A Simple Uniprocessor Scheduling GUI Simulator for RMS and EDF

GUI Tool Implementation and Analysis

Zhanghao Wen
CPR E 558– Section 1
Fall 2020
Department of Electrical and Computer Engineering
Iowa State University

1. Introduction

In this project, a GUI Simulator for RMS and EDF algorithm in a simple uniprocessor scheduling is developed by Java in Eclipse IDE.

The main program takes task sets and parameters from user input, and then does scheduling analysis, constructs scheduler in the form of a graph, and provides text feedback for uniprocessor scheduling.

The evaluation and test include all possible scenarios for RMS and EDF such as text feedback when utilization test fails, GUI display when utilization test passed, and user input task set numbers.

How to run

- The program can be run in standard Eclipse IDE.
- After setting up the project, simply run "Scheduler.java" file and follow instructions in the console.
- If schedulable by the user's desired algorithm, a scheduler graph will be displayed.

2. Implementation

2.1 Introduction to all object class:

RMSTask.java: this object is mainly used for the tasks in RMS algorithm. It includes computation time, period.

EDFTask.java: this task is mainly for the EDF algorithm. It includes properties such as computation time, period, period count that indicates current computation cycle, last period end time that serve as next cycle available starting time, deadline which is used for comparing task priority, color that indicates this task on the graph.

One of the two main logic for this program is function update() that is called whenever is this task's turn.

```
public void update() {
    remainComputationTime--;
    if (remainComputationTime == 0) {
        periodConut++;
        currentDeadline = period * periodConut;
        remainComputationTime = computationTime;
    }
}
```

Another main logic is to know whether this.task is the next turn or not. it depends on its remaining computation time, current time unit as a parameter, deadline, and available starting time (last period time).

```
public boolean getCanDraw(int timeUnit) {
    if (this.remainComputationTime != 0 && timeUnit < getCurrentDeadline() && timeUnit >= lastPeriodTime()) {
        return true;
    }
    return false;
}
```

Rectangle.java: this object is used for drawing rectangles on canvas representing tasks. It has an index that indicates position on coordinate on canvas and color that is based on task.

Scheduler.java (Main program):

User interaction: the program asks user how many tasks in the task set. The program also asks user to enter task set details in the format of computation time followed by ',' and then followed by task period, then use a semicolon to separate different tasks.

Use ArrayList<task> to store all the task parameters. Use string to store user's choice for algorithm. And based on later input "RMS" or "EDF", one of the two branches is executed.

2.1 RMS operation:

```
public static void RMSOperation(ArrayList<Task> taskList) {
    if (RMSEDFSchedubilityCheck(taskList, "RMS")) {
        System.out.println("this task set is schedulable by RMS utilization test");
        // draw graph
        drawGraphFunction(taskList);
        System.out.println("\nScheduler graph is showing......");
    } else if (exactAnalysis(taskList)) {

        System.out.println("this task set is schedulable by exact analysis");
        // draw graph
        drawGraphFunction(taskList);
        System.out.println("\nScheduler graph is showing......");
    } else {
        System.out.println("this task set is NOT schedulable !");
    }
}
```

2.2 EDF operation:

```
public static void EDFOperation(ArrayList<Task> taskList) {
    if (RMSEDFSchedubilityCheck(taskList, "EDF")) {
        System.out.println("This task set is schedulable!");
        // draw graph
        drawEDFGraphFunction(taskList);
        System.out.println("\nScheduler graph is showing......");
    } else {
        System.out.println("this task set is NOT schedulable !");
    }

}
```

2.3 Test Method:

As shown below, based on the different algorithm, EDF and RMS use different target values to compare. For EDF, the target value is 1. If RMS, the target value is Math.pow(2, 1.0 / numberOftask) – 1.

```
public static boolean RMSEDFSchedubilityCheck(ArrayList<Task> taskList, String algorithm) {
    int numberOftask = taskList.size();
    double value1 = 0;
    if (algorithm.equals("EDF")) {
        value1 = 1;
    } else if (algorithm.equals("RMS")) {
        value1 = numberOftask * (Math.pow(2, 1.0 / numberOftask) - 1);
    }

    DecimalFormat df = new DecimalFormat("#.####");
    double value2 = 0;
    for (int i = 0; i < numberOftask; i++) {
        value2 = value2 + 1.0 * taskList.get(i).getComputationTime() / (1.0 * taskList.get(i).getPeriod());
    }
    if (value2 <= value1) {

        System.out.println(
                algorithm + "Schedubility Check passed! Because " + df.format(value2) + " <= " + df.format(value1));
        return true;
    } else {
        System.out.println("Failed " + algorithm + " Schedubility Check test because " + df.format(value2) + " > "
                + df.format(value1));
        return false;
    }

}
```

If EDF passed this test, go to draw graph function immediately. Otherwise, return false and output the reason that failed this test.

If RMS is not passed, use exact analysis to check schedulability again.

```
public static boolean exactAnalysis(ArrayList<Task> taskList) {
    // find biggest period
    System.out.println("Try Exact Analysis");
    int biggestPeriod = 0;
    for (int i = 0; i < taskList.size(); i++) {
        if (biggestPeriod < taskList.get(i).getPeriod()) {
            biggestPeriod = taskList.get(i).getPeriod();
        }
    }
    System.out.println("The biggest task period is:" + biggestPeriod);

    // exact analysis step by step
    int newT = 0;
    int lastT = 0;
    int counter = -1;
    while (newT <= biggestPeriod) {
        lastT = newT;
        newT = exactAnalysisEquation(taskList, lastT);
        counter++;
        System.out.println("t_" + counter + " is:" + newT);
        if (newT == lastT) {
            System.out.println("Schedulable because " + "t_" + counter + " = " + "t_" + --counter + " = " + newT
                    + " < " + biggestPeriod);
            return true;
        }
    }
    if (newT > biggestPeriod) {
        System.out.println("Not schedulable since t_" + counter + " = " + newT + " > " + biggestPeriod);
        return false;
    }
    return false;
}
```

```java
public static int exactAnalysisEquation(ArrayList<Task> taskList, int time) {
    int numberOftask = taskList.size();
    int newTime = 0;

    if (time == 0) {
        for (int i = 1; i <= numberOftask; i++) {
            newTime = newTime + taskList.get(i - 1).getComputationTime();
        }
    } else {
        for (int i = 1; i <= numberOftask; i++) {
            newTime = (int) (newTime + taskList.get(i - 1).getComputationTime()
                    * Math.ceil(1.0 * time / taskList.get(i - 1).getPeriod()));
        }

    }

    return newTime;

}
```

2.4 Draw Graph:

Use another Canvas class that extends JComponent and set up properties like LCM, and paint coordinate first:

```java
public void setProps(ArrayList<Task> list, String algorithm) {
    for (int i = 0; i < list.size(); i++) {
        taskList.add(list.get(i));
    }
    numberOfTask = taskList.size();
    this.algorithm = algorithm;
    // find LCM from all period
    for (int i = 0; i < taskList.size(); i++) {
        LCM = findLCM(taskList.get(i).getPeriod(), LCM);
    }

    repaint();// mark this component to be repainted
}

public void paint(Graphics g) {

    // Get the current size of this component
    Dimension d = this.getSize();
    int width = d.width;
    int height = d.height;

    // draw a centered horizontal line
    Graphics2D g2 = (Graphics2D) g;
    // draw in black
    g.setColor(Color.BLACK);
    g2.setStroke(new BasicStroke(3));
    // draw coordinate
    g2.drawLine(50, height / 2, width - 50, height / 2);
    double unitLength = (width - 100) / LCM;
    for (int i = 0; i <= LCM; i++) {
        g2.drawLine((int) (50 + i * unitLength), height / 2, (int) (50 + i * unitLength), height / 2 - 10);
        g2.drawString(new Integer(i).toString(), (int) (50 + i * unitLength) - 4, height / 2 + 15);
```

The canvas size can dynamically change corresponds to the window size.
Since no matter RMS or EDF, the LCM is the same. We have to draw LCM time unit with LCM rectangles. A color array up to LCM is created to be filled:

```java
for (int i = 0; i < LCM; i++) {
    g2.setColor(myfinalarray[i]);
    g2.fillRect((int) (50 + i * unitLength), height / 2 - (int) unitLength, (int) unitLength, (int) unitLength);
}
```

For both RMS and EDF, is no color is added into the color array, then it means idle time:

```
for (int i = 0; i < colors.length; i++) {
    if (colors[i] == null) {
        // colors[i] = Color.WHITE; // cover all idle time slot to color white
        colors[i] = UIManager.getColor("Panel.background");
    }
    // System.out.println(colors[i]);
}
```

2.5 Draw RMS algorithm:

The following code is in a while loop. Since the priority is fixed and static, the basic idea is to draw the highest priority task first, and then the second-highest priority task, and so on. If a lower priority task encounters the situation when the time slot is taken by a higher priority task, it finds the next available time slot and secures that slot. This works because it already passed the exact analysis so no task will miss its deadline.

```
currentLowPTaskIndex = 0;
// find smallest period which is highest priority
int period = taskListforDraw.get(0).getPeriod();
for (int i = 0; i < taskListforDraw.size(); i++) {
    if (period > taskListforDraw.get(i).getPeriod()) {
        period = taskListforDraw.get(i).getPeriod();
        currentLowPTaskIndex = i;
    }
}

// fill in highest priority task RMS
int periodTimes = LCM / period;
for (int i = 0; i < periodTimes; i++) {
    for (int j = i * period; j < taskListforDraw.get(currentLowPTaskIndex).getComputationTime()
            + i * period; j++) {
        if (colors[j] == null) { // if index is not taken by higher priority value
            colors[j] = randomColor;
        } else {
            // check next available index
            int index = checkNextNullIndex(j, colors);
            colors[index] = randomColor;
        }
    }
}
// remove current task so iterate to next task in task list, if list size reduce
// to 0, end while loop
taskListforDraw.remove(currentLowPTaskIndex);
```

2.6 Draw EDF algorithm:

Compared to draw the RMS algorithm, this algorithm is more complicated. Since we cannot draw a block of time unit as RMS does. For each time unit one by one, we have to determine the highest priority task, and whether that task can draw that time slot, means that the remaining computation time is not zero, it passed last period's finish time.

```java
public int findSmallestDeadlineAndCanRraw(ArrayList<testEDFTask> EDFtaskList, int timeUnit) {

    int index = 0;
    int deadline = 999;
    for (int i = 0; i < EDFtaskList.size(); i++) { // find among all tasks

        if (EDFtaskList.get(i).getCanDraw(timeUnit)) {
            if (deadline > EDFtaskList.get(i).getCurrentDeadline()) {
                deadline = EDFtaskList.get(i).getCurrentDeadline();
                index = i;
            }
        }
    }
    return index;
}
```

```java
Color[] colors = new Color[LCM];
for (int i = 0; i < LCM; i++) { // i is current time

    int index = findSmallestDeadlineAndCanRraw(testEDFtaskList, i);
    if (testEDFtaskList.get(index).remainComputationTime != 0
            && i < testEDFtaskList.get(index).getCurrentDeadline()
            && i >= testEDFtaskList.get(index).lastPeriodTime()) {
        testEDFtaskList.get(index).update();
        colors[i] = testEDFtaskList.get(index).getColor();

    } else {
        colors[i] = UIManager.getColor("Panel.background");
    }

}
```

3. Screenshots and Testing

3.1 General interaction process:

First, the program asks you how many tasks in your task set. After answering that, program asks you what algorithm the user wants to pick. Only two options provided: RMS or EDF. After user picks algorithm, user is required to enter task set details.

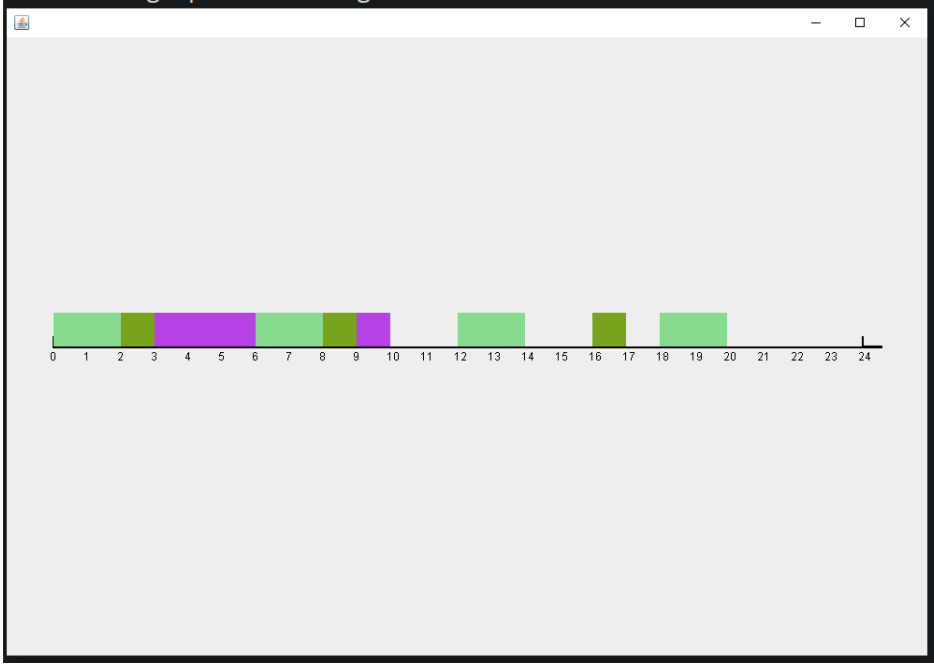
For example, to represent {(1, 8), (2, 6), (4, 24)}, user needs to enter 1,8;2,6;4,24
Then no more input will be required from user.
Program will check RMS utilization test or EDF schedulability check based on the algorithm option. If schedulable based on the given algorithm and task set, then program will display computation results and showing the scheduler graph.

```
How many task? 2
RMS or EDF?
RMS
You picked RMS

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
1,5;2,5
Task set is:
(1,5)  (2,5)  RMSSchedubility Check passed! Because 0.6 <= 0.8284
this task set is schedulable by RMS utilization test

Scheduler graph is showing......
```

3.2 RMS test

1. {T1,T2,T3} = {(1, 8), (2, 6), (4, 24)}.

```
How many task? 3
RMS or EDF?
RMS
You picked RMS

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
1,8;2,6;4,24
Task set is:
(1,8)  (2,6)  (4,24)  RMSSchedubility Check passed! Because 0.625 <= 0.7798
this task set is schedulable by RMS utilization test

Scheduler graph is showing......
```

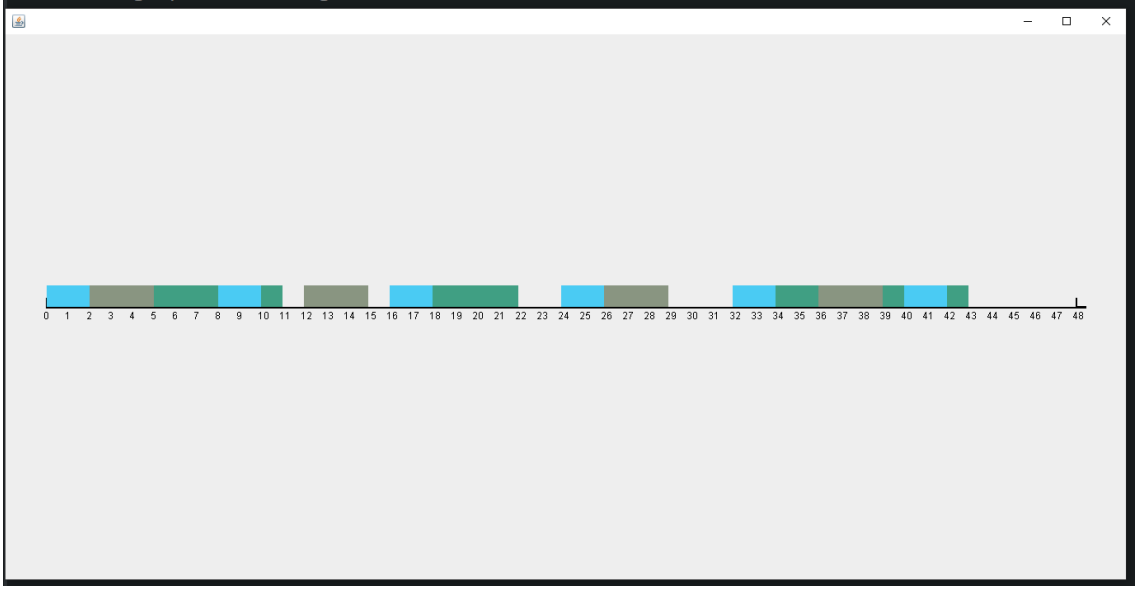


2. {T1,T2,T3} = {(3, 12), (3, 12), (8, 16)}.
   In case not schedulable, no graph will display and only test results will show in console.

```
How many task? 3
RMS or EDF?
RMS
You picked RMS

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
3,12;3,12;8,16
Task set is:
(3,12)  (3,12)  (8,16)  Failed RMS Schedubility Check test because 1 > 0.7798
Try Exact Analysis
The biggest task period is:16
t_0 is:14
t_1 is:20
Not schedulable since t_1 = 20 > 16
this task set is NOT schedulable !
```

3. {T1,T2,T3} = {(2, 8), (3, 12), (4, 16)}.

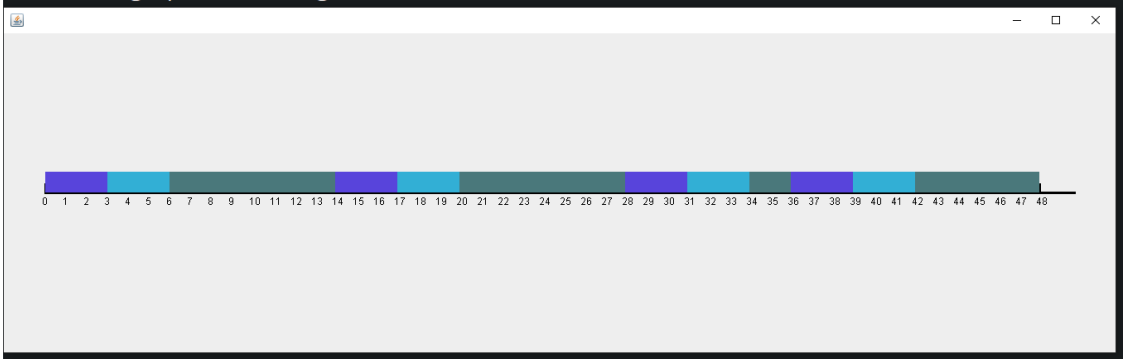
```
How many task? 3
RMS or EDF?
RMS
You picked RMS

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
2,8;3,12;4,16
Task set is:
(2,8)  (3,12)  (4,16)  RMSSchedubility Check passed! Because 0.75 <= 0.7798
this task set is schedulable by RMS utilization test

Scheduler graph is showing......
```



3.3 EDF Test

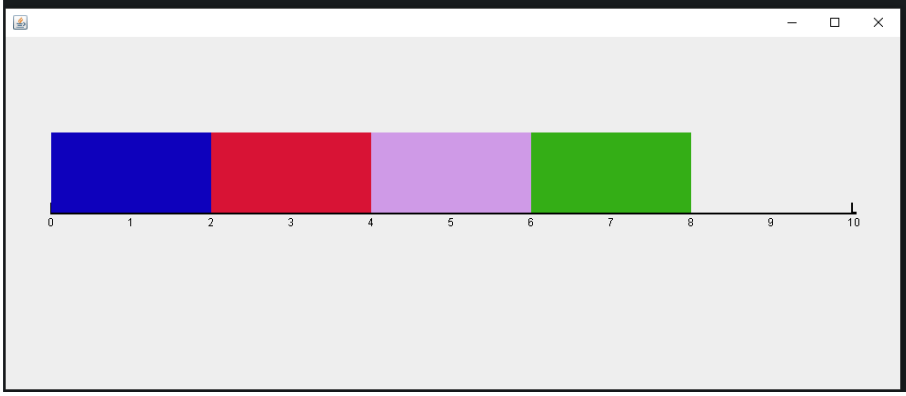
1. {T1,T2,T3} = {(3, 12), (3, 12), (8, 16)}.

```
How many task? 3
RMS or EDF?
EDF
You picked EDF

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
3,12;3,12;8,16
Task set is:
(3,12)  (3,12)  (8,16)  EDFSchedubility Check passed! Because 1 <= 1
This task set is schedulable!

Scheduler graph is showing......
```



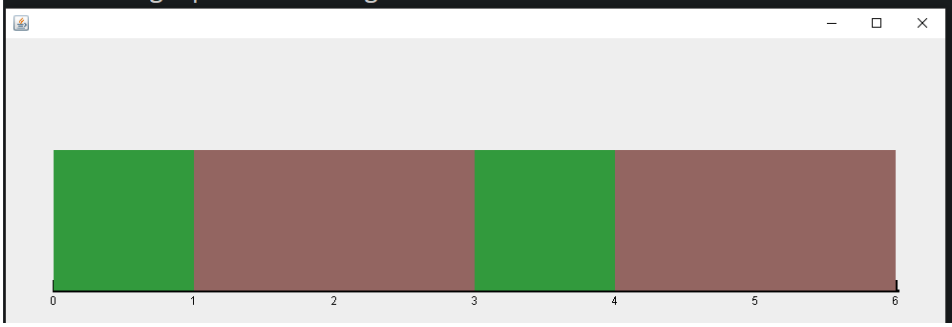
2. {T1,T2,T3,T4} = {(1, 5), (1, 5), (1, 5), (1, 5)}

```
How many task? 4
RMS or EDF?
EDF
You picked EDF

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
2,10;2,10;2,10;2,10
\Task set is:
(2,10)  (2,10)  (2,10)  (2,10)  EDFSchedubility Check passed! Because 0.8 <= 1
This task set is schedulable!

Scheduler graph is showing......
```



3. {(1,3),{4,6}}

```
How many task? 2
RMS or EDF?
EDF
You picked EDF

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
1,3;4,6
Task set is:
(1,3)  (4,6)  EDFSchedubility Check passed! Because 1 <= 1
This task set is schedulable!

Scheduler graph is showing......
```



Schedulability Test

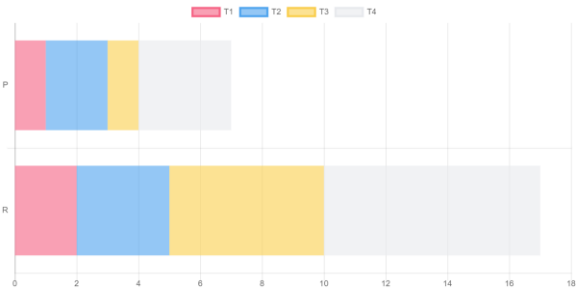
1. EDF test for {(3,8),(2,3)}. In this case, sums of Ci/Pi is bigger than 1, therefore not schedulable.

```
How many task? 2
RMS or EDF?
EDF
You picked EDF

Enter task set computation time and period in format (assume 3 tasks): c1,p1;c2,p2;c3,p3
3,8;2,3
Task set is:
(3,8)  (2,3)  Failed EDF Schedubility Check test because 1.0417 > 1
this task set is NOT schedulable !
```

2. RMS test for {(1,5),(2,5)}. To reduce the computation time, RMS will first check utilization test before doing exact analysis as this is more time efficient.

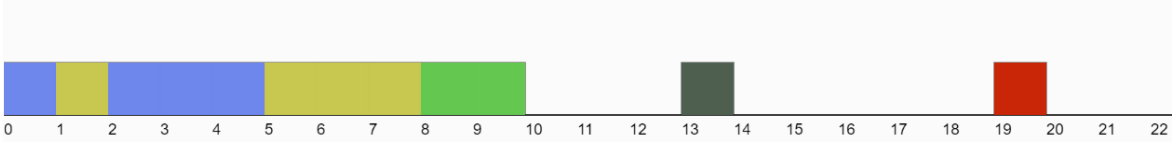
4. Design Progress from Beginning:

First choice is using stacked-bar-chart framework because the provided UI is simple, clear, and elegant. The horizontal coordinate is provided so we can use that as canvas to draw our scheduler. Ng2-Charts and framework of Angular, and Vue are tried to be combined.



I have successfully integrated Ng2-Charts and Angular locally and be able to run the program. However, as the implementation process continues, they are not convenient as it is not easy for you to modify the basic structures of chart by using advanced framework. Because the established data structure and inner functions are all set, facing complicated API document is not really worth the efforts.
For example, two variables' names can be the same, however, they have to use different colors to represent them. And the bar chart segment must be consecutive, which means that it is hard to draw the "idle" part of our scheduler. The existing API is easy to use when its functions match most of your needs. And users only need to fill the small part of the code to generate a chart.

So, I tried to use a more simplified API instead and my next choice is P5 API to achieve basic functions. p5.js is a JavaScript library for creative coding. Using the metaphor of a sketch, p5.js has a full set of drawing functionality.



After figuring out how to scale and draw coordinates, paint rectangles with corresponding task colors, and set up local server to run the code to show canvas. A new problem arises which is it is hard to integrate p5

to an existing logic program. And also faced a problem that local server is not responding to the changes made in the code even after refreshing the page or restart the server.

- Use local server to host: https://github.com/processing/p5.js/wiki/Local-server
- After setting up local server, go to p5 folder and run command *http-server* and open http://localhost:8080/scheduler/

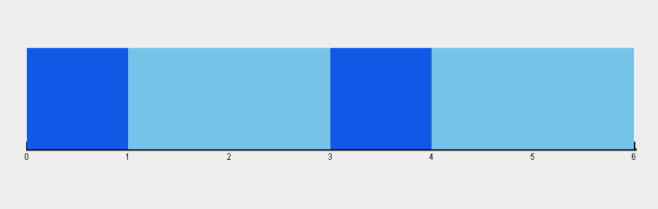
Furthermore, to reduce web page front end and integration with p5, I decided to use console based approach to get user's input and draw canvas to show scheduler as GUI.

After all, my final decision is to use the most reliable and basic tools – Java and Jpanel in Eclipse IDE. This way both logic and UI representation can both present smoother and more coherent.

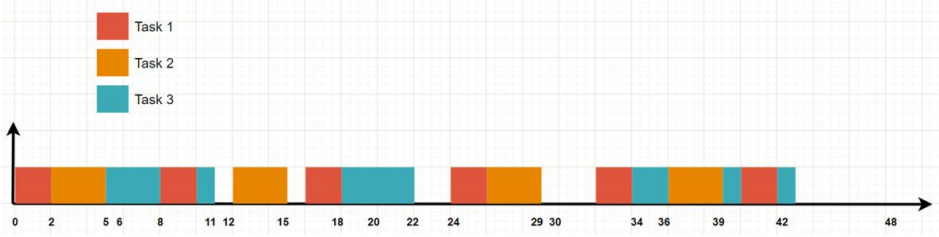
5. Things to Improve

Due to time constraints, some functions can be added:

1. In EDF algorithm in our program, if two tasks have the same deadline, then the task which have smaller index in the task list (determined by the user task input order) will be executed first, so in this case, the numbers of context switch are not the smallest.
   For example, the last two block period of time from time 3 to 4 and 4 to 6 can be swapped to reduce the context switch.



2. In the program graph, only different color blocks are showed and represent a different task. It could be better if create an indicator that showing which color represents which task. Like the following graph



6. Conclusion

- Successfully use Eclipse as IDE and Java, Jpanel to achieve scheduling analysis, construct scheduler graph, and provide text feedback.
- Achieved standard RMS and EDF in format (task computation time, task period). Note: deadline is equals to task period and task is always ready at the beginning of each period.
- User Input: workload parameters: number of tasks, "RMS" or "EDF" option, and task set details in form of (c, p).
- Output: RMS; EDF; Include utility test and exact analysis. Use analysis to test whether the given workload is schedulable by "RMS" or "EDF". If it can pass analysis test, then draw on canvas that can show scheduler details. If not pass test, then show the arithmetic reason on console.