Pre-Final Mock Practice Sample Questions

Each question is of either 2 or 3 marks.

- Show the utility of dup2() system call using an example.
- Differentiate working of execl() and execv() system calls using a suitable program.
- How are process groups and control terminals useful in managing foreground processes?
- How are read() and write() system calls useful in implementation of pipes?
- Give pseudocode showing the working of signal() system call.
- Show the utility of pipe() system call using an example.
- Why do we use a pipe() system call?
- Summarize the working of alarm() system call.
- How do we use SIGCONT and SIGSTOP signals? Give an example to show their handling.
- What do we primarily achieve by handling SIGUSR1 and SIGUSR2 signals? How do we implement them?
- List different ways in which a kill() system call is used.
- Write a C program invoker.c that runs another C program invoked.c. Both programs just execute a simple printf statement as given below.

  printf("I am a C program represented by a process ID %d\n", getpid());

  Clearly specify the code for invoker.c and invoked.c, and show its test run.

  (Hint: make use of one of the exec calls)
- Give an implementation of a pipe connecting two shell commands. (Hint: use pipe, dup2 calls)
- What do we mean by a blocked signal? How are blocked signals handled in a typical 32-bit Linux system?
- How are pending signals handled in a typical 64-bit Linux system?
- Write a C program using signal() system call showing how to ignore a SIGQUIT signal, and then restore it back to execute its default handler.
- Give output of the following program (along with line numbers that get executed as per your specified output) with reasons. Will both printf() statements gets executed atleast once? In not, what changes may be needed in the code given below.

```
int main()
{
 1.      pid_t child = fork();
 2.      if(child==0)
             {
 3.              while(1){
 4.                  printf("I am the child process\n");
                 }
 5.      else {
 6.              kill(child,SIGINT);
 7.              printf("I am the parent process that killed child\n");
               }
 8.      return 0;
 }
```