COSC1112/1114: Operating Systems Principles

Tutorial 02 (week 03)

1. Describe the differences among short-term, medium-term and long-term scheduling.

2. Describe the actions taken by a kernel to context-switch between processes.

3. Including the initial parent process, how many processes are created by the following program:

```
#include <stdio.h>
#include <unistd . h>
#include <stdlib.h>

int main ( ) {
        /* for a child process */
        fork ( ) ;
        /* fork another child process */
        fork ( ) ;
        /* and fork another */
        fork ( ) ;
        return EXIT_SUCCESS;
}
```

4. Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

5. Explain the output of line A of the following program:

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

int value = 5;

int main ( ) {
        pid_t pid ;
        /* for a child process */
        pid = fork ( ) ;
        if (pid == 0) { /* chi ld */
                value += 15;
                return EXIT_SUCCESS;
        } else if ( pid >0) { /* parent process */
                wait (NULL) ;
                printf ( "PARENT: value = %d\n" , value ) ; /* line A */
                return EXIT_SUCCESS;
        }
}
```

6. Explain the role of the init process on UNIX and Linux systems in regards to process termination.

7. Explain the circumstances when the line of code marked printf("LINE J") is reached.

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

int main(){
        pid t pid;
        /* fork a child process */
        pid = fork();

        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                return 1;
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls","ls",NULL);
                printf("LINE J");
        }
        else { /* parent process */
                /* parent will wait for the child to complete */
                wait(NULL);
                printf("Child Complete");
        }
        return 0;
}
```