Project: Process Synchronization using C/Linux and Java [80 points = 40 + 40]

Implement the following project using:

a) C/Linux, and
b) Java

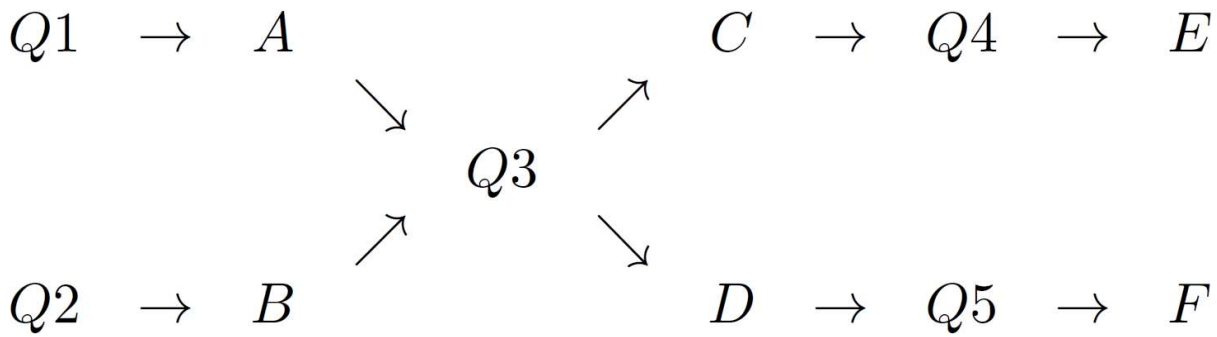
A sausage factory has 6 machines: machines A and B are fat trimming machines, machines C and D are sausage cutting machines, and machines E and F are packaging machines. Packages of meat come in either packages of 20 lb or packages of 25 lb. The inputs and outputs of each machine are stored in queues.

A takes as input 20 lb of sausage mixture from Q1 while B inputs 25 lb of sausage mixture from Q2. Both A and B "output" 15 lb packages of leaner mixture to Q3, however, A produces a "lean" mixture while B produces an "extra lean" mixture. Thus the contents of Q3 are heterogenous; it contains 15lb blocks of "lean" and "extra lean" mixture: they do not mix.

Both C and D input 15 lb of sausage mixture from Q3. However, C cuts large sausages and produces 100 large sausages while D cuts small sausages and produces 400 small sausages. Since Q3 contained both "lean" and "extra lean" blocks, this means that depending on the kind of block processed, C may produce large & "lean" or large and "extra lean" sausages. Similarly, D may produce small & lean or small & "extra lean" sausages. C outputs its sausages to Q4. D outputs its sausages to Q5.

Finally, E takes as "input" 100 large sausages and produces 20 packages of 5 sausages each. F takes as input 400 small sausages and produces 20 packages of 20 small sausages at a time.

Each machine does its job in a random amount of time that can fall anywhere between 3 and 7 minutes (i.e., 180 and 420 seconds). This is simulated by generating a number n between 180 and 420 at random and running a "for loop" that increments a variable n times. Every time E or F completes a package, it updates a database that records the number and types of packages produced.



Here are the restrictions for the machines in the factory:

A and B can work concurrently. They can only process one package of meat at a time.

C and D can work concurrently; They can only start working once outputs from A or B have been produced. They can take outputs from either A or B. **Also, and very importantly, D can produce a package only after C has produced 4 and only one out of three outputs from B can go to D. In other words, we don't want more than 1/5th packages of small sausages and only 1/4th of these packages can be extra lean.** As soon as an output from A or B arrives, if C and D are both ready to produce (based on the restrictions highlighted in bold above), they must race to process it. The one who wins the race takes it. C and D can only race if they have finished their previous job.

E and F can work concurrently. E can only take as input the output of C while F can only take as input the output of D. They can only do so when they are done with their previous task. Once they've done their jobs, E and F will each modify a table of 4 integers that reports the number of packages of 1) small lean sausages; 2) large lean sausages; 3) small extra lean sausages; 4) large extra lean sausages. This table should be a global array shared between E and F.

Use semaphores (Mutex and Counting Semaphores) to synchronize C and D's processing of the 15 lb lean or extra lean meat packages according to the rules stated above in bold; as well as E and F's access to the shared table.

Please test your system in the following three scenarios:

- 30 packages of 20 lb and 20 packages of 25 lb
- 100 packages of 20 lb and 200 packages of 25 lb
- 3,000 packages of 20 lb and 600 packages of 25 lb

The results should be:

	Large	Small
Lean	22	8
Extra Lean	18	2

	Large	Small
Lean	52	48
Extra Lean	188	12

	Large	Small
Lean	2460	540
Extra Lean	420	180

Interface

C Interface

For this to work requires three different files. Your code, which we'll call `ASSN3_example.c`, a set of functions I wrote called `ASSN3_grader.o`, and a header file `ASSN3_declarations.h` which allows your code in `ASSN3_example.c` to access some of the functions in `ASSN3_grader.o`. In order for C to see all the functions, put all three of these files in the `~/OS-VM` directory. You can then compile and run the program in your VMs `/vagrant` directory with:
`gcc -pthread ASSN3_example.c ASSN3_grader.o -o ASSN3_example; ./ ASSN3_example`
Be sure that your program contains the line `#include "ASSN3_declarations.h"`, so that its definitions can be linked into your program.

Declare an array of two integers, called `input` in your main program. Call the `get_input` function with a pointer to `input` in order to fill the `input` array. The values in this array represent the amounts of 20lb packages in Q1 and 25lb packages in Q2. Run your program to process the input. Then save your output to a 2x2 array (say `output`), whose i,jth entry has the same interpretation as in the examples above.
Call `return done(&output)`, which will check the correctness of the program, printing correct and returning 0, or not correct and returning 1.

Algorithm 1 ASSN3_example.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "ASSN3_delarations.h"

int main()
{
    //srand(time(0));       //set the rng seed
    int input [2];          // declare input array
    get_input(&input);     // initialize the input array
    /***** YOUR PROGRAM HERE *****/
    // the correct output for the default RNG seed
    int output [2][2]= {{52, 48},{188, 12}};
    return done(&output);
}
```

Java Interface

The Java interface works similarly. We define a class called `Grader.class`, which the student's program (say `Example_Program.java`) must call. With both files in the same directory, your program should begin by creating a `Grader` object. `Grader`'s instance variable `input` contains a randomly generated test input `[a,b]`, where a is the amount of 20lb blocks of sausage in Q1, and b is the amount of 25lb blocks of sausage in Q2. Run your program to determine the correct output. Save your output to a 2D array (say `ouput`), via:

`int [][] output = { { 22, 8 }, { 18, 2 } };` An example program is shown below. **Be sure your program ends with the if-else conditions shown in 12-17**.

Algorithm 1 Example_Program.java

```
1  import java.util.*;
2  import java.lang.*;
3  class Example_Program
4  {
5      public static void main(String args[]){
6          Grader obj = new Grader();
7          int[] input = obj.input;                          // get random input array
8          System.out.println("The input is: " + Arrays.toString(input));
9          /***** YOUR PROGRAM HERE *****/
10         int [][] output = { { 22, 8 }, { 18, 2 } };      // Correct ouput for [30, 20] input
11         System.out.println("The output is: " + Arrays.deepToString(output));
12         if (Grader.done(output)==0) {  // 0 is a normal exit
13             System.exit(0);
14         }
15         else {
16             System.exit(1);
17         }
18     }
19 }
```

Lastly, please don't compress your files: just upload each file to blackboard individually. Please try to give your files unique and identifiable names, e.g:

`lastname_firstname_ASSN3.c`
`lastname_firstname_myclass1.java`
`lastname_firstname_myclass2.java`