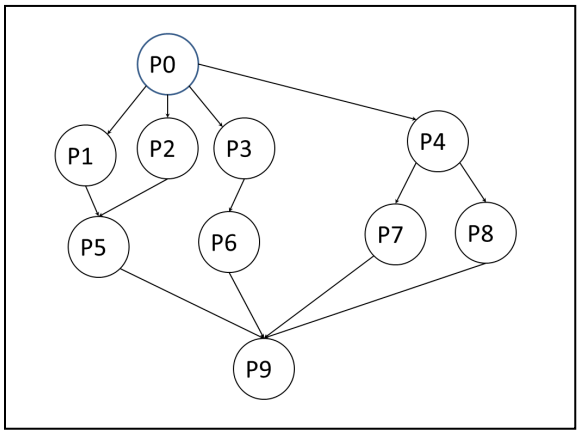
**50.005 Handout: On the use of Threads for solving Programming Assignment 1.**

This document briefly explains why it may be useful to employ threads to exploit concurrency in Programming Assignment 1. First of all, please note that the use of Threads is *purely optional* for Programming Assignment 1. In other words if you do not use Threads you will not be penalized. However, the use of Threads will help expose the concurrency in the problem. We use the terms *node* and *process* interchangeably. If there is a dependency from A to B (i.e. an arc from A to B), then let us call B a descendant of A. Please note that B is NOT a child process of A as in UNIX.

Let us say that we have a DAG as shown below:




**Solution without threads**

Your program's outer loop can only start P0 since all other nodes have input dependencies. Once P0 has completed, your program can start P1, P2, P3 and P4 before it waits for any one of these nodes to complete. If you are using ProcessBuilder and have objects for these processes in an array (let's say pb). You can only wait for one process completion at a time. Let us say your program decides to wait for pb[1], i.e. the process corresponding to node P1, to complete. The consequence will be that even if other nodes have finished (e.g. pb[4]), their descendants(pb[7] and pb[8]) in the DAG will not be scheduled for execution until P1 has finished. This problem can continue. In other words whenever the outer loop waits for process completion, other nodes ready to be executed in the DAG will also be forced to wait. This does not say that nodes are not scheduled in parallel, but rather the full extent of parallelism in the DAG is not exploited because of the manner that waiting for process termination is done.

**Solution with threads.**

The solution with threads is as follows. Your outer loop never waits for process completion for a single node. Instead it creates a thread for every node. It is the job of all the threads to invoke ProcessBuilder to start a process under the operating system and to wait for the completion of that process. Thus, the outer loop can proceed independently and continuously to check the status of every node. When any of the node is ready for execution, the outer loop starts the thread associated with that node. When all nodes have completed the program terminates. Notice, that if you choose to use threads you will also need to use lock to access shared data.