Experiment No. 1 Roll No. 79 Date: 01/02/2019

1. System Calls

Aim:

To implement system calls for the given scenario

Programs:

Task 1. Write a program p1.c that forks a child and prints the following:
Parent: My process ID is: 12345
Parent: The child process ID is: 15644
and the child process prints
Child: My process ID is: 15644
Child: The parent process ID is: 12345.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main() {
      int p;
      printf("Hello, World!\n");
      p = fork();
      printf("Welcome to Batch E, %d\n", p);
      if(p == 0 && p != -1) {
            printf("I'm child and my PID is: %d\n",getpid());
            printf("I'm child and my PPID is: %d\n",getppid());
      } else if(p > 0 && p != -1) {
            printf("I'm parent and my PID is: %d\n",getpid());
            printf("I'm parent and my PPID is: %d\n",getppid());
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p1.exe
Hello, World!
Welcome to Batch E, 20995
I'm parent and my PID is: 20994
I'm parent and my PPID is: 20920
Welcome to Batch E, 0
I'm child and my PID is: 20995
I'm child and my PPID is: 20994
```

Task 2. Write another program p2.c that does exactly same as in previous exercise, but the parent process prints it's messages only after the child process has printed its messages and exited. Parent process waits for the child process to exit, and then prints its messages and a child exit message, Parent: The child with process ID 12345 has terminated.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
pid_t p = -1;
void main() {
      printf("Hello, World!\n");
      p = fork();
      wait();
      printf("Welcome to Batch E, %d\n", p);
      if(p == 0 && p != -1) {
            printf("I'm child and my PID is: %d\n",getpid());
            printf("I'm child and my PPID is: %d\n",getppid());
      } else if(p > 0 && p != -1) {
            printf("I'm parent and my PID is: %d\n",getpid());
            printf("I'm parent and my PPID is: %d\n",getppid());
            printf("Parent: Child with PID %d has terminated\n",p);
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p2.exe
Hello, World!
Welcome to Batch E, 0
I'm child and my PID is: 21593
I'm child and my PPID is: 21592
Welcome to Batch E, 21593
I'm parent and my PID is: 21592
I'm parent and my PPID is: 20920
Parent: Child with PID 21593 has terminated
```

Task 3. A program mycat.c, available as part of this lab, reads input from stdin and writes output to stdout. Write a program p3.c that executes the binary program mycat (compiled from mycat.c) as a child proces of p3.

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main() {
	int p;
	printf("Welcome to task 3 program in PDF\n");
	p = fork();
	wait();
	if(p == 0 && p != -1) {
		printf("I am child process and my PID is : %d\n",getpid());
		char *args[] = {"./MYCAT",NULL};
		execvp(args[0], args);
	} else {
		printf("I am parent process and my PID is : %d\n",getpid());
		printf("My parent PID is: %d\n",getppid());
	}
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p3.exe
Welcome to task 3 program in PDF
I am child process and my PID is : 21686
Hello from MYCAT
Hello from MYCAT
Exiting
Exiting
^C
```

Task 4a. Write a program p4a.c which takes a file name as command line argument. Parent opens file and forks a child process. Both processes write to the file, “hello world! I am the parent” and “hello world! I am the child”. Verify that the child can write to the file without opening it. The parent process should wait for the child process to exit, and it should display and child exit message.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc, char *argv[]) {
      int p;
      FILE *fptr;
      printf("Welcome to task 4a in PDF.\n");
      if(argc == 1) {
            printf("File name is not passed in the command line
arguments\n");
      } else {
            fptr = fopen(argv[1], "w");
            if(fptr == NULL) {
                  printf("Error! File cannot be opened.\n");
                  exit(1);
            }
            p = fork();
            wait();
            if(p == 0 && p != -1) {
                  printf("IN child: PID = %d\n",getpid());
                  printf("IN child: Writing to file.\n");
                  fprintf(fptr,"%s","Hello World! I am child!");
                  printf("IN child: Write complete\n");
            } else {
                  printf("IN parent: File opened.\n");
                  printf("IN parent: PID = %d\n",getpid());
                  printf("IN parent: Writing to file.\n");
                  fprintf(fptr,"%s","\nHello World! I am parent!");
                  printf("IN parent: Write complete\n");
                  printf("IN parent: Closing file\n");
                  fclose(fptr);
            }
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p4a.exe
sample.txt
Welcome to task 4a in PDF.
IN parent: File opened.
IN child: PID = 21780
IN child: Writing to file.
IN child: Write complete
IN parent: PID = 21779
IN parent: Writing to file.
IN parent: Write complete
IN parent: Closing file
```

Task 4b. Write a program p4b.c which takes a file name as a command line argument. The program should print content of the file to stdout from a child process. The child process should execute the mycat program. Cannot use any library functions like printf, scanf, cin, read, write in the parent of child process.

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
int fd, p;
char content[2000];
void main(int argc, char *argv[]) {
      printf("Hello from task 4b in PDF\n");
      if(argc == 1) {
            printf("Please pass file name as command line argument.\n");
            exit(1);
      } else {
            fd = open(argv[1], O_RDONLY | O_CREAT | O_TRUNC);
            if(fd == -1) {
                  printf("Error! In opening file\n");
                  exit(0);
            }
            dup2(fd, 0); // duplicate in stdin
            close(fd);
            p = fork();
            if (p > 0) {
                  printf("Parent: File opened. fd = %d\n", 0);
            } else {
                  printf("Child: Execute mycat to stdout file descriptor
stored.\n");
                  char *args[]={"./MYCAT", NULL};
                  execlp(args[0],args);
            }
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p4b.exe
sample.txt
Hello from task 4b in PDF
Parent: File opened. fd = 0
Child: Execute mycat to stdout file descriptor stored.
```

Task 5. Write a program p5.c to demonstrate the state of process as an orphan. The program forks a process, and the parent process prints the following messages.
Parent: My process ID is: 12345
Parent: The child process ID is: 15644
The child process prints message
Child: My process ID is: 15644
Child: The parent process ID is: 12345
Sleeps for few seconds, and it prints the same messages one more time. By the time the child process wakes from sleep, the parent process should have exited.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main() {
      int p;
      p = fork();
      if(p == 0 && p != -1) {
            printf("I'm child and my PID is: %d\n",getpid());
            printf("I'm child and my PPID is: %d\n",getppid());
            sleep(15);
            printf("I'm child and my PID is: %d\n",getpid());
            printf("I'm child and my PPID is: %d\n",getppid());
      } else if(p > 0 && p != -1) {
            printf("I'm parent and my PID is: %d\n",getpid());
            printf("I'm parent and my PPID is: %d\n",getppid());
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p5.exe
I'm parent and my PID is: 21997
I'm parent and my PPID is: 20920
I'm child and my PID is: 21998
I'm child and my PPID is: 21997
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ps
  PID TTY          TIME CMD
20920 pts/0    00:00:00 bash
21998 pts/0    00:00:00 p5.exe
22000 pts/0    00:00:00 ps
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ I'm child
and my PID is: 21998
I'm child and my PPID is: 2475
```

Task 6. Write a program p6.c to demonstrate the presence of zombie processes. The programs forks a process, and the parent process prints the message
Parent : My process ID is: 12345
Parent : The child process ID is: 15644
and the child process prints the message
Child : My process ID is: 15644
Child : The parent process ID is: 12345
After printing the messages, the parent process sleeps for 1 minute, and then waits for the child process to exit. The child process waits for some keyboard input from user after displaying the messages, and then exits.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main() {
      int p; int random;
      p = fork();
      if(p == 0 && p != -1) {
            printf("I'm child and my PID is: %d\n",getpid());
            printf("I'm child and my PPID is: %d\n",getppid());
            printf("Enter some random integer: \n");
            scanf("%d",&random);
            printf("\nYou entered: %d\n",random);
            exit(0);
      } else if(p > 0 && p != -1) {
            sleep(20);
            printf("I'm parent and my PID is: %d\n",getpid());
            printf("I'm parent and my PPID is: %d\n",getppid());
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p6.exe
I'm child and my PID is: 13436
I'm child and my PPID is: 13435
Enter some random integer:
12
You entered: 12
^Z
[1]+  Stopped                 ./p6.exe
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ps
  PID TTY          TIME CMD
```

```
12412 pts/0    00:00:00 bash
13435 pts/0    00:00:00 p6.exe
13436 pts/0    00:00:00 p6.exe <defunct>
13441 pts/0    00:00:00 ps
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ fg
./p6.exe
I'm parent and my PID is: 13435
I'm parent and my PPID is: 12412
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ps
  PID TTY          TIME CMD
12412 pts/0    00:00:00 bash
13444 pts/0    00:00:00 ps
```

Task 7a. Write a program p7a.c that takes a number n as a command line argument and creates n child processes recursively, i.e., parent process creates first child, the first child creates second child, and so on. The child processes should exit in the reverse order of the creation, i.e., the inner most child exits first, then second inner most, and so on. Print the order of creation of the child process, and their termination order as shown in sample output.

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc, char *argv[]) {
      int p = 0;
      int i = 0;
      printf("Hello to task 7A in PDF.\n");
      int parentId = getpid();
      printf("Process : PID = %d; PPID = %d -->
created\n",getpid(),getppid());
      if(argc == 1) {
            printf("Value of n is not passed.\n");
            exit(1);
      } else {
            int n = atoi(argv[1]);
            for(i = 1; i <= n; i++) {
                  p = fork();
                  wait(0);
                  if(p == 0 && p != -1) {
                        printf("Process : PID = %d; PPID = %d -->
created\n",getpid(),getppid());
                  } else {
                        printf("Process : PID = %d; PPID = %d -->
exited\n",getpid(),getppid());
                        exit(0);
                  }
            }
            printf("Process : PID = %d; PPID = %d -->
exited\n",getpid(),getppid());
      }
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p7a.exe
5
Hello to task 7A in PDF.
Process : PID = 22066; PPID = 20920 --> created
Process : PID = 22067; PPID = 22066 --> created
Process : PID = 22068; PPID = 22067 --> created
Process : PID = 22069; PPID = 22068 --> created
Process : PID = 22070; PPID = 22069 --> created
Process : PID = 22071; PPID = 22070 --> created
Process : PID = 22071; PPID = 22070 --> exited
Process : PID = 22070; PPID = 22069 --> exited
Process : PID = 22069; PPID = 22068 --> exited
Process : PID = 22068; PPID = 22067 --> exited
Process : PID = 22067; PPID = 22066 --> exited
Process : PID = 22066; PPID = 20920 --> exited
```

Task 7b. Write a program p7b.c that takes a number n as command line argument and creates n child processes sequentially, i.e., the first parent process creates all children in a loop without any delays. Let the child processes sleep for a small random duration, and print the creation and exit order of the child processes.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
int pid, j;
int main(int argc, char *argv[]) {
      int i,n,current_pid;
      unsigned int seed = n;
      srand(seed);
      n = atoi(argv[1]);
      for(i=0, j=n; i<n; i++,j--) {
            int random = rand_r(&seed);
            pid = fork();
            if(pid == -1) {
                  printf("Fork failed.\n");
            } else if(pid == 0){
                  if (i==0) {
                        printf("Parent is: %d\n", getppid());
                        printf("Number of children: %d\n", n);
                  }
                  int time = (random % 10 + 1);
                  printf("Child %d is created. Parent (%d). Sleep %d\n",
getpid(), getppid(), time);
                  sleep(time);
                  return getpid();
            }
      }
      // Need to wait for all
      for(i=0; i<n; i++) {
            int a = wait(NULL);
            printf("Child %d exited\n", a) ;
      }
      printf("Parent exited.\n");
      return 0;
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p7b.exe
5
Parent is: 4253
Child 4255 is created. Parent (4253). Sleep 10
Number of children: 5
Child 4254 is created. Parent (4253). Sleep 5
Child 4257 is created. Parent (4253). Sleep 5
Child 4256 is created. Parent (4253). Sleep 3
Child 4258 is created. Parent (4253). Sleep 4
Child 4256 exited
Child 4258 exited
Child 4254 exited
Child 4257 exited
Child 4255 exited
Parent exited.
```

Task 7c. Write another program p7c.c, that creates n child processes similar to sequential creation of p7b.c. Further, each child also does the similar sleep action for a random duration. In the parent process, set it up so that child termination is in the reverse order of sequential creation.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
int pid, j;
int main(int argc, char *argv[]) {
      int i,n,current_pid;
      n = atoi(argv[1]);
      for(i=0, j=n; i<n; i++,j--) {
            pid = fork();
            if(pid == -1) {
                  printf("Fork failed.\n");
            } else if(pid == 0) {
                  if (i==0) {
                        printf("Parent is: %d\n", getppid());
                        printf("Number of children: %d\n", n);
                  }
                  printf("Child %d is created. Parent (%d). Sleep %d\n",
getpid(),getppid(), j);
                  sleep(j);
                  return getpid();
            }
      }
      // Need to wait for all
      for(i=0; i<n; i++) {
            int a = wait(NULL);
            printf("Child %d exited\n", a) ;
      }
      printf("Parent exited.\n");
      return 0;
}
```

Output:

```
jimil@jimil-Lenovo-G50-80:~/Documents/SPIT/Sem 4/OS/Pracs/Lab 1$ ./p7c.exe
5
Parent is: 4398
Child 4400 is created. Parent (4398). Sleep 4
```

```
Number of children: 5
Child 4399 is created. Parent (4398). Sleep 5
Child 4402 is created. Parent (4398). Sleep 2
Child 4401 is created. Parent (4398). Sleep 3
Child 4403 is created. Parent (4398). Sleep 1
Child 4403 exited
Child 4402 exited
Child 4401 exited
Child 4400 exited
Child 4399 exited
Parent exited.
```

Conclusion:

Thus, we implemented solutions for the various scenarios provided to us and used a number of system calls in our implementations.