**OS LAB WEEK 3**

**ADIT LUHADIA**

**190911112**

**IT A**

**LAB 5 - SYSTEM CALLS FOR PROCESS CONTROL**

1. Write a C program to create a child process. Display different messages in parent process and child process. Display PID and PPID of both parent and child process.

// 190911112

// Adit Luhadia

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main()

{

    printf("190911112\nAdit Luhadia\n");

    pid\_t pid;

    if (fork() == 0)

    {

        printf(" |-> Inside child \n");

        printf(" |-> Pid of child: %d \n", getpid());

        printf(" |-> PPid of child: %d \n", getppid());

        printf(" |-> Child Terminated \n\n");

        return 1;

    }

    else if (fork > 0)

    {

        printf("Inside parent \n");

        pid = wait(NULL);

        printf("Pid of Parent: %d \n", getpid());

        printf("PPid of Parent: %d \n", getppid());

        printf("Parent Terminated \n");

    }

    else

    {

        printf("Fork Failed");

    }

}

Text

Description automatically generated with medium confidence

2. Write a C program to accept a set of strings as command line arguments. Sort the strings and display them in a child process. Parent process should display the unsorted strings only after the child displays the sorted list.

// 190911112

// Adit Luhadia

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*\*argv)

{

    int i, j;

    printf("190911112\nAdit Luhadia\n");

    if (fork() == 0)

    {

        char \*temp;

        printf("Sorted Array: ");

        for (i = 1; i < argc; i++)

        {

            for (j = 1; j < argc - i; j++)

            {

                if (strcmp(argv[j], argv[j + 1]) > 0)

                {

                    temp = argv[j];

                    argv[j] = argv[j + 1];

                    argv[j + 1] = temp;

                }

            }

        }

        for (i = 1; i < argc; i++)

        {

            printf("%s ", argv[i]);

        }

        printf("\n");

        return 1;

    }

    if (fork() > 0)

    {

        wait(NULL);

        printf("Original Array: ");

        for (i = 1; i < argc; i++)

        {

            printf("%s ", argv[i]);

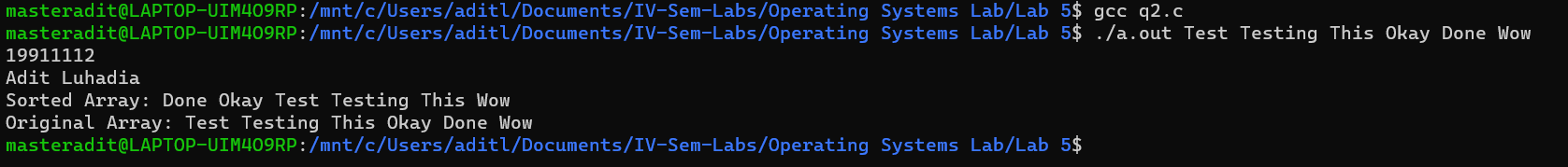
        }

        printf("\n");

        return 1;

    }

}



3. Write a C program to read N strings. Create two child processes, each of this should perform sorting using two different methods (bubble, selection, quicksort etc). The parent should wait until one of the child process terminates.

// 190911112

// Adit Luhadia

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*\*argv)

{

    printf("190911112\nAdit Luhadia\n");

    int i, j;

    int fork1 = fork();

    if (fork1 == 0)

    {

        char \*temp;

        printf("Sorted Array by child 1: ");

        for (i = 1; i < argc; i++)

        {

            for (j = 1; j < argc - i; j++)

            {

                if (strcmp(argv[j], argv[j + 1]) > 0)

                {

                    temp = argv[j];

                    argv[j] = argv[j + 1];

                    argv[j + 1] = temp;

                }

            }

        }

        for (i = 1; i < argc; i++)

        {

            printf("%s ", argv[i]);

        }

        printf("\n");

        return 1;

    }

    else if (fork1 > 0)

    {

        int fork2 = fork();

        if (fork2 == 0)

        {

            char \*key;

            for (i = 2; i < argc; i++)

            {

                key = argv[i];

                j = i - 1;

                while (j >= 1 && strcmp(argv[j], key) > 0)

                {

                    argv[j + 1] = argv[j];

                    j = j - 1;

                }

                argv[j + 1] = key;

            }

            printf("Sorted Array by child 2: ");

            for (i = 1; i < argc; i++)

            {

                printf("%s ", argv[i]);

            }

            printf("\n");

            return 1;

        }

        else if (fork2 > 0)

        {

            // sleep(3);

            wait(NULL);

            printf("Original Array: ");

            for (i = 1; i < argc; i++)

            {

                printf("%s ", argv[i]);

            }

            printf("\n");

            return 1;

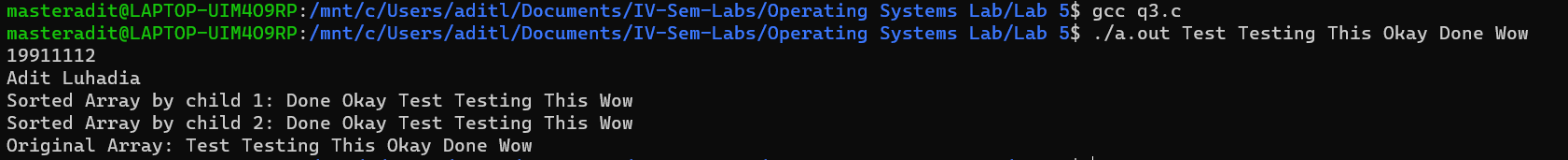
        }

        else

            printf("fork failed");

    }

}



**LAB 6 - SYSTEM CALLS FOR PROCESS CONTROL**

1. Develop a menu driven C program to implement the following process scheduling algorithms: preemptive-SJF, RR and non-preemptive priority scheduling algorithms.

// 190911112

// Adit Luhadia

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main()

{

    printf("190911112\nAdit Luhadia\n");

    pid\_t pid;

    if (fork() == 0)

    {

        printf(" |-> Inside child \n");

        printf(" |-> Pid of child: %d \n", getpid());

        printf(" |-> PPid of child: %d \n", getppid());

        printf(" |-> Child Terminated \n\n");

        return 1;

    }

    else if (fork > 0)

    {

        printf("Inside parent \n");

        pid = wait(NULL);

        printf("Pid of Parent: %d \n", getpid());

        printf("PPid of Parent: %d \n", getppid());

        printf("Parent Terminated \n");

    }

    else

    {

        printf("Fork Failed");

    }

}

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated