50.005 Computer System Engineering

Programming Assignment #1: Process Tree Management

Due date: Friday of recess week (10th March 2017)

You can work in a group of max 2 students

1. Purpose

- To develop a program (processmgt.c / processmgt.java) that uses UNIX system calls (fork, exec, wait etc.) in C or (ProcessBuilder) in java, to traverse a directed acyclic graph (DAG) of user programs in parallel.
- The DAG of user programs is traversed and executed by reading commands in a text file.
- The program models control dependencies and data dependencies.
- A control dependency is one where a program may start only when its parent(s) is / are finished.
- A data dependency is one where a program requires input from its parent(s) before it can run.
- *Example*: the UNIX **make** program creates a dependence graph for all the files involved in making an executable binary file. Independent files are compiled individually, whereas dependent files have to wait for all their control dependencies to be cleared before they can be compiled or linked.

2. Description

- You will need to analyze a graph of user programs and determine which programs are eligible to run, and then run those programs.
- According to Figure 1, the programs associated with nodes P0 and P2 may be run first because they are not children of any other node.
- After node P0 finishes running, node P1 may be run because its dependency upon P0 is cleared. Similarly, node P3 may be run as soon as P2 finishes running, however node P4 may be run only when both P1 and P3 have finished.
- Each node in the graph represents one program to be run. Each node as it appears in the input file contains:

    - The program name with its arguments if any
    - Pointers to child(ren) node(s) (the node indices of its children nodes)
    - The input file to be used as standard input for that program
    - The output file to be used as standard output for that program
- Only when all of its parent nodes (nodes that point to a given node) have completed running, can that node start.
- The program processmgt.c / processmgt.java will fork and exec each of the nodes as they become eligible to run
- *Input / Output redirection,* which is a feature of UNIX, must be used so that each node can get its input from a file called the "standard input file", and write its output to a file called the "standard output file" for that node
- Your program should handle multiple root nodes which may be executed concurrently.

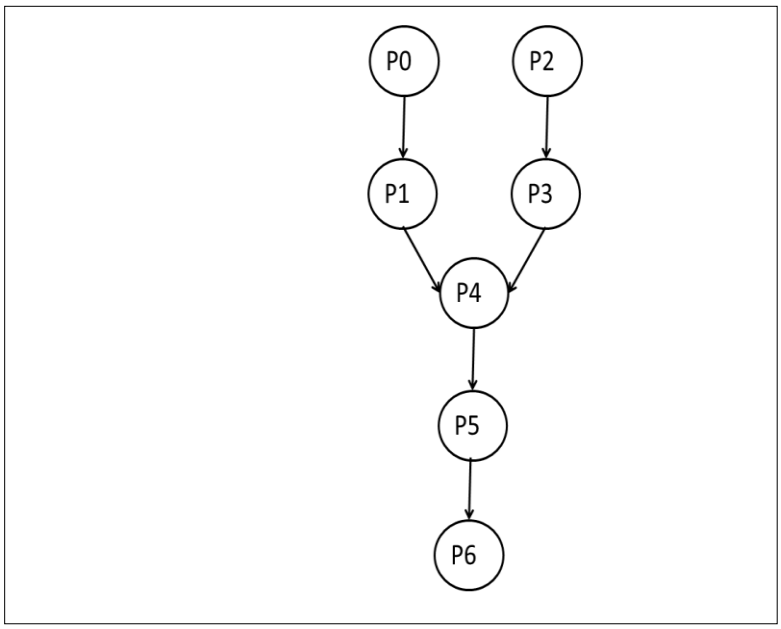

Figure 1. Example graph

3. Process Tree Specification

- Run processmgt.(c/java) program as follows in a terminal, as follows: $*./processmgt graph-file* OR $java *processmgt graph-file* . You may also do it from an IDE.

- A text file (eg graph-file), will represent the structure of the graph.
- The program processmgt reads the input file line by line, parsing the information contained in it (colon delimited). Each node of the graph is represented by a line with the following format:
  <program name with arguments:list of children ID's:input file:output file>

  For example, in the sample test file provided the lines are as follows:

```
sleep 10:1:stdin:stdout
echo "Process P1 running. Dependency to P4 is cleared.":4:stdin:out1.txt
sleep 15:3:stdin:stdout
echo "Process P3 running. Dependency to P4 is cleared.":4:stdin:out2.txt
cat out1.txt out2.txt:5:stdin:cat-out.txt
grep 3:6:cat-out.txt:grep-out.txt
wc -l:none:grep-out.txt:wc-out.txt
```

- NOTE: If there are no children for a node, the corresponding field must appear as "none". If the input and output files to a command are not redirected and they use the default STDIN and STDOUT streams, they are specified using "stdin" and "stdout" in the input and output file fields respectively.
- The nodes will implicitly be numbered from 0 to (n-1) from the order the nodes appear in the text file (where n = total number of nodes in the graph). The children IDs will correspond to this numbering system. The ID numbers are used mainly to create pointers from parent nodes to their corresponding children nodes.

4. Node Structure

Code for suggested node structure

```
// for 'status' variable:
#define INELIGIBLE 0
#define READY 1
#define RUNNING 2
#define FINISHED 3

typedef struct node {
    int id; // corresponds to line number in graph text file
    char prog[MAXLENGTH]; // prog + arguments
```

```
        char input[MAXLENGTH]; // filename
        char output[MAXLENGTH]; // filename5.
        int parents [MAX_PARENTS]; //parents' IDs
        int children [MAX_CHILDREN]; //childrens' IDs
        int status;       // ineligible / ready / running / finished
        pid_t pid;       // Process id when it's running
}       node_t;
```

5. Overview of processmgt.c / processmgt.java

The processmgt.c program will first parse the text file (eg. testproc.txt) in the first argument
The program will then construct a data structure that models the process tree management
The program will start executing the nodes
Important points in the program:

- Determine which nodes are eligible to run
- Run those nodes, modifying their states as needed
- Wait for any node to finish
- Repeat this process until all nodes have finished executing

A sample text file is given to you as input of your program and a sample output files are provided to you, to compare your results with the sample output files.

6. Useful System Calls and functions

- In C:
    - fork, dup2, execvp, wait, strcmp
- In Java:
    - BufferedReader, String, ProcessGraph, ProcessGraphNode, ProcessBuilder, Process, ProcessBuilder.redirectInput(), ProcessBuilder, redirectOutput(), Process.WaitFor()

7. Hints and Error Handling

Hints:

- Implement a function which marks the node that are ready to run
- A sample parsing function is provided. The function takes as input one line of the input file and a pointer to a node_structure
- It populates the prog, input, children and num_children fields of the node and returns a value of 1 on success.

Error Handling:

- To check the return value of all system calls in processmgt. c / processmgt.java for error conditions.
- To make sure the proper number of arguments are used when the program is executed
- A useful error message should be printed to the screen if there is an error
- Your program should recover from errors, if possible
  In C:
  use perror() function to print error messages and exit the program
  In java:
  try and catch exception error

8. Documentation

Include a README file which describes your program. It needs ot contain the following:

- The purpose of your program
- How to compile your program
- What exactly your program does

The README file does not have to be very long, as long as it is understandable by the first time user
Within your code you should use one or two sentences to describe each function that you write

At the top of your README file and main C or java source file please include the following comment:

```
/* Programming Assignment 1
 * Author : Full Name
 * ID: Student ID
 * Date: dd/mm/yyyy */
```

9. Grading

- 5% README file
- 20% Documentation within code, coding and style (indentations, readability of code, use of defined constants rather than numbers)
- 75% Test cases (correctness, error handling, meeting the specifications)
- Please pay attention to documentation and coding style. A perfectly working program will not receive full credit if it is undocumented and very difficult to read.
- The test cases will not be given to you upfront. They will be designed to test how well your program adheres to the specifications. So make sure that you read the specification very carefully. If there is anything that is not clear to you, you should ask for a clarification.
- This is a group programming assignment. Collaboration between at most two students is allowed. Detected cheating between groups will automatically result in a Zero for this assignment.

10. Additional features (optional)

NOTE: You are encouraged to do section 10, however it will not carry any weightage.

a) Demonstration of non-determinism:
   Since the underlying operating system returns from processes non-deterministically, demonstrate different outputs for different runs although there is no change in the input.

b) Dynamic process graph monitoring interface:
   Develop a dynamic graph monitoring interface that shows the changing states of the process graph nodes as the computation proceeds over time.

The interface should minimally be from the Command Line Interface, but an interface using appropriate GUI tools (eg Swing) would be appreciated (but no extra credit).

Use code E (or color green in case of GUI) to represent executing processes, code R (or color Yellow) to represent Ready processes and code W (or color Red) to represent waiting processes in the graph. A process that has completed should disappear from the monitoring interface.

Hint: to test your monitoring interface you may introduce larger graphs with delay nodes which can be shown to change state dynamically on the user interface.

Note: You may need to launch an external process and incorporate simple IPC through a shared file to implement the monitoring interface.