Results:

**Program code**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main() {

    if(!fork())

    {

        printf("hello! I'm from child and my process id is %d.\nMy parent process id is %d\n",getpid(),getppid());

    }

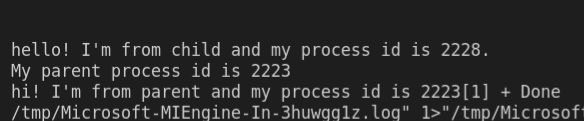
    else{

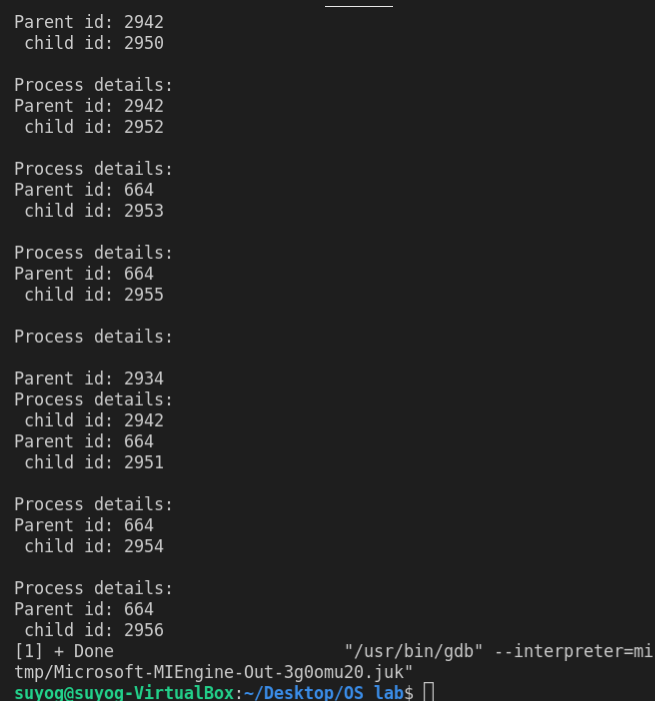
        printf("hi! I'm from parent and my process id is %d",getpid());

    }

    return 0;

}





There are eight process in total out of which seven are child. Their process id along with their parent id is shown as above.

2.

**Program code:**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#define DEL1 10000

#define DEL2 50000

int main() {

    int i,d;

    char c;

    if(!fork())

    {

        for(c='a';c<='z';c++)

        {

            printf("%c\t",c);

            fflush(stdout);

            for(d=0;d<DEL1;d++);

        }

        exit(0);

    }else{

        for(i=0;i<=10;i++)

        {

            printf("%i\n",i);

            fflush(stdout);

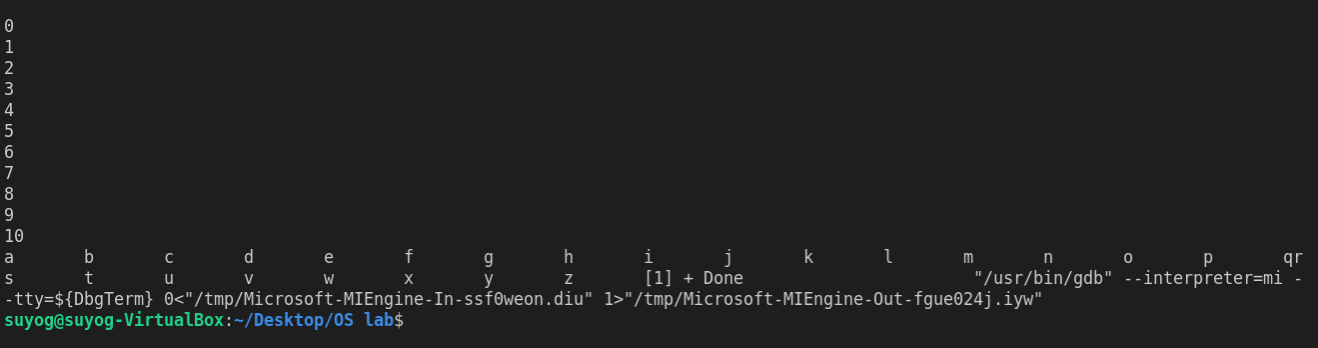
            for(d=0;d<=DEL2;d++);

        }

        exit(0);

    }

}



In this program, the parent executes its program and the DEL2 holds the screen and again child executes and holds screen from DEL1.

When we increase the delays DEL1 and DEL2 then we saw that the output of two process overlapped. This is due to non-uniform delays and both process are running at the same time.

3.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    int pid;

    int fork();

    if (pid == 0)

    {

        printf("i’m the child, my process ID is %d\n", getpid());

        printf("I’m the child and my parent’s ID is %d\n", getppid());

        sleep(5);

        printf("(after sleep)i’m the child, my process ID is %d\n", getpid());

        printf("(after sleep)Im the child and my parent’s ID is %d\n", getppid());

    }

    else

    {

        //anchor

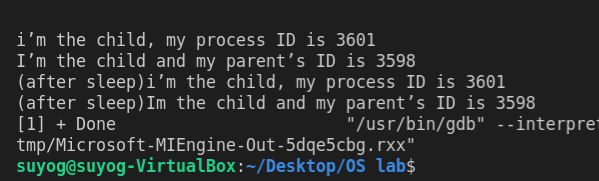
        sleep(10);

        printf("I’m the parent, my process ID is %d", getpid());

        printf("the parent’s process ID is %d", getppid());

    }

}



Here, the execution of fork() was successful so the value of pid will be 0 and hence only the statements under if with condition pid=0 is executed.

Adding the sleep statement delayed the output by some time . The child process is an orphan process and a new parent is given to it by the system.

4.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    int i = 0, j = 0, pid;

    pid = fork();

    if (pid == 0)

    {

        for (i = 0; i < 500; i++)

            printf("% d\t", i);

    }

    else

    {

        if (pid > 0)

        {

            //anchor

            //wait(0);

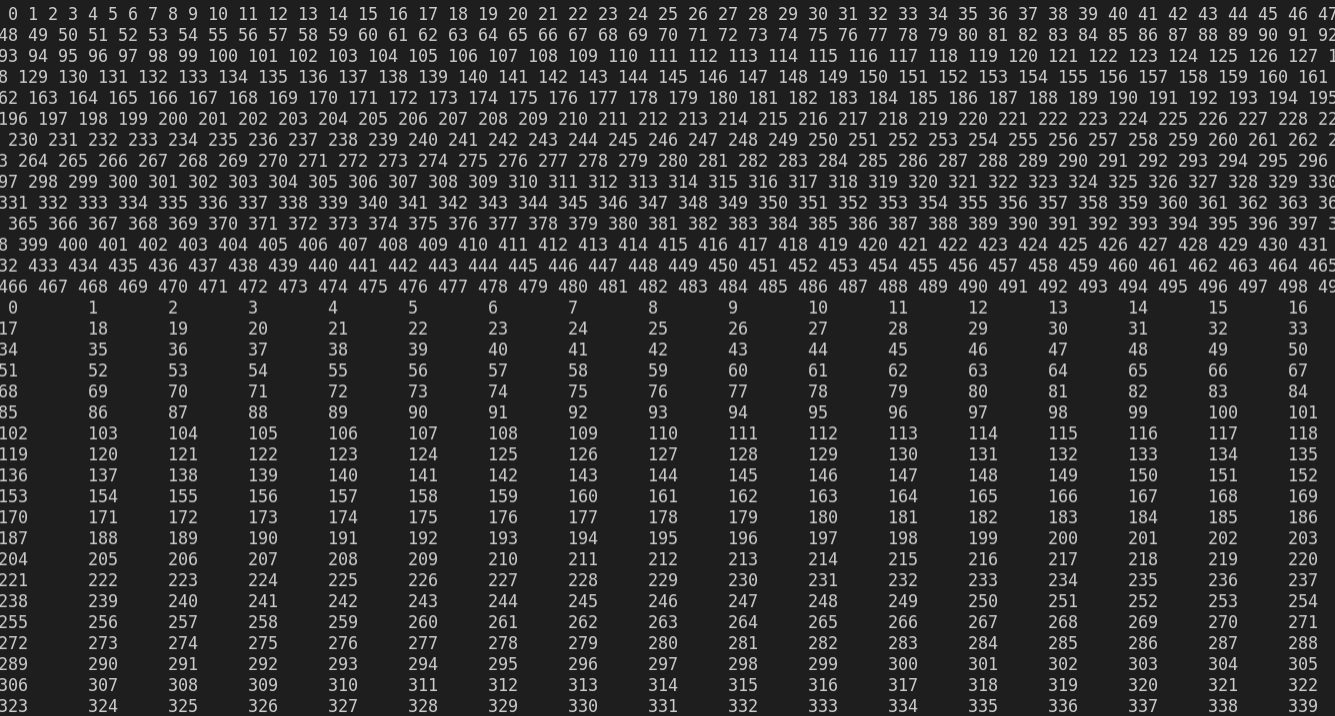
            for (j = 0; j < 500; j++)

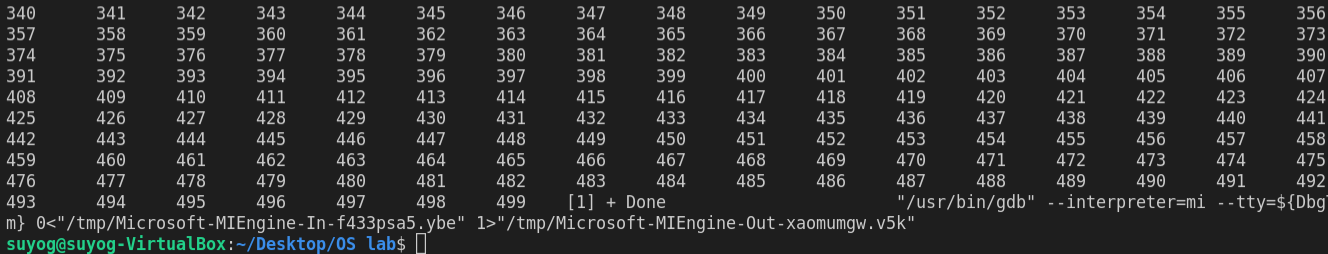
                printf("% d", j);

        }

    }

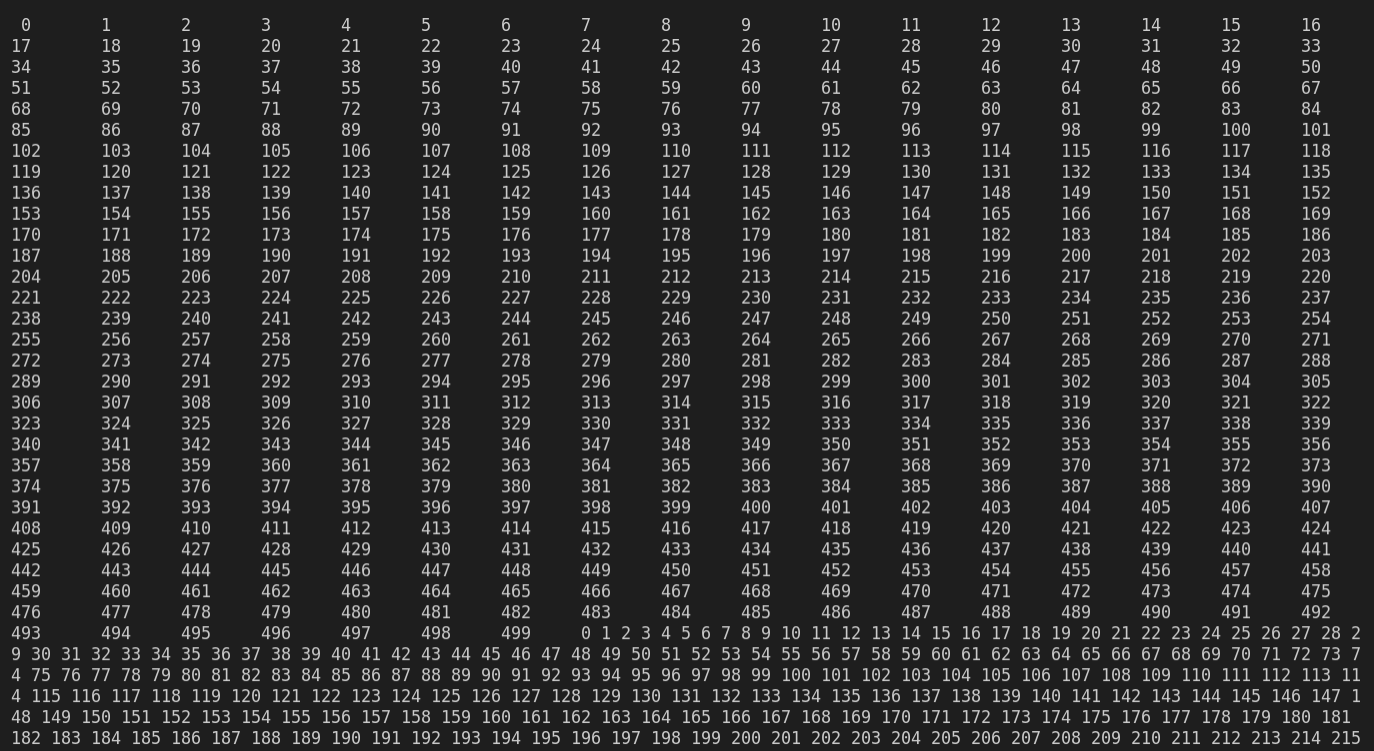
}

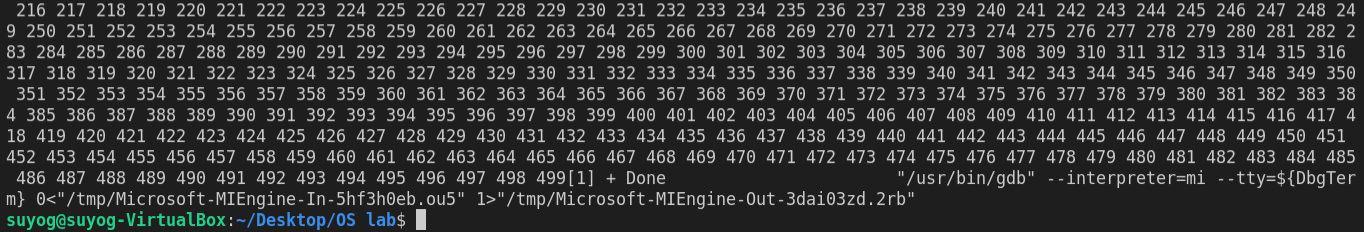




In this program, first the parent executed and then the execution of the child took place.

After edit





When the wait(0) statement is added, then the parent waits for its child to complete its execution and the executes itself. Only the parent can wait for the child and not the other way around.