for (int i=0; i<5; i++) {
```

```
        s = i*100 + "";
        G.drawString(s, i*100, 50);
    }
    G.setColor(Color.BLUE);
    y = 90;
    G.fillRect(x1, y, x2-x1, 5);
    G.drawString("IO Burst", x2+5, y+5);
    for (int i=0; i<jobSize; i++){
        if (plen[i] > 500)
            burstNo = 5 + (int) (5*Math.random());
        else
            burstNo = (int) (6*Math.random());
        for (int j=0; j<burstNo; j++) {
            burstSize = (int) (plen[i]/(2*burstNo));
            cpuBurst = (int) (burstSize * (1+0.3 * Math.random()));
            cpuT[j] = cpuBurst;
            ioT[j] = (2*burstSize) - cpuBurst;
        }
       G.setColor(Color.RED);
       x1 = 10 + (int) ptime[i];
       x2 = x1 + cpuT[0];
       y = 100 + i*yscale;
       G.fillRect(x1, y, x2-x1, 5);
       G.drawLine(x1, y+1, x2, y+1);
       G.setColor(Color.BLUE);
       for (int j=0; j<burstNo; j++) {
            G.setColor(Color.BLUE);
            x1 = x2;
            x2 = x1 + ioT[j];
            y = 100 + i*yscale;
            G.fillRect(x1, y, (x2-x1),5);

            G.setColor(Color.RED);
            x1 = x2;
            x2 = x1 + cpuT[j+1];
            y = 100 + i*yscale;
            G.fillRect(x1, y, (x2-x1), 5);
                    }
        }
    repaint();
}

private void drawProcess(Graphics g){
        double lambda= 8;

        ptime[0] = 0;
        for (int i=1; i<20; i++) {
```

```
                    ptime[i] = ptime[i-1] + poissonDF(lambda);
                }

//              for (int i=0; i<10; i++) plen[i] = 1000;
                for (int i=0; i<20; i++) {
                    if (i%10 == 0) plen[i] = (int) (50 + 200 * Math.random());
                    if (i%10 == 1) plen[i] = (int) (50 + 80 * Math.random());
                    if (i%10 == 2) plen[i] = (int) (50 + 120 * Math.random());
                    if (i%10 == 3) plen[i] = (int) (150 + 150 * Math.random());
                    if (i%10 == 4) plen[i] = (int) (50 + 50 * Math.random());
                    if (i%10 == 5) plen[i] = (int) (100 + 100 * Math.random());
                    if (i%10 > 5) plen[i] = (int) (50 + 100 * Math.random());
            }
            for (int i=0; i<jobSize; i++)
            System.out.print(ptime[i] + "-" + plen[i] + " ");
          g.drawLine(10, 1, 10, 800);
          g.drawLine(700, 10, 750, 10);
          g.drawString("> Time", 750, 15);

                int x1, x2, y;
                for (int i=0; i<jobSize; i++) {
                    x1 = 10 + (int) ptime[i];
                    x2 = x1 + plen[i];
                    y = 100 + i*yscale;
                    g.drawLine(x1, y, x2, y);
                }
            }
}

public static void main(String[] args) throws java.io.IOException {
            JFrame frame = new JFrame("CPU Scheduling");
            MainWindow mWind = new MainWindow(frame);
            frame.getContentPane().add(mWind);
            frame.setSize(800,600);
            frame.setVisible(true);
            frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
                }
            });
   }
}
```