# CENG 334 - Intr. to Operating Systems
Programming HW #1 - Process Management

*Due 14-3-2010 23:59*

## Introduction

This homework is on process management basics, pipes, signals and other UNIX system calls. You will learn how to work with these by simulating the cell phone traffic of a small base station on a remote island. The system to be simulated is composed of two types of entities, the base station and the users. In order to make a call, a user makes a request from the base station. The base station responds with either acknowledgement (meaning the call is made), or failure (meaning the system is busy at the moment). Being a banana republic and all, the technology in this island is not very advanced, and the base station allows only one user to make call at any instant. The users who try to make a call while the base station is busy serving someone else will simply get a "busy" tone, wait a random amount of time, and try again, until they make their calls.

## Specifications

Your program will be run with a command line argument indicating the number of users. The communication between the child processes (users) and the main process (the base station) will be done via pipes. So, you should set up pipes (using the pipe() system call) prior to spawning child processes. You should then fork() n child processes. (n being the number of users), and print their pid's line by line to standard output. For instance,

```
> ./hw1 5
<pid of child process 0>
<pid of child process 1>
..
<pid of child process 4>
```

After the processes are created, the parent process will take the role of the base station and the child processes will take the role of the users. The system will then begin simulating the traffic.

### Child processes (Users)

Each child process will read is input from user$_i$.inp (i.e. first child will use user0.inp, second user1.inp and so on) and write its output to user$_i$.out. The first line of the input file contains the number of calls this user intends to make. The lines following this one contains the duration of each call.

For example, a user intending to make 3 calls with durations 5, 10 and 3 seconds respectively will have an input file like:

```
3
5
10
3
```

When a user begins, it immediately attempts to make the first call. The user then waits response from the base station and behaves accordingly. An important point is that users send call duration in the request so that main process knows how long a call will take.

- If the base station responds with acknowledgement, the user will then sleep for the duration of the call, and then try to make the second call after waking up.
- If the base station responds with failure, the child process will then sleep a random amount time between [1-5] seconds, and retry. (Note that waiting times are multiples of a second, so like there's no 2.5 second waits). The user keeps trying and waiting until the call is finally made.

Users keep going on like this until they have no more calls to make. Before terminating, the user processes must write what they were doing at each second to their output files. For example, a user that tries to make two calls which are 3 and 2 seconds but waits 1 second between them must write:

```
calling
calling
calling
waiting
calling
calling
```

## Main process (The base station)

The base station receives call requests from users from pipes. It is responsible for scheduling calls from the users, so that at most one user is making a call at any instant. When a user attempts a call, the base station should acknowledge if nobody else is making a call at the moment. After a call attempt is acknowledged, all other call attempts must be denied for the duration of the call.

When the main process receives a request, it must log the time the request came and details of the request to standard output like this,

```
time 0 user 0 accepted for 3 seconds
time 0 user 1 rejected
time 0 user 2 rejected
time 2 user 1 rejected
time 3 user 2 accepted for 1 seconds
```

In this example, the system contains 3 users, each attempting to make a call at startup (time 0). The base station accepts user0’s request and rejects others. After 2 seconds, user1 tries to make a call again but since the first user is still making a call, user1 gets rejected again. At time 3, the system finally becomes available and the request coming from user2 is accepted.

One important point is that, main process must not block while reading from a user's pipe, as this would result in other users' starvation. You can use the select() system call to monitor input on multiple pipes.

Main process should terminate after all the child processes are terminated. (See signal() and SIGCHLD). You should use wait() system call on terminated child processes. so that child processes are not left at “zombie” state.

## Notes

- Most of the system calls are rather new to you, so start early and don't try to implement everything at once. Man pages is your friend.
- The specifics of the protocol between the main process and child process is up to you, but keeping it simple is recommended.
- Keep in mind that time/sleep functions might not always be accurate, so you should either define a confidence interval (i.e. treat 1900ms as 2 seconds), or do all timing calculations on the main process and and synchronize users by, for example, sending them a special signal every second.
- It is crucial that the outputs produced by the main process and child processes are consistent. Don't let them get out of sync.
- The program must be written in C. C++ will not be accepted.
- Submissions will be through the COW system.
- You should turn in a single tar file containing your source/header files, and a Makefile. We will test your program on ink machines by issuing the following commands.

  ```
  tar -xvf hw1.tar
  make
  ./hw1 3
  ```
- Before submitting, make sure that your code compiles/works on inek machines and your tar file does not contain directories, since the above commands will not work if it’s the case.
- Late submission policy
  - You have total 7 late submission days for 3 homeworks, you won’t lose any grades for submissions during this period.
  - After that, late submission penalty is 5*day*day.