Project Documentation

Zihan Jin

1. Explain your choice for the data structure you used for D in this project.

   I chose an arraylist named processItems to temporarily store the data for the following reasons:

   a. **A list, not like an array, allows the undeclared length of variables in the list. Since the exact number of the variables that will be written into the data structure is unknown, an arraylist is a good choice with more flexibility.**

   b. **An arraylist is easy to add, update, insert, delete and sort the variables in it. When passing variables to the priority queue (PQ), it is easier to sort the variables by arrival time first and then delete the first variable after passing it to the PQ to keep the first element always be the one that has the earliest arrival time and is about to be passed.**

2. For processes that had equal priority, it may have been better to execute the process with earlier arrival time instead of choosing arbitrarily. At a high level, how would you have to modify your project to accommodate this?

   I will add an if condition to make sure that the one with earlier arrival time will run first. The pseudocode is written as below:

   for all the processes j in the PQ:

   create an arraylist to record the processes in PQ

   sort the arraylist by j.getArrivalTime

   If (several processes have the same priority):

   currentProcess = process with the smallest j.getArrivalTime

3. What other changes could you consider making to your project to improve efficiency or readability or reusability?

   a. **When creating the Process class, besides the four basic factors (id, priority, duration and arrivalTime), I also added two more factors.**

```
public class Process implements Comparable<Process>{
    2 usages
    private int id;
    2 usages
    private int priority;
    2 usages
    private int duration;
    3 usages
    private int arrivalTime;
    2 usages
    private int runTimeLeft;
    2 usages
    private int waitTime;
```

It might be redundant, but I could not figure out any other methods to solve the runTimeLeft and waitTime problem.

b. For the loop through PQ and update the priority

```
// loop through the PQ to lower the priorities for all the processes that reach the
for (Entry<Integer, Process> pro: priorityQueue){

    // create an arraylist to record all processes in PQ
    ArrayList<Process> updatedWaitTimes = new ArrayList<>();
    updatedWaitTimes.add(pro.getValue());

    for (Process i: updatedWaitTimes){
        if (currentIdCheck != i.getId()){
            i.setWaitTime(i.getWaitTime() + 1);
            waitTime += 1;
        }

        // find the objects that reach the maxWaitTime
        if (i.getWaitTime() == maxWaitTime){

            // update priorities of processes that have been waiting longer than max
            priorityQueue.replaceKey(pro, key: i.getPriority() - 1);

            System.out.println("Process " + i.getId() +
                    " reached maximum wait time... " +
                    "decreasing priority to " + (i.getPriority() - 1));
```

```
// update the priority
i.setPriority(i.getPriority() - 1);
// reset the waitTime
i.setWaitTime(0);
```

I used the waitTime factor and set it to 0 every 30 time units. There might be an improvement by not using this factor.