Lab: 1

Name: Vinayak Kumar **Roll No. : 2K15CSUN01063** **Class: CSE - 4C**

Laboratory Objective: To implement Basic system calls of UNIX Operating System
Learning Outcome: Familiarity with the use of basic calls of UNIX

Programs using the following system calls of UNIX operating system :-

1. Program to use fork() call

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

int main(int argc, char const *argv[])
{
        printf("To show Fork()\n");
        fork();
        printf("Hello World\n");
        return 0;
}

//Note-----------
//Displays "Hello World" Twice
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./fork
To show Fork()
Hello World
Hello World
Hello World
Hello World
```

2. Program to create Child Process using fork() call

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

int main(int argc, char const *argv[]){
      int pid;
      pid=fork();
      if(pid>0){
        printf("Parent Process ID PID is %d\n",pid);
      }
}
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./fork_child
Parent Process ID PID is 4847
```

3. Program to implement getpid() call

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

int main(){
        int pid;
        pid =getpid();
        printf("Process ID is %d\n",pid);
}
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./process_identification
Process ID is 5142
```

4. Program to implement getppid() call

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

int main(int argc, char const *argv[])
{
        int ppid;
        ppid=getppid();
        printf("Parent Process ID is %d\n", ppid);
        return 0;
}
// Retreiving Parent Process ID
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./parent_child
Parent Process ID is 4009
```

5. getpid() & getppid() call to get process ID and parent process ID

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

int main(int argc, char const *argv[])
{
        int pid=fork();
        if(pid == 0){
                printf("I am child Process with ID = %d\n",getpid() );
                printf("My Parent Process ID is = %d\n",getppid() );
        }
        else{
                printf("I am Parent with ID = %d\n",getpid() );
                printf("My Parent Process ID is = %d\n",getppid() );
                //this getppid() will return PPID of shell($ps -el)
        }
        return 0;
}
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./child_parent_id
I am Parent with ID = 4978
My Parent Process ID is = 4009
I am child Process with ID = 4979
My Parent Process ID is = 1555
```

6. Program to implement Orphan Process

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

int main(int argc, char const *argv[]){
        int pid=fork();
        if(pid == 0){
                printf("I am child Process with ID = %d\n",getpid() );
                printf("My Parent Process ID is = %d\n",getppid() );
                sleep(20);
                printf("I am child Process with ID = %d\n",getpid() );
                printf("My Parent Process ID is = %d\n",getppid() );
        }
        else{
                printf("I am Parent with ID = %d\n",getpid() );
                printf("My Parent Process ID is = %d\n",getppid() );
        }
        return 0;
}
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./orphan_process
I am Parent with ID = 5237
My Parent Process ID is = 4009
I am child Process with ID = 5238
My Parent Process ID is = 5237
```

7. Program to implement Zombie Processes

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

int main(int argc, char const *argv[])
{
        if (fork() > 0)
        {
                printf("parent\n");
                sleep(50);
        }
        return 0;
}

//Run this as background process

// $ps -el  ; and press enter
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./zombies &
[1] 5445
vinayak@vinayak:~/Desktop/OS LAB/t1$ parent
ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

```
0 S  1000  5445  4009  0  80   0 -  1089 hrtime pts/4    00:00:00 zombies
1 Z  1000  5446  5445  0  80   0 -     0 exit   pts/4    00:00:00 zombies <defunct>
0 R  1000  5447  4009  0  80   0 -  7229 -      pts/4    00:00:00 ps
```

8. Program To check process stats

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

int main(int argc, char const *argv[])
{
        long i;
        printf("Process ID is %d\n", getpid());
        for (i = 0; i < 4000000; i++);
                printf("i is %ld\n", i);
        return 0;
}

//Note ----
// Run this twice as background Process
// ./a.out &
// $ps -e press enter
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./pro2 &
[1] 5598
vinayak@vinayak:~/Desktop/OS LAB/t1$ Process ID is 5598
i is 4000000
./pro2 &
[2] 5599
[1]   Done                    ./pro2
vinayak@vinayak:~/Desktop/OS LAB/t1$ Process ID is 5599
i is 4000000
ps -e
  PID TTY          TIME CMD
 5578 ?        00:00:00 kworker/u16:2
 5600 pts/4    00:00:00 ps
[2]+  Done                    ./pro2
```

9. Program to implement exec(), perror(), exit(), and wait() calls

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char const *argv[], char *env[]){
        pid_t my_pid, parent_pid, child_pid;
        int status;

        my_pid=getpid();
        parent_pid = getppid();
        printf("\nParent: my pid is %d\n\n", my_pid);
        printf("Parent: My Parent's pid is %d\n\n", parent_pid);

        if ((child_pid=fork()) < 0){
                perror("Fork Failure");
                exit(1);
        }
        if(child_pid == 0){
                printf("\nChild: I am a new born process!\n\n");
                my_pid=getpid();
                parent_pid = getppid();
                printf("Child: My PID is %d\n\n", my_pid);
                printf("Child: My Parent's PID is %d\n\n", parent_pid);
                printf("Child: I will sleep 3 Seconds and then execute - date -command \n\n");
                sleep(3);
                printf("Child: Now, I woke up and am excuting date command \n\n");
                execl("/bin/date","date",0,0);
                perror("execl() failure!\n\n");
                printf("This print is after execl() and should not have been executed if execl was successful!\n\n");
                exit(1);
        }
        else{
                printf("\nParent: I created a child Process.\n\n");
                printf("Parent: My child's PID is %d\n\n",child_pid );
                system("ps -acefl | grep ercal");
                printf("\n\n");
                wait(&status);
                printf("Parent: My child is dead. I am going to leave.\n\n");
        }
        return 0;
}
```

```
vinayak@vinayak:~/Desktop/OS LAB/t1$ ./all

Parent: my pid is 5860

Parent: My Parent's pid is 4009


Parent: I created a child Process.

Parent: My child's PID is 5861


Child: I am a new born process!

Child: My PID is 5861

Child: My Parent's PID is 5860

Child: I will sleep 3 Seconds and then execute - date -command

0 S vinayak   5862  5860 TS   19 -  1127 wait   23:26 pts/4    00:00:00 sh -c ps -acefl | grep ercal
0 S vinayak   5864  5862 TS   19 -  3556 pipe_w 23:26 pts/4    00:00:00 grep ercal


Child: Now, I woke up and am excuting date command

Wed Mar  1 23:27:03 IST 2017
Parent: My child is dead. I am going to leave.
```