Assignment 3: Semaphores

ETH Zurich

1 Precedence to Implementation

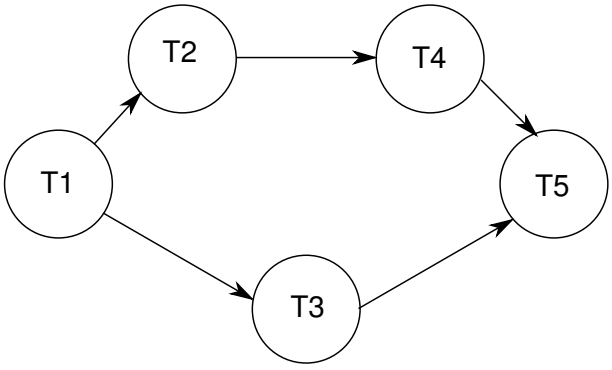
1.1 Background

This task is taken from *Foundations of Multithreaded, Parallel, and Distributed Programming* [1].

1.2 Task

A precedence graph is a DAG. Nodes represent tasks and edges indicate the order in which tasks are to be completed. In particular, a task can execute as soon as all of its predecessors have been completed.

1. Given the following precedence graph,



   give an implementation in Java that satisfies the order restrictions, using a minimum of semaphores.

2. Devise a general scheme that given any DAG, can assign semaphores to edges or processes. Do not worry about minimizing the number of semaphores, as this is an NP-hard problem (for an arbitrary DAG).

1.3 Solution

1. Given in code, the minimal number of semaphores is 3.

2. The general scheme will give, for every edge, a semaphore. The source of the edge will signal its semaphore, while the target will wait on it.

2 Interleaving with Semaphores

2.1 Background

This task is also taken from *Foundations of Multithreaded, Parallel, and Distributed Programming* [1].

2.2 Task

Given the following processes and code, give the possible outputs of the interleavings:

s.count := 0 r.count := 1 x := 0		
P_1	P_2	P_3
s.down r.down x := x * 2 r.up	r.down x := x * (x + 1) r.up s.up	r.down x := x + 2 r.up

2.3 Solution

The possible execution orders are
P_2, P_1, P_3 or P_2, P_3, P_1 or P_3, P_2, P_1.
The corresponding outputs are: 2, 4, 12.

3 Unisex bathroom

3.1 Background

This task has been adapted from *Foundations of Multithreaded, Parallel, and Distributed Programming* [1]. In an office there is a unisex bathroom with n toilets. The bathroom is open to both men and women, but it cannot be used by men and women at the same time.

3.2 Task

1. Develop a Java program that simulates the above scenario using semaphores from the Java concurrency library. Your solution should be deadlock free, but it does not have to be starvation free.

2. Justify why your solution is deadlock free.

3.3 Solution

The program and the justifications can be found in the source that comes with this solution.

References

[1] Gregory R. Andrews. Foundations of Multithreaded, Parallel, and Distributed Programming. Addison Wesley, 1999.