

OPERATING SYSTEMS LAB - PRACTICAL 6 - PIPES

Name - Sakshi Soni

Roll No - 13

AIM :

Develop an application using Inter-Process Communication using pipes:
Write a C program in Linux to generate the Fibonacci series in the child process and pass numbers to parent process using pipe and parent process should separate the odd and even numbers.

PROGRAM AND OUTPUT :

PROGRAM 1

```
#include <unistd.h>
int main(){
    int pid, pip[2];
    char instring[20];
    pipe(pip);
    pid = fork();
    if (pid == 0) /* child : sends message to parent*/
    {
        /* send 7 characters in the string, including end-of-string */
        write(pip[1], "Hi Mom!", 7);
    }
    else /* parent : receives message from child */
    {
        /* read from the pipe */
        read(pip[0], instring, 7);
    }
}
```

PROGRAM 2

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define MSG_LEN 64
int main(){
    int result;
    int fd[2];
    char *message="Linux World!!!";
    char recvd_msg[MSG_LEN];
    result = pipe (fd); //fd[0] is for reading fd[1] is for writing
    if (result < 0) {
        perror("pipe ");
        exit(1);
    }
    //writing the message into the pipe
    result=write(fd[1],message,strlen(message));
    if (result < 0) {
        perror("write");
        exit(2);
    }
    //Reading the message from the pipe
    result=read (fd[0],recvd_msg,MSG_LEN);
    if (result < 0) {
        perror("read");
        exit(3);
    }
}
```

```
winter@windows:~/OS/prac6$ gedit pipes2.c
^C
winter@windows:~/OS/prac6$ gcc pipes2.c
winter@windows:~/OS/prac6$ ./a.out
Linux World!!!
winter@windows:~/OS/prac6$
```

PROGRAM 3 - AIM

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
int fibonacci(int n);
```

```
void separateOddEven(int num);
```

```
int main() {
```

```
    int pipefd[2];
```

```
    pid_t pid;
```

```
    if (pipe(pipefd) == -1) {
```

```
        perror("pipe");
```

```
        return 1;
```

```
    }
```

```
    pid = fork();
```

```
    if (pid < 0) {
```

```
        perror("fork");
```

```
        return 1;
```

```
    }
```

```
    if (pid == 0) { // Child process
```

```
        close(pipefd[0]); // Close the read end of the pipe
```

```

int fibCount;
printf("Enter the number of Fibonacci numbers to generate: ");
scanf("%d", &fibCount);

int fibNumber;
for (int i = 1; i <= fibCount; i++) {
    fibNumber = fibonacci(i);
    write(pipefd[1], &fibNumber, sizeof(fibNumber));
}

close(pipefd[1]); // Close the write end of the pipe
printf("Child process exiting.\n");
} else { // Parent process
    close(pipefd[1]); // Close the write end of the pipe

    int number;
    printf("Even numbers: ");
    while (read(pipefd[0], &number, sizeof(number)) > 0) {
        if (number % 2 == 0)
            printf("%d ", number);
    }
    printf("\n");

    printf("Odd numbers: ");
    lseek(pipefd[0], 0, SEEK_SET); // Reset file offset
    while (read(pipefd[0], &number, sizeof(number)) > 0) {
        if (number % 2 != 0)
            printf("%d ", number);
    }
    printf("\n");

    close(pipefd[0]); // Close the read end of the pipe
    printf("Parent process exiting.\n");
}

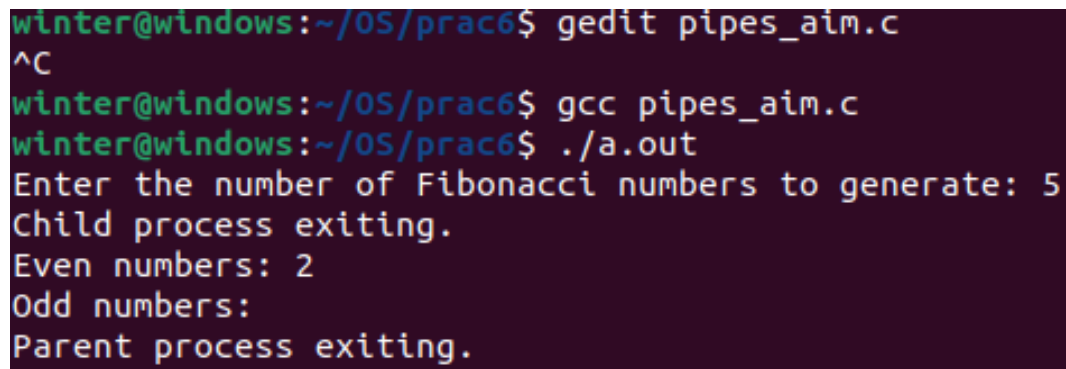
```

```

    return 0;
}

int fibonacci(int n) {
    if (n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

```



```

winter@windows:~/OS/prac6$ gedit pipes_aim.c
^C
winter@windows:~/OS/prac6$ gcc pipes_aim.c
winter@windows:~/OS/prac6$ ./a.out
Enter the number of Fibonacci numbers to generate: 5
Child process exiting.
Even numbers: 2
Odd numbers:
Parent process exiting.

```

PROGRAM 4 - CREATION OF A ONE WAY PIPE

```

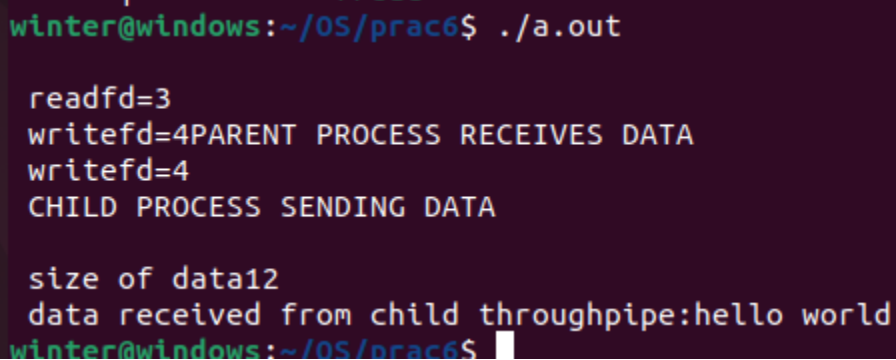
#include<stdio.h>
#include<stdlib.h>
int main(){
    int pipefd[2],n,pid;
    char buff[100];
    pipe(pipefd);
    printf("\n readfd=%d",pipefd[0]);
    printf("\n writefd=%d",pipefd[1]);
    pid=fork();
    if(pid==0){
        close(pipefd[0]);
        printf("\n CHILD PROCESS SENDING DATA\n");
        write(pipefd[1],"hello world",12);
    }
}

```

```

}
else{
close(pipefd[1]);
printf("PARENT PROCESS RECEIVES DATA\n");
n=read(pipefd[0],buff,sizeof(buff));
printf("\n size of data%d",n);
printf("\n data received from child throughpipe:%s\n",buff);
}
return 0; }

```



A terminal window with a dark purple background. The prompt is 'winter@windows:~/OS/prac6\$'. The command './a.out' has been executed. The output is as follows:

```

readfd=3
writefd=4PARENT PROCESS RECEIVES DATA
writefd=4
CHILD PROCESS SENDING DATA

size of data12
data received from child throughpipe:hello world
winter@windows:~/OS/prac6$

```

PROGRAM 5 - CREATION OF A TWO WAY PIPE

```

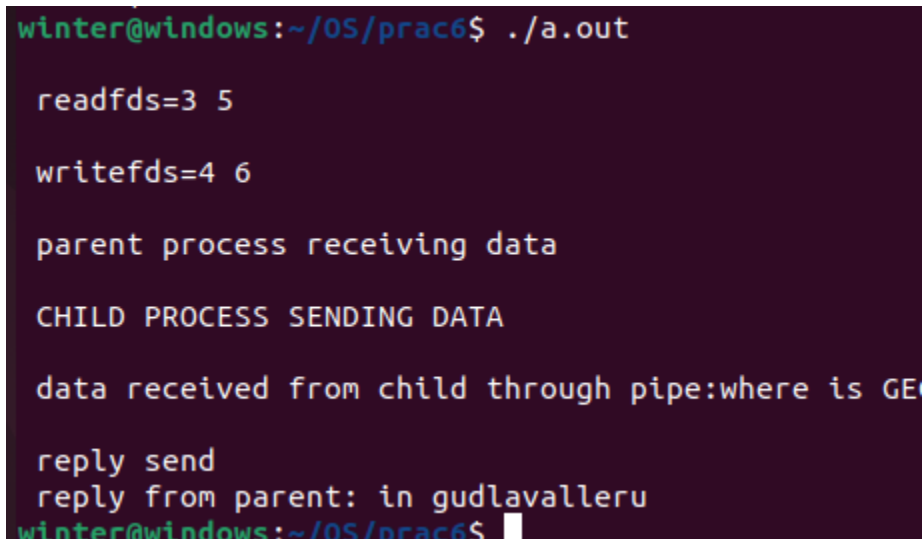
#include<stdlib.h>
main(){
    int p1[2],p2[2],n,pid;
    char buf1[25],buf2[25];
    pipe(p1);
    pipe(p2);
    printf("\n readfds=%d %d\n",p1[0],p2[0]);
    printf("\n writefds=%d %d\n",p1[1],p2[1]);
    pid=fork();
    if(pid==0){
        close(p1[0]);
        printf("\n CHILD PROCESS SENDING DATA\n");
        write(p1[1],"where is GEC",25);
        close(p2[1]);
    }
}

```

```

    read(p2[0],buf1,25);
    printf(" reply from parent:%s\n",buf1);
    sleep(2);
}
else{
    close(p1[1]);
    printf("\n parent process receiving data\n");
    n=read(p1[0],buf2,sizeof(buf2));
    printf("\n data received from child through pipe:%s\n",buf2);
    sleep(3);
    close(p2[0]);
    write(p2[1]," in gudlavalleru",25);
    printf("\n reply send\n");
}
}

```



A terminal window with a dark purple background. The prompt is 'winter@windows:~/OS/prac6\$'. The command './a.out' has been executed. The output is as follows:

```

readfds=3 5
writefds=4 6

parent process receiving data

CHILD PROCESS SENDING DATA

data received from child through pipe:where is GE

reply send
reply from parent: in gudlavalleru
winter@windows:~/OS/prac6$

```

PROGRAM 6

```

#include<stdio.h>
#include<unistd.h>
#include<string.h>
int main(){

```

```

int p1[2],p2[2],p3[2],p4[2];
int i,j=0,k=0,l=0;
char r[10],s[10],t[10],u[10];
printf("\t PROCESS 1. ENTER THE STRING");
scanf("%s",r);
pipe(p1);
pipe(p2);
write(p1[1],r,sizeof(r));
write(p2[1],r,sizeof(r));
int a=fork();
if(a==0){
    printf("\n\t PROCESS 2: it splits the given string\n");
    read(p1[0],r,sizeof(r));
    int n=strlen(r);
    for(i=0;i<n/2;i++){
        s[i]=r[i];
    }
    for(i=n/2;i<=n;i++){
        t[j++]=r[i];
    }
    pipe(p3);
    pipe(p4);
    write(p3[1],s,sizeof(s));
    write(p4[1],t,sizeof(t));
    int b=fork();
    if(b==0){
        printf("p4 %d\t",getpid());
        printf("p2 %d\n",getppid());
        read(p3[0],s,sizeof(s));
        printf("\t PROCESS 4: sub string \t %s \t",s);
        printf("no of char=%d \n",strlen(s));
    }
    else{
        int c=fork();
        if(c==0){

```



```

        printf("p5 %d\t",getpid());
        printf("p2 %d\n",getppid());
        read(p4[0],t,sizeof(t));
        printf("\t PROCESS 5:sub string \t %s \t",t);
        printf("no of char=%d \n",strlen(t));
    }
    else{
        wait();
        printf("p2 %d\t",getpid());
        printf("p1 %d\n",getppid());
    }
}
}
else{
    wait();
    int d=fork();
    if(d==0){
        printf("p3 %d\t",getpid());
        printf("p1 %d\n",getppid());
        read(p2[0],r,sizeof(r));
        for(i=strlen(r)-1;i>=0;i--){
            u[l++]=r[i];
        }
        for(i=0;i<strlen(r);i++){
            if(u[i]==r[i])
                k++;
            else
                continue;
        }
        if(k==strlen(r))
            printf("\t PROCESS 3: the given string is
palindrome\n");
        else
            printf("\t PROCESS 3: the given string is not palindrome\n");
    }
}

```

```

    else{
        printf("p1 %d\t",getpid());
        printf("kernal %d\t\n",getppid());
    }
}
return 0;
}

```

```

47 |                                     wait();
    |                                     ^~~~
winter@windows:~/OS/prac6$ ./a.out
    PROCESS 1. ENTER THE STRING sakshi

    PROCESS 2: it splits the given string
p4 7110 p2 7109
    PROCESS 4: sub string      sak      no of char=3
p5 7111 p2 7109
    PROCESS 5: sub string      shi      no of char=3
p2 7109 p1 7108
p1 7108 kernal 4400
p3 7112 p1 7108
    PROCESS 3: the given string is not palindrome
winter@windows:~/OS/prac6$

```

QUESTION

Write a C program to implement the following game. The parent program P first creates two pipes, and then spawns two child processes C and D. One of the two pipes is meant for communications between P and C, and the other for communications between P and D. Now, a loop runs as follows. In each iteration (also called round), P first randomly chooses one of the two ags: MIN and MAX (the choice randomly varies from one iteration to another). Each of the two child processes C and D generates a random positive integer and sends that to P via its pipe. P reads the two integers; let these be c and d. If P has chosen MIN, then the child who sent the smaller of c and d gets one point. If P has chosen MAX, then

the sender of the larger of c and d gets one point. If c = d, then this round is ignored. The child process who first obtains ten points wins the game. When the game ends, P sends a user-defined signal to both C and D, and the child processes exit after handling the signal (in order to know who was the winner). After C and D exit, the parent process P exits. During each iteration of the game, P should print appropriate messages (like P's choice of the ag, the integers received from C and D, which child gets the point, the current scores of C and D) in order to let the user know how the game is going on. Name your program childsgame.c .

PROGRAM -

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>

#define MAX_SCORE 10

int c_score = 0;
int d_score = 0;

int generateRandomNumber() {
    return rand() % 100 + 1; // Generate a random positive integer between 1
    and 100
}

void handleSignal(int signal) {
    printf("Game ended. Final scores:\n");
    printf("Child C score: %d\n", c_score);
    printf("Child D score: %d\n", d_score);
```

```

    exit(0);
}

void playGame(int pipe1[2], int pipe2[2]) {
    signal(SIGUSR1, handleSignal);

    int round = 1;

    while (1) {
        printf("\nRound %d:\n", round);
        round++;

        int min_or_max = rand() % 2; // 0 for MIN, 1 for MAX
        printf("Parent's choice: %s\n", min_or_max == 0 ? "MIN" : "MAX");

        int c_number, d_number;
        read(pipe1[0], &c_number, sizeof(c_number));
        read(pipe2[0], &d_number, sizeof(d_number));

        printf("Numbers received from child C: %d\n", c_number);
        printf("Numbers received from child D: %d\n", d_number);

        if (c_number == d_number) {
            printf("Numbers are equal. Ignoring this round.\n");
            continue;
        }

        int point_winner;
        if (min_or_max == 0) {
            point_winner = (c_number < d_number) ? 1 : 2;
        } else {
            point_winner = (c_number > d_number) ? 1 : 2;
        }

        if (point_winner == 1) {

```

```

        c_score++;
        printf("Child C gets 1 point!\n");
    } else {
        d_score++;
        printf("Child D gets 1 point!\n");
    }

    printf("Current scores:\n");
    printf("Child C score: %d\n", c_score);
    printf("Child D score: %d\n", d_score);

    if (c_score >= MAX_SCORE || d_score >= MAX_SCORE) {
        printf("Game ended. Final scores:\n");
        printf("Child C score: %d\n", c_score);
        printf("Child D score: %d\n", d_score);
        break;
    }
}

kill(getpid(), SIGUSR1); // Send signal to exit gracefully
}

int main() {
    srand(time(NULL)); // Seed random number generator

    int pipe1[2], pipe2[2];
    pid_t c_pid, d_pid;

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("pipe");
        return 1;
    }

    c_pid = fork();

```

```

if (c_pid == -1) {
    perror("fork");
    return 1;
}

if (c_pid == 0) { // Child process C
    close(pipe1[0]); // Close read end of pipe1
    close(pipe2[0]); // Close read end of pipe2

    int number;
    while (1) {
        number = generateRandomNumber();
        write(pipe1[1], &number, sizeof(number));
        sleep(1); // Sleep to avoid rapid generation of numbers
    }
} else {
    d_pid = fork();

    if (d_pid == -1) {
        perror("fork");
        return 1;
    }

    if (d_pid == 0) { // Child process D
        close(pipe1[0]); // Close read end of pipe1
        close(pipe2[0]); // Close read end of pipe2

        int number;
        while (1) {
            number = generateRandomNumber();
            write(pipe2[1], &number, sizeof(number));
            sleep(1); // Sleep to avoid rapid generation of numbers
        }
    } else { // Parent process P
        close(pipe1[1]); // Close write end of pipe1
    }
}

```

```

        close(pipe2[1]); // Close write end of pipe2

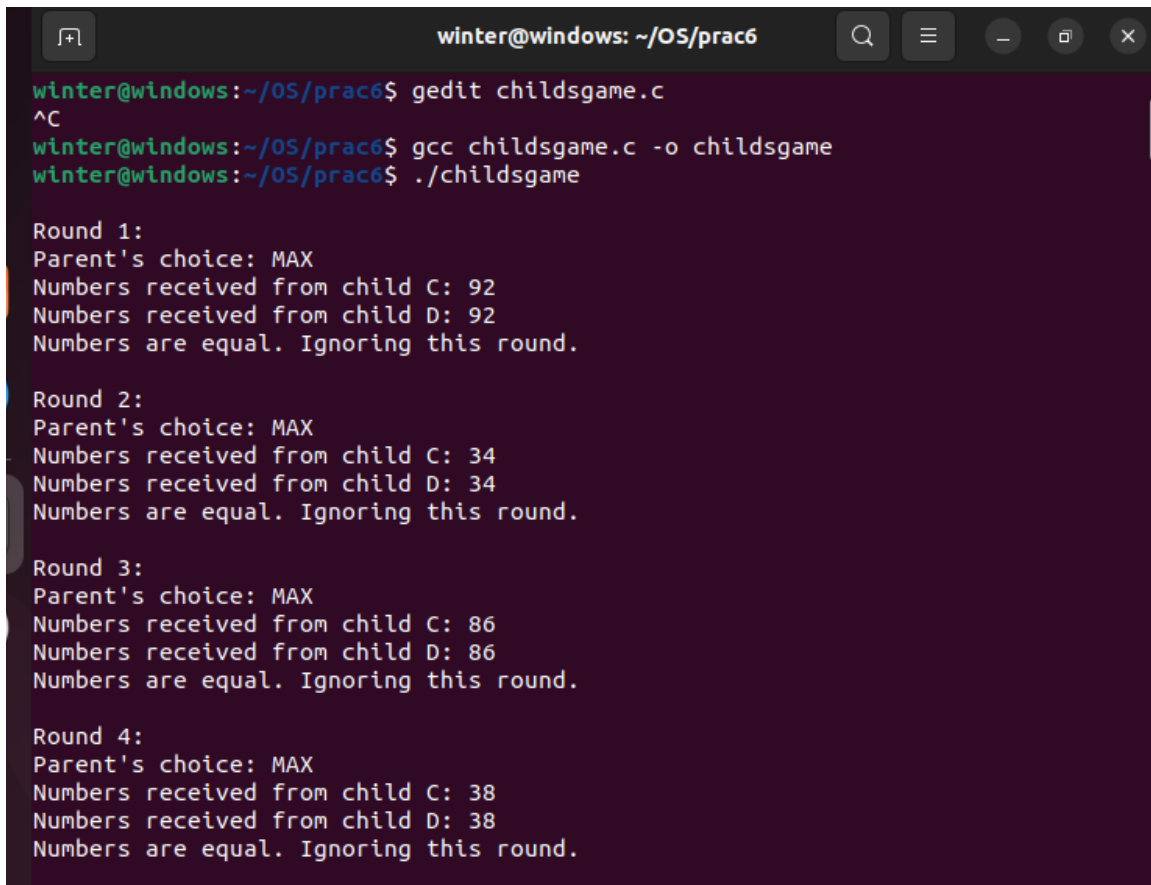
        playGame(pipe1, pipe2);

        int status;
        waitpid(c_pid, &status, 0);
        waitpid(d_pid, &status, 0);
    }
}

return 0;
}

```

OUTPUT -



```

winter@windows: ~/OS/prac6
winter@windows:~/OS/prac6$ gedit childsgame.c
^C
winter@windows:~/OS/prac6$ gcc childsgame.c -o childsgame
winter@windows:~/OS/prac6$ ./childsgame

Round 1:
Parent's choice: MAX
Numbers received from child C: 92
Numbers received from child D: 92
Numbers are equal. Ignoring this round.

Round 2:
Parent's choice: MAX
Numbers received from child C: 34
Numbers received from child D: 34
Numbers are equal. Ignoring this round.

Round 3:
Parent's choice: MAX
Numbers received from child C: 86
Numbers received from child D: 86
Numbers are equal. Ignoring this round.

Round 4:
Parent's choice: MAX
Numbers received from child C: 38
Numbers received from child D: 38
Numbers are equal. Ignoring this round.

```

RESULT :

Linux C programs for Inter-Process Communication using pipes has been implemented.