CEG 434/634: Concurrent Software Design

(Fall 2002)

PROGRAMMING ASSIGNMENT I

Forest Fire Simulation Using Multiple Processes and Pipes

Distribution date: October 1 (Tuesday)
Due Date: October 15 (Tuesday)

Instructions (These apply to all programming assignments)

1. Programming assignment is to be done by each student working alone.
2. All programming assignment should be submitted on the due date, unless prior arrangements are made.
3. Show all your work, and state any special assumptions you make about a problem.
4. Carefully comment your code.
5. Describe your implementation, testing, features, known bugs etc. in your writeup.
6. **Neatness counts**!

Part 1: Process Groups

In lecture two we discussed process groups briefly. For more information see section 2.4 from *Interprocess Communication in Unix* by J. Gray. Perform the following exercises to learn about process groups.

Start a process, such as an editor or web browser, as a background task by using an ’&’ at the end of the command line. For example:

```
    emacs&
```

will start the emacs editor as a background process, and the terminal prompt will return immediately for more commands.

Use the ps command (**man ps** for more options) to list your processes, and their group ids.

```
        ps  -j    or   ps j
```

Look at the process ids, parent process ids and group ids of your processes. Type in the following program:

```
#include <stdio.h>
#include<sys/types.h>
#include<unistd.h>

/* Displaying process group ID information  */

void main(void)
```

```
{
   int i;
   printf("\n\nInitial process  \t PID %6d \t PPID %6d \t GID %6d\n\n",
                   getpid(),getppid(),getpgid(0));
   for(i=0; i<3; ++i)
     if  (fork() == 0)         /* Generate some process */
     printf("New process     \t\t  PID  %6d  \t  PPID %6d \t  GID  %6d \n",
                    getpid(), getppid(), getpgid(0));
   sleep(15);
}
```

Compile the program and execute it in the background as follows:

```
   gcc -o p21 p21.c
   p21 > p21.log  &
```

Use the ps command to determine the processes created, and their relationships. Use the Unix script command to record the output of the ps command and draw the process relationships as in the diagram of Fig 2.6 p.46. Record the program name in addition to the PID for each process and identify the groups by circling all processes within a group. Identify the group leaders. Use fflush() after printf() statements in the above program to see the effect of line buffering.

Part 2: Forest Fire Simulation with a Chain of Processes

Execute a forest fire simulation with multiple processes computing the simulation in parallel. Create the processes as a chain with bi-directional communication between the processes using unnamed pipes.

Your program will be invoked with 5 commands line arguments as given below.

```
fire  <procs> <iterations> <pb> <pa> <output freq>
```

where procs is the number of processes to create, iterations is the number of simulation iterations to execute, pb is the probability that a live tree will start burning, pa is the probability that a new live tree will grow to replace a dead tree, and output freq is the number of iterations between output of the simulation space. For example, if output freq = 1 then the simulation space should be displayed every iteration , if output freq = 2 then the simulation space should be displayed every other iteration.

For example

```
    fire   4 20 .01 .30 2
```

The simulation space for the forest should be represented as a two dimensional array. Each array element can be in one of three states: alive, burning or dead. The problem size for this assignment will be fixed at 32×62 (32 rows each of length 62). For each iterations, values from the previous iteration are used to derive the values for the current iteration as follows :

1. If a live tree is next to a burning tree, it burns; otherwise it catches fire with probability pb. We consider only four neighbours (left, right, top, bottom) of a tree as next to the tree.

2. A burning tree dies.
3. A dead tree has probability pa of being replace by a live tree.

See the attached pseudo code for details on determining the state of each tree and also details on the ASCII display of the simulation space. **Assume:** initially all trees are alive.

You should develop your program in three phases:

1. Write a sequential program to perform the forest fire simulation with a single process.
2. Implement a two process version of the program, where each process simulates half of the forest (half of the rows) and exchange border row data with the other process, via a pipe.
3. Support any number of process (up to 16). Such that for a given process p_i, it must exchange border information with process p_{i-1} and p_{i+1}

The result of the forest simulation should be output to a file, one file for each simulation iteration. Since each process is responsible for the simulation of one part of the forest, the result from each process should be properly ordered in the output file.

Makefile:

Along with your program you must submit a makefile for compiling your code. The TA should be able to compile your forest fire simulation program by typing:

```
make fire
```

You will be penalized (loss of points) for not having a working makefile. See the links in the course website for a brief introduction to makefiles.

Files:

fire.c (or .cc): the forest fire simulation in part 2 above; fire.h: an optional header file; Makefile.

Writeup:

Include the output from the ps command when the p21 program was executing and the diagram of process relationships, process groups, etc as described above. Look at the man pages for kill and describe how the kill command can be used to terminate all of the process created when p21 executes.

Describe your forest fire program and any known bugs in the program. The explanation and description part of your writeup should be no more than two pages.

Writeup format requirements: use Times New Roman, font size 11, margins: 1.0 inch, top, bottom, left, right.

Submission:

Gamma

For students working on gamma you must electronically submit your program using the turning program. Time stamp will be used to determine whether the assignment was submitted on time. Also submit a hard copy of your source code and sample output of the simulation along with your writeup.

To electronically submit type the following from gamma:

```
/nfs/valhalla/users28/ceg434ta/turnin/turnin-proj1 f1 f2...
```

Extra Credit: Make a new version of the fire program (ffire) that uses FIFOs instead of unnamed pipes. Describe in your write-up the differences between your pipe based implementation and your FIFO implementation. Submit a separate copy of code using FIFO implementation.

Grading: The assignment is worth 15 points (15% of your overall grade):

- Writeup:
  - Part 1: 3
  - Part 2: 2
- Program Verfication
  - Sequential: 2 points
  - 2 processes: 4 points
  - $p > 2$ processes: 4 points
- Extra credits: upto 2 points
- Deductions:
  - Lateness: 0.75 points/day upto 4 days
  - Nonconformance to specifications: upto 2 points

Pseudo Code

```
procedure  nextstate(var u: subgrid; i,j: integer);
{  1 <= i <= m,  1 <= j <= m }
var x: real;
begin
  case u[i,j] of
      alive:
           if (u[i-1,j] = burning ) or
              (u[i+1,j] = burning ) or
              (u[i,j+1] = burning ) or
              (u[i,j-1] = burning )
           then u[i,j] := burning
           else
               begin
                   random(x);
                   if x<=pb then
                       u[i,j] := burning
               end;
      burning:
          u[i,j] := dead;
      dead:
           begin
               random(x);
               if x<=pa then
                   u[i,j] := alive
               end;
           end;
  end;

var u: grid;
procedure display(var u:grid);
var i,j: integer;
begin
    for i := 1 to n do
       begin
           for j:= 1 to n do
               case u[i,j] of
                   alive: write('+');
                   burning: write('*');
                   dead: write (' ' );
               end;
           writeln;
       end
end
```