University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Operating Systems (H)**			
Coursework Number	**1/1**			
Deadline	**Time:**	**05:00 pm**	**Date:**	**15th FEBRUARY 2017**
% Contribution to final course mark	**20%**			
Solo or Group ✓	**Solo**	✓	**Group**	
Anticipated Hours	**20**			
Submission Instructions	**See details in assignment and on the Moodle page.**			
Please Note: This Coursework cannot be Re-Assessed				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via https://studentltc.dcs.gla.ac.uk/ for all coursework UNLESS submitted via Moodle

A. Rationale

The rationale of this assignment is to evaluate the performance of a set of scheduling algorithms based on the methodology of: **algorithmic simulation**. This is the most popular performance assessment method, in which *each* algorithm is given specific pre-determined process workloads with certain statistical properties. Then, *all* the considered algorithms are evaluated *onto the same process workloads*, thus, providing the analyst / programmer / developer a useful insight on the behaviour of those algorithms. At the end of the evaluation, the analyst provides a written report on selected performance metrics for each algorithm along with reasoning on the algorithms' behaviour with respect to different statistical properties of the process workloads.

B. Overall Description

You are about to analyse the performance of the three fundamental scheduling algorithms: **First Come First Served** (FCFS), **non-preemptive Short Job First** (SJF), and **Round Robin** (RR).

Overall, the tasks of this assignment are: (i) to implement those algorithms, (ii) to generate certain process workloads, (iii) to calculate the (statistical) results with respect to **Average Waiting Time** and **Average Turnaround Time**, (iv) to illustrate your evaluation results through figures, and (v) to provide a **brief critical evaluation** for each algorithm based on the evaluation results.

C. Detailed Tasks Description

Note 1: The following tasks and scheduling algorithms should be implemented in **Java or ANSI/ISO/Standard C.** You are advised to avoid any nonstandard library for implementation/programming.

Task 0: Process Workload (W1)

You have to 'generate' a **process workload W1** containing N = 1000 processes (N is fixed). A process is represented by a **unique ID** (PID), e.g., an integer from 0 to 999, and a fixed **CPU Burst Time** (in time units, e.g., milliseconds).

- The **CPU Burst Time (CBT)** for each process is randomly generated by a **Gaussian distribution G1** with *mean value* M1 = 20 (time units) and *standard deviation* STD1 = 3 (time units). To avoid possible negative values, we define a minimum value for CBT, which is CBT_MIN = 5 (time units). Hence, the lower-bounded CBT value (CBT_bounded) is:

  CBT_bounded = max(CBT_MIN, CBT)

  Also, for convenience of implementation, use the **ceiling function (CEILING)** to get the smallest integer greater than or equal to CBT_bounded. *Hereinafter*, when we refer to CBT value, we refer to the ceiling of the lower-bounded CBT, i.e., CEILING(CBT_bounded).
- Thus, each generated process has:
  - **A unique PID**, which is an integer in [0,999],
  - **CPU Burst Time (CBT)**, which is a real number from Gaussian distribution **G1(M1, STD1),**

    - **Absolute Arrival Time (AAT)** to the Scheduler, which is a positive integer time index.
- All processes are sequentially generated, i.e., first generate process 0, then process 1, process 2, and so on.
- The **Absolute Arrival Time (AAT)** to the Scheduler of a process *i* is generated by the AAT of the previously generated process (*i*-1) **plus** an inter-arrival interval ΔT. This interval ΔT is randomly generated by a **Poisson distribution** with *mean* L = 5.
- In the case of the first process, i.e., process 0, the AAT(0) is set to 0, i.e., time zero. Then, the AAT for the second process, i.e., process 1, is: AAT(1) = AAT(0) + ΔT, where ΔT is generated by Poisson(L). The AAT for the third process, i.e., process 2, is AAT(2) = AAT(1) + ΔT, where ΔT is a new generated by Poisson(L), and so on.

Example 1: Here, we provide an example for generating the process workload. Specifically, consider a workload with only four processes.

- The first process has PID = 0. The CBT of the first process, CBT(0) is a **random number** from G1(20,3), say, 18 time units. The AAT(0) for the first process is AAT(0)=0, which is the starting time instance.
- The second process has PID = 1. The CBT of the second process, CBT(1) is a **new random number** from G1(20,3), say, 23 time units. The AAT(1) for the second process is AAT(1) = AAT(0) + ΔT. The value of ΔT is given by the Poisson(5), say, ΔT = 7. Hence, AAT(1) = 0+7 = 7, which is the absolute arrival time instance for process PID = 1.
- The third process has PID = 2. The CBT of the third process, CBT(2) is a **new random number** from G1(20,3), say, 21 time units. The AAT(2) for the third process is AAT(2) = AAT(1) + ΔT. The **new value** of ΔT is given by the Poisson(5), say, ΔT = 3. Hence, AAT(2) = 7 + 3 = 10, which is the absolute arrival time instance for process PID = 2.
- The fourth process has PID = 3. The CBT of the third process, CBT(3) is a **new random number** from G1(20,3), say, 20.5 time units. The AAT(3) for the fourth process is AAT(3) = AAT(2) + ΔT. The **new** value of ΔT is given by the Poisson(5), say, ΔT = 11. Hence, AAT(3) = 10 + 11 = 21, which is the absolute arrival time instance for process PID = 3.

Hence, the example workload has four processes PID = 0, 1, 2, 3 arriving at **0, 7, 10, and 21** time instances, respectively, and requiring: **18, 23, 21, and 20.5** CBT, respectively.

Pseudocode 1: Here, we provide the pseudocode for generating the workload of N processes.

```
Input N: number of processes
Start
PID[0] = 0;
CBT[0] = generate a random number from Gaussian(20,3);
CBT[0] = ceiling(max(CBT_MIN,CNT[0])); //ceiling of the bounded CBT w.r.t. CBT_MIN
AAT[0] = 0;
For k=1 to N-1
        PID[k] = k;
        CBT[k] = generate a random number from Gaussian(20,3); //a new value is generated here
        CBT[k] = ceiling(max(CBT_MIN,CNT[k])); //ceiling of the bounded CBT w.r.t. CBT_MIN
        ΔT = generate a random number from Poisson(5); //a new value if generated here
        AAT[k] = AAT[k-1] + ΔT;
End For
End.
Return arrays: PID, CBT, AAT of N entries.
```

Each triplet: (PID, CBT, AAT) corresponds to a process.

Task 1: Process Workload (W2)

Just *repeat* Task 0 to 'generate' a **process workload W2** containing N = 1000 processes with:

- The **CPU Burst Time (CBT)** for each process is randomly generated by a **Gaussian distribution G2** with *mean value* M2 = 60 (time units) and *standard deviation* STD2 = 3 (time units). Use the CEILING(CBT_bounded), again.
- The **Absolute Arrival Time (AAT)** to the Scheduler of a process *i* is generated by the AAT of the previously generated process (*i*-1) **plus** an inter-arrival interval ΔT. This interval ΔT is randomly generated by a **Poisson distribution** with *mean* L = 5 (**as in W1**).

In process workload W2, the CBT_MIN = 5 (time units), i.e., the same as in process workload W1.

Note 2: The outputs of Task 0 and Task 1 refer to two different **.csv files** (file W1.csv and file W2.csv), each one containing N = 1000 triplets of the form: '**PID, CBT, AAT'** , with no headers, no titles, and no meta-data. Please refer to Section D for more information. Also, please ensure the Task 0 and Task 1 print the corresponding process workloads into the standard output (e.g., display) for debugging purposes.

Task 2: Simulation of the Scheduling Algorithms

In this task, you will implement a simulator of *each* scheduling algorithm. Then each scheduling algorithm will be fed with the process workloads W1 and W2 from **Task 0 and Task 1.**

Note 3: The **input** of *each* scheduling algorithm is **a process workload file**, e.g., W1.csv file.

Run first each algorithm with the workload W1 (Task 0) and record the statistical metrics (discussed later). Then, run again each algorithm with the workload W2 (Task 1) and record the corresponding metrics.

- **Sub-Task 2.1:** Simulation of the **FCFS algorithm**. The input of this algorithm is the workload W1 from Task 0 / the workload W2 from Task 1. This algorithm treats the processes one by one as indicated by their arrival time instances from the array **AAT**.
- **Sub-Task 2.2:** Simulation of the **SJF non preemeptive** algorithm. The input of this algorithm is the workload W1 from Task 0 / the workload W2 from Task 1
- **Sub-Task 2.3:** Simulation of the **RR algorithm** with quantum: **Q = 15 time units**. The input of this algorithm is the workload W1 from Task 0 / the workload W2 from Task 1.
- **Sub-Task 2.4:** Simulation of the **RR algorithm** with quantum **Q = 5 time units**. The input of this algorithm is the workload W1 from Task 0 / the workload W2 from Task 1.
- **Sub-Task 2.5:** Simulation of the **RR algorithm** with quantum **Q = 40 time units**. The input of this algorithm is the workload W1 from Task 0 / the workload W2 from Task 1.

Statistical Metrics:

For *each* of the sub-tasks T2.1, T2.2, T2.3, T2.4, and T2.5, proceed with the following statistical calculations:

- Calculate the **Waiting Time (WT)** for *each* process of the workload W1 (and W2 separately).
- Calculate the **Turnaround Time (TA)** for *each* process of the workload W1 (and W2 separately).
- Calculate the **Average Waiting Time (AWT)** for *all* N *processes* based on the formula:
  - **AWT = (WT[0] + WT[1] + … + WT[N])/N**

  for the workload W1 and W2 separately.
- Calculate the **Average Turnaround Time (ATT)** for *all* N *processes* based on the formula:
  - **ATT = (TA[0] + TA[1] + … + TA[N])/N**

  for the workload W1 and W2 separately.

Note 4: The **output** of *each* scheduling algorithm is printed into the standard output with the following format (**information per line**):

- **The PID (process ID)**
- **The (WT) waiting time for the process PID**
- **The (TA) turnaround time for the process PID**

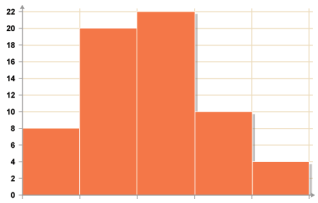
At the end, print out the: **average waiting time (AWT)** and the **average turnaround time (ATT)** in the standard output.

Task 3: Illustration of the Statistical Results

In this task, you will illustrate in **Figure 1.1** and in **Figure 1.2** the values of the statistics: **AWT** and **ATT** for all scheduling algorithms, i.e., FCFS, SJF (non pre-emptive), RR (Q=16), RR (Q=2), and RR (Q=30), respectively, for the process workload **W1**.

Similarly, you will illustrate in **Figure 2.1** and in **Figure 2.2** the values of the statistics: **AWT** and **ATT** for all scheduling algorithms, i.e., FCFS, SJF (non pre-emptive), RR (Q=16), RR (Q=2), and RR (Q=30), respectively, for the process workload **W2**.

Note 5: You could use a *bar-chart illustration* for showing the values of AWT and ATT for each algorithm per process workload, where the *x*-axis refers to each algorithm and the *y*-axis to the statistic metric, e.g., as in the example below:



Example of a bar-chart

Task 4: Critical Evaluation (max 300 words)

Critically evaluate the results as illustrated by the four figures for each algorithm and for each workload. Specifically, *reason* and *explain* the behaviour / performance of *all* considered algorithms / variants with respect to the process workloads W1 and W2. Focus on the following reasoning questions (as an example, you could introduce *more* reasoning questions) and provide a **PDF file** with up to **300 words (±10%)** concluding on e.g.:

- Why algorithm X behaves better in W1 than in W2 for the same statistical metric, *e.g.*, AWT?
- Why the behaviour of algorithm is (or, is not) independent of the W1 and W2 under a specific statistical metric, *e.g.*, ATT?
- Why algorithm X behaves similarly (or differently) with algorithm Y in W1 (or in W2)?
- Why algorithm X with different parameter values behave similarly in both workloads given a specific statistical metric?
- …

D. Deliverable & Submission

Submissions should be made through **Moodle**. Each student must complete the online **Declaration of Originality Form** from:

https://webapps.dcs.gla.ac.uk/ETHICS/index.cfm

By the stated deadline, you must submit the following items:

1. Upload **two .csv files** (one for each workload) with names: **W1.csv** and **W2.csv**, where each row contains the triplet that represents the process, i.e., **PID, CBT, AAT**. Obviously, each file contains N = 1000 triplets. Do not insert headers, titles, and any other meta-data.
2. Fill in the following workloads statistics Table. Specifically, calculate the *min* and *max* values of all CBT times and the average (mean) AAT values that you generated in each workload.

Workloads	**Max** value of all CBT times	**Min** value of all CBT times	**Average** (*mean*) of all AAT values
W1			
W2			

3. Upload the *commented* source code (one or more files, depending of your implementation strategy) for each implemented scheduling algorithm. In each source code, provide in-line comments on the parameters, data structures and **anything** you consider meaningful for explaining your implementation.
4. Provide a **PDF document** of your critical analysis and evaluation. This document contains:
    a. The *four* (*4*) *Figures* of Task 3. **Note**: **do put** labels on the *x*-axis and on *y*-axis **for each figure**, and **associate** each figure with the corresponding workload. **Failing to do so, it will not be considered for marking.**
    b. The *critical evaluation text* of Task 4, with maximum number of words: **300 words (±10%)**.
    c. The filled in workloads statistics Table (see item 2 in this list).

Here is a *checklist* of the submitted items:

Submitted Item	Yes / No	Marks (out of 100)
W1.csv file		**5.0**
W2.csv file		**5.0**
Commented Source code for **FCFS**		**10.0**
Commented Source code for non-preemeptive **SJF**		**15.0**
Commented Source code for **RR**		**15.0**
Document of critical analysis and evaluation: **four (4) Figures**		**20.0**
Document of critical analysis and evaluation: **Critical Evaluation**		**30.0**
		Total: 100

E. Marking Scheme

	Code	**Critical Analysis**	**Graphs**
A	*Algorithms are implemented in a highly efficient manner. Code is carefully documented. Code compiles and results are correct.*	*The report is concise and very clear with sound judgment of the comparative merits of the scheduling algorithms. The report identifies high-level conclusions.*	*The graphs are well-presented with appropriate labels and legends.*
B	*Algorithms are moderately efficient. Code has some documentation. Code should compile and results are mostly correct.*	*The report is fairly clear. There is some indication of the comparative merits of the different algorithms.*	*The graphs are fairly well-presented. Some labels and legends are given.*
C	*Algorithms are generally correct but may be inefficient. Documentation is present but not clear. Code may not compile.*	*The report is somewhat clear, but may be too long. There is some attempt to compare the algorithms' merits.*	*The graphs are unclear. Labels etc may be missing.*
D	*Algorithms are sketched out but not implemented fully. Documentation is minimal. Code may not compile.*	*The report is unclear. It does not generalise from the results of the experiments.*	*The graphs are inaccurate and incomplete.*
E	*Algorithms are not implemented properly. Documentation is insufficient. Code does not compile.*	*The report is of an unacceptable length. There is no empirical analysis or high-level conclusions.*	*No graphs are provided.*
F *or lower*	*The code is incorrect and does not compile. No documentation is provided.*	*The report is incomplete.*	*No graphs are provided.*

Appendix

For your convenience, we provide a Java source code (class Generator) for generating 1000 **Gaussian random numbers** from a Gaussian distribution with mean value M1 and standard deviation STD1, and for generating 1000 **Poisson integer random numbers** from a Poisson distribution with mean value L. **Note**: the generated values *in the appendix* are neither truncated nor bounded by any constant.

```
import java.util.Random;
public class Generator {
        public static void main(String[] args) {
                double M1 = 20;
                double STD1 = 3;
                double L = 5;
                Random random = new Random(System.currentTimeMillis());
                //generate 1000 Gaussians and Poisson values
                for (int i=1; i<1000; i++){
                        double gaussianValue = random.nextGaussian() * STD1 + M1;
                        int poissonValue = Generator.myPoisson(L, random);
                System.out.println("Gauss: "+gaussianValue+" and Poisson: "+poissonValue);
                }//loop
        }//main

public static int myPoisson(double mean, Random random) {
      int k = 0;
      double p = 1.0;
      double expLambda = Math.exp(-mean);
      do {
         k++;
         p *= random.nextDouble();
      } while (p >= expLambda);
      return k-1;
  }// myPoisson
}//Generator
```