ECE437/CS481 Operating Systems, Fall 2014

Programming Assignment #03 (3%)

Due: Wednesday, 09/17/2014

1. **(20%) Process and process IDs.**

   Perform the following steps using the command "ps" with various options:

   a) Find at least 5 PIDs and their corresponding states and command names of your user processes ("ps -l" may help). If you don't have so many, create some and suspend them.

   b) Repeat for part 1.a), but include a Zombie process (you can on purpose to create one with a few lines of C program: a parent process creates a child process and exits without wait; the Child process sleeps for a while then exits.) Who will be parent process of your Zombie process?

   c) List at least 3 system processes whose PID < 10, show their corresponding states and command names ("ps –el" may help).

   d) Trace one of your processes all way to the root by following the parent-child (P-C) process relation ("grep" with PID may help). List the path from PID=0 all way to your own process. What is the depth of your path? Verify your newly created P-C tree path with command "pstree".

2. **(20%) Process creation with fork and wait.**

   Let's use the ***fork*** system call to create Linux processes. Write a program, named as myfork.c, which (named as P0) creates two child processes (named as C1 & C2), where the 2nd child process C2 will not be created until the completion of the 1st child process C1. **At the end** (upon completion of both C1 and C2), the parent process should use ***printf*** to display "from P0: own PID=xxxx, PID of C1=yyyy, PID of C2=zzzz, total elapsed time in milliseconds=wwww". In every child process, **at the beginning**, display "from CX: own PID=xxxx, parent's PID=yyyy", then call fib(20). Your answer to this question includes the source code as well as the copy of the output (all pasted together, no use of separated files). Function fib(n) is a slow/recursive implementation of Fibonacci numbers:

```
int fib(int x) { /* slow/recursive implementation of Fib */
   int i, rint = (rand()%30);  double dummy;
      for (i=0; i<rint*100; i++) {dummy=2.345*i*8.765/1.234;}
      if (x==0) return(0); else if (x==1) return(1); else return(fib(x-1)+fib(x-2));
}
```

3. **(10%) Program execution with execl.**

   Based on your work done in Q2, continue to learn another system call ***execl.*** Write a new program, named as *myexec.c*, in which the first child process will execute *date* after calling fib(20), and the second child process will execute ***who*** after calling fib(20). Your answer to this question includes the source code as well as the copy of the output.

4. **(10%) Process performance measurement with gettimeofday and getrusage.**

   Provided is a general matrix multiplication program (mm437_seq.c) with use of system calls: gettimeofday, getrusage, and System V IPC package's shared memory part. You are free to edit/change the source code as you like and run the program for N (matrix size) =750, 1500, and 3000, respectively to fill up the table in milliseconds:

   a) By checking your system information from "/proc/cpuinfo", list # of cores, its clock speed, and load average (use "uptime")

   b) Fill in the blanks (in msecs):

N	Elapsed Time	USR Time	SYS Time	USR+SYS Time	# of Context Switches
750					
1500					
3000					

If you run your program multiple times for the same value of N, pick one of them to fill in the table.
Number of Context Switches includes both voluntary and involuntary context switches. Also note that:

[1] The provided program takes N and M as inputs, but didn't use M (number of child processes)
[2] The provided program uses System V IPC's shared memory although we do not need it since there is no child process.
[3] You can use "ipcs" and "ipcrm" commends to check and clean IPCs your program created.
[4] Here is a brief explanation about time & clocks.

Wall Clock time
- the amount of time the process takes to run, sometimes called elapsed time.

User (CPU) time
- the CPU time attributed to user instructions

System (CPU) time
- the CPU time attributed to the kernel when it executes on behalf of the process

5. **(40%) Multiple Processes Creation with shared memory.**

Based on the provided matrix multiplication program (mm437l_seq.c), write your program as mm437_par.c to create M child processes to do matrix multiplication. Run the program for N (matrix size) =750, 1500, and 3000, respectively to fill in the tables in milliseconds:

5.a) For M=1:

N	Elapsed Time	USR Time	SYS Time	USR+SYS Time	# of Context Switches
750					
1500					
3000					

5.b) For M=2:

N	Elapsed Time	USR Time	SYS Time	USR+SYS Time	# of Context Switches
750					
1500					
3000					

5.c) For M=4:

N	Elapsed Time	USR Time	SYS Time	USR+SYS Time	# of Context Switches
750					
1500					
3000					

5.d) Copy/paste your program's output for 5.a) and 5.c) with N=3000.

5.e) Compare "Elapsed Time" and "USR+SYS Time" from 5.a) and 5.c) for N=3000. Discuss your findings.

P.S. For PA03, submit your write-up to answer all questions onto learn.unm.edu under PA03 (please show your name in every page of your submission, and name your file as "YourLastName_YourFirstNameInitial_PA3") and submit your program "mm437_par.c" from Q5 onto PA03b as a single file (.c source).