National University
Of Computer and Emerging Sciences

Term Project

Due Date: 1st June 2022 11:59 PM

Important Instructions:

1. The students who are involved in plagiarism will get zero marks.
2. Start early so you can finish on time. No extra time will be given.
3. All the submissions must be on google classroom.
4. You have to submit one zipped file containing the solution code files. (i20-XXXX_TermProject.zip)
5. Be prepared for demos after the submission of the project.

Summary:

In this project you will familiarize yourself with detection of deadlocks in a multi-process environment. Please note that you do not need to lock any files to implement this project.

You will be given text files containing the description of running processes and the files they need to execute, where each process is associated with two files. The first is the file that the process has locked and the second is the file that it is waiting to lock. The goal is to determine whether given the current resource allocation and requests there is a deadlock or not.

If there is no deadlock, you are asked to compute the minimum execution time required for all processes to finish. If a process locked a file, it takes 1 time unit to operate on this file and release it. If a process requested a file, after a process is given this file, it takes 1 time unit to operate on this file and release it. Hence, any process takes at least 2 units time to finish. If k processes have requested the same file, it will take k time units for the processes to finish, where $k \geq 1$.

Processes that have locked or requested distinct files can operate in parallel. That is, if two processes have requested two different files then it will take a total of 1 time unit for the processes to operate on these files if both files are not locked by other processes. If there is a deadlock, you are asked to return process(es) that have to be terminated in order to resolve the deadlock.

Description:

Following is the detailed description of project and tasks need to be completed.

Input File Format:

Each line of the file corresponds to a process and consists of a space-separated tuple: where the first **file-id** is the file that the process has locked and the **second file-id** is the file requested by the process.

For example, **resources.txt** with the following information: describes a system where process 0 has locked file 1 and is waiting for file 3 while process 1 has locked file 2 and waiting for file 7.

0 1 3

1 2 7

If **resources.txt** has the following information: then the system has processes 0 and 1, with locks on files 1 and 2, respectively. In addition, process 0 requested file 2 while process 1 requested file 1. Hence, there is a deadlock.

0 1 2

1 2 1

Tasks:

1. The initial task to print the statistics i.e., total number of processes and files in the system (both locked and requested).
2. In this subtask, you are required to compute minimum execution time, assuming that there is no deadlock in the file allocation for this task only. You are asked to print minimum execution time required for all processes to finish given the specification in summary.
3. This task requires you to detect deadlock i.e., program should print “No deadlocks” if there are no deadlocks, or “Deadlock detected” if there is at least one deadlock.
4. Once deadlock detected, your program should return a list of processes that need to be terminated to break the deadlock. You should print them on a separate line, separated by spaces. If there are many processes which can be terminated to break a particular deadlock, choose the process with the smallest ID.

5. (Challenge Task) If a deadlock was detected, you are asked to come up with a transcript that allocates files to processes in deadlock-free manner. For that, assume that all resources are available and no files have been locked yet and both file-ids associated with a process in the input file are two files required for the process to execute. A process can only execute if it obtains both files at the same time. Once it obtains the files, it locks them and releases at the next time unit. Hence, a file cannot be allocated to more than one process at any given time. A hacky way of allocating files to processes would be to allocate both files to a process at each time unit. This would however lead to a slow execution that does not have any parallelism. Your task is instead to allocate files to separate processes at the same time while avoiding deadlocks.

For task 5, you are asked to print an allocation transcript in the following format. If a process is allocated its two files, print on a separate line: (see sample output 3)
time process-id file-id, file-id

Sample Outputs:

Sample output 1
Given resources.txt with the following information:
0 1 2
1 2 1

On **./detect -f resources.txt** your program should print:
Processes 2\n
Files 2\n
Deadlock detected\n
Terminate 0\n

Sample output 2
Given resources.txt with the following information:
0 1 3
1 2 4

On **./detect -e -f resources.txt** your program should print:
Processes 2\n
Files 4\n
Execution time 2\n

Sample output 3
Given resources.txt with the following information:
0 1 2
1 2 3
2 3 4
3 4 1

./detect -f resources.txt -c could print:
0 0 1,2\n
0 2 3,4\n
1 1 2,3\n
1 3 4,1\n
Simulation time 2\n

Program Requirements:

Your program will be invoked via the command line. It must be called ./detect and take the following command line arguments. The arguments can be passed in any order but you can assume that they will be passed exactly once.

-f filename specifies the path to the file describing requests.

-e is an optional parameter, which when provided requires your code to compute execution time. If this option is not given, your code should identify presence or absence of deadlocks and which processes need to be terminated to resolve deadlock(s).

-c is an optional parameter, which when provided invokes your file per process allocation from challenge task.