Linux Lab week 6 and 7

Week 6:

Write a C program to emulate the Unix ls-l command.

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
        int pid; // process id
        pid = fork(); // create another process

        if (pid < 0)
        { // fail
        perror("Fork failed\n");
        exit(-1);
        }
        else if (pid == 0)
        { // child
        execlp("/bin/ls", "ls", "-l", NULL); // execute ls
        perror("execlp failed\n");          // exec function returns only on error
        exit(-1);
        }
        else
        { // parent
        wait(NULL); // wait for child
        printf("\nChild complete\n");
        exit(0);
        }
}
```

Input and output:

```
guest-glcbIs@ubuntu:~$gcc –o lsc.out lsc.c
guest-glcbIs@ubuntu:~$./lsc.out
total 100
-rwxrwx—x 1 guest-glcbls guest-glcbls 140 2012-07-06 14:55 f1
```

drwxrwxr-x 4 guest-glcbls guest-glcbls 140 2012-07-06 14:40 dir1
child complete

Write a C program to list for every file in a directory, its inode number and fill name

Code:

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

int main(int argc, char *argv[])
{
        if (argc == 2)
        {
        char d[50];
        memset(d, 0, sizeof(d));  // Use memset instead of bzero (deprecated)
        strcat(d, "ls ");
        strcat(d, "-i ");
        strcat(d, argv[1]);
        system(d);
        }
        else
        {
        printf("\nInvalid number of inputs\n");
        }

        return 0;  // Added a return statement for consistency
}
```

Input and output:

```
student@ubuntu:~$ mkdir dd
student@ubuntu:~$ cd dd
student@ubuntu:~/dd$ cat >f1
hello
^z
student@ubuntu:~/dd$ cd
student@ubuntu:~$gcc –o flist.out flist.c
student@ubuntu:~$./flist.out dd
hello
46490 f1
```

Write a C Program that demonstrates redirection of standard output to a file.
EX:: ls>f1.

Code:

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

int main(int argc, char *argv[])
{
        char d[50];

        if (argc == 2)
        {
        memset(d, 0, sizeof(d));  // Use memset instead of bzero (deprecated)
        strcat(d, "ls > ");
        strcat(d, argv[1]);
        system(d);
        }
        else
        {
        printf("\nInvalid number of inputs\n");
        }

        return 0;
}
```

Output:

```
student@ubuntu:~$ gcc –o std.out std.c
student@ubuntu:~$ls
downloads   documents   listing.c   listing.out   std.c   std.out
student@ubuntu:~$ cat > f1
^z
student@ubuntu:~$./std.out f1
student@ubuntu:~$cat f1
downloads
documents
listing.c
listing.out
std.c
std.out
```

Week 7:

Write a C program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen

Code:

```
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h> // Include the necessary header

int main(void)
{
        int pid;
        int status;

        printf("Hello World!\n");

        pid = fork();

        if (pid == -1)
        {
        perror("bad fork");
        exit(1);
        }

        if (pid == 0)
        printf("I am the child process.\n");
        else
        {
        wait(&status); /* parent waits for child to finish */
        printf("I am the parent process.\n");
        }

        return 0;
}
```

Output:

```
student@ubutnu:$gcc –o child.out child.c
student@ubutnu: ./child.out
Hello World!
I am the child process.
I am the parent process
```

Write a C program to create a Zombie process.

Code:

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

```
#include <sys/types.h>
#include <unistd.h>
int main ()
{
  int pid_t,child_pid;
  child_pid = fork ();
  if (child_pid > 0) {
          sleep (60);
  }
  else {
          exit (0);
  }
  return 0;
}
```

Output:

```
guest-glcbIs@ubuntu:~$gcc  zombie.c
guest-glcbIs@ubuntu:~$./a.out
```

Then command prompt will wait for some time(60 sec) and then again command prompt will appear later.