Programming Assignment #02

1. Fibonacci using Fork and inter-process communications (65%)

Let's use the fork system call to create processes to compute the Fibonacci sequence of a given number. Write a program that takes two parameters: "-F ***n*** –S ***m***", where ***n*** is an input number to compute the ***n***th number of the Fibonacci sequence and ***m*** specifies a computing threshold.

- ❖ In general, fib(x)=fib(x-1)+fib(x-2), where fib(0) = 0 and fib(1) = 1
- ❖ **Case 1**: if (n-1) > m or (n-2)>m, your program must use the fork system call to recursively spawn child processes to compute the next Fibonacci numbers.
  - ✓ When the child process completes its Fibonacci number computation, it will send the result back to its parent process based on inter-process communications (e.g., shared memory, message queue (mq_open), named pipe (mkfifo), unnamed pipe (pipe()) etc.).
  - ✓ The parent process will receive the results by receiving data/messages from its child processes.
  - ✓ The parent process should wait for all its child processes terminated and then close the inter-process communications if existing.
- ❖ **Case 2**: if (n-1) <= m and (n-2)<=m, your program recursively call fib_seq(x-1) to compute the next Fibonacci numbers, where

```
int fib_seq(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_seq(x-1)+fib_seq(x-2));
}
```

Your program should follow the given input/output format: "myfib "-F ***n*** –S ***m***". You should output/print ONLY the ***n***th Fibonacci number. See the example below.

```
$ myfib –F 6 –S 6
8
$ myfib –F 6 –S 3
8
```

The above two cases will output the same result "8". The former one (**Case 2**) will do every computation in sequential while the later one (**Case 1**) will create several child processes to do the computation. In a multicore system, the later one should take less time to complete since multiple processes can utilize multiple cores. Hint:

- ❖ You can use C-library call "getopt" to handle your commend line input.
- ❖ Be careful about how many child processes to be created.

2. Inter-process communication using signals (25%)

Write a program that will utilize signal handlers to intercept keyboard interrupts to manage control of a child process. Your main program should fork and execve a new child process to run the "yes" command (use man yes to learn more). You should then register signal handlers for both ctrl-c and ctrl-z in your main program. The ctrl-z signal handler should toggle the child "yes" process, i.e., stop the child process if it is running, or resume the child process if it is stopped, all while keeping the main process running. The ctrl-c signal should kill the child process, and then exit the main process. In both signal handlers, add print statements indicating what the program is doing – i.e. “ctrl-z caught, stopping child process”. Your main program should sit in an infinite loop once the setup has been completed. It should only exit when the user hits ctrl-c, after cleaning up the child process.

3. Submission requirement (10%)

- If submitting as a group, please include a paragraph detailing each member's assigned tasks, similar to PA01.
- Please name your two source codes as PA02fib.c and PA02sig.c, respectively, and submit them into PA02. We will compile and test your program from there. Also, please submit a report including the snapshots and the explanations of your outcomes for the two source codes.