1. (5 pts.) Assume the code below is a solution to the producer-consumer problem.

```c
#define N = 100
int count = 0;
void producer () {
    int item;
    while (true) {
        item = produce_items();
        while(count==N);
        insert_item(item);
        count++;
    }
}

void consumer () {
    int item;
    while (true) {
        while(count==0);
        item = remove_items();
        count--;
        consume_item(item);
    }
}
```

This solution is:
B) wrong, because race condition produces inconsistent results.

2. (5 pts.) If 10 processes run the code below concurrently, and the initial value for the shared variable s is equal to 0. What cannot be the final value for s after all the 10 processes finish? (i is a local variable.)

```c
for (i=0;i<5;i++)
    s+=1;
```

A) 2  B) 1  C) 6  D) neither 1 and 2

3. (5 pts.) Using the Round Robin algorithm and assuming that the time quantum is equal to 1, and context switch time is equal to 0.5, what is mean response time of the processes?
4. (5 pts.) Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when the time quantum is 1 millisecond?

The time quantum is 1 millisecond. The scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of \( \frac{1}{1.1} \times 100 = 91\% \).

5. (20 pts.) To solve a problem, you need to perform a number of tasks, \( n_t \). Some of them can be executed in parallel; if all the information a task needs (its input) are available, the task proceeds with its computation, producing a result. Results from tasks are input to other tasks; this fact defines a Data Dependency Graph (DDG). We allow the graph to contain cycles. The whole system has a number of exogenous inputs, \( n_i \), which have no data dependencies. The following figure depicts an example of such system.
Solution:

DDG – Matrix representation of the Data Dependency Graph
Input – Matrix representation of the Input Dependency
Output – indicates which task is the output
Finish – Flag to signal when computation is done

Every link in the graph corresponds to a consumer-producer relation.

Where a semaphore stops the producer until data has been consumed, and stops consumer until data has been produced.
From the DDG and Input matrices, we build the corresponding semaphores.

Inputs(i) – returns identifiers of input variables needed by task i.
Producers(i) – returns identifiers of tasks required by task i.
Consumers(i) – returns identifiers of tasks that require data produced by task i.
Done() – predicate that indicates when computation is finished.

Computing tasks wait for input data and then for output from tasks, computes, and then waits for output place to be available (read by every consumer).

Input tasks just wait for output place to be available (read by every consumer).
The launcher establishes semaphores and then creates all threads.