Assignment 3

Que-1: Explain the role of the init process on UNIX and Linux systems in regard to process termination.

Ans: A process that has terminated, but whose parent has not yet called wait(), is known as a zombie process. All processes transition to this state when they terminate, but generally they exist as zombies only briefly. Once the parent calls wait(), the process identifier of the zombie process and its entry in the process table are released. But if a parent did not invoke wait() and instead terminated, thereby leaving its child processes as orphans. Linux and UNIX address this scenario by assigning the init process as the new parent to orphan processes. The init process periodically invokes wait(), thereby allowing the exit status of any orphaned process to be collected and releasing the orphan's process identifier and process-table entry.

Que-2 Including the initial parent process, how many processes are created by the program ?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
   int i;
   for (i = 0; i < 4; i++)
   fork();
   return 0;
}
```

Ans: There are four fork() system call. Therefore, 16 processes are created, 1 parent process and 15 child processes.

Que-3: Using the program, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
 pid t pid, pid1;
        /* fork a child process */
        pid = fork();
        if (pid < 0) { /* error occurred */
             fprintf(stderr, "Fork Failed");
             return 1;
         }
         else if (pid == 0) { /* child process */
             pid1 = getpid();
             printf("child: pid = %d",pid); /* A */
```

```
            printf("child: pid1 = %d",pid1); /* B */
        }
        else { /* parent process */
            pid1 = getpid();
            printf("parent: pid = %d",pid); /* C */
            printf("parent: pid1 = %d",pid1); /* D */
            wait(NULL);
        }
        return 0;
    }
```

Ans: A) child: pid = 0
B) child: pid1 = 2603
C) parent: pid = 2603
D) parent: pid1 = 2600

Que-4: Using the program, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
int i;
pid t pid;
        pid = fork();
        if (pid == 0) {
          for (i = 0; i < SIZE; i++) {
              nums[i] *= -i;
              printf("CHILD: %d ",nums[i]); /* LINE X */
          }
        }
        else if (pid > 0) {
          wait(NULL);
          for (i = 0; i < SIZE; i++)
              printf("PARENT: %d ",nums[i]); /* LINE Y */
        }
          return 0;
    }
```

Ans: Because the child is a copy of the parent, any changes the child makes will occur in its copy of the data and won't be reflected in the parent. As a result :
The output values by the child at line X are 0, -1, -4, -9, -16.
The output values by the parent at line Y are 0, 1, 2, 3, 4.