OPERATING SYSTEMS LAB

PRACTICAL 6(PART 2)

NAME: VEDANT BHUTADA

ROLL: 69

BATCH: A4

<u>Aim</u>: Develop an application for Inter-Process Communication using pipes.

Program-1:

```
#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); //Creating a pipe fd[0] is for reading and 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);
```

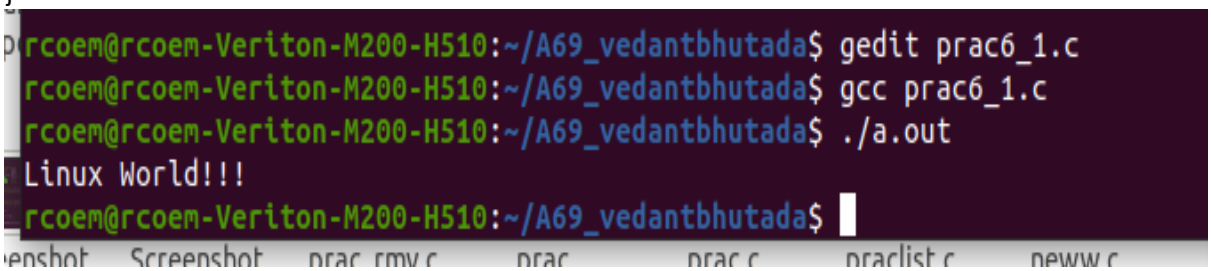
```
    }

    //Reading the message from the pipe

    result=read (fd[0],recvd_msg,MSG_LEN);

    if (result < 0) {
       perror("read");
       exit(3);
    }

    printf("%s\n",recvd_msg);
    return 0;
}
```



Program 3: Write a C program in Linux to generate Fibonacci series in child process and pass numbers to parent process using pipe and parent process should separate the odd and even numbers.

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

int fibonacci(int n) {
   if (n == 0)
      return 0;
   else if (n == 1)
      return 1;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}

void generateFibonacci(int n, int pipefd[2]) {
   int fib;
   for (int i = 0; i < n; i++) {
      fib = fibonacci(i);
      write(pipefd[1], &fib, sizeof(int));
   }
   close(pipefd[1]);
}
```

```
void separateOddEven(int pipefd[2]) {
    int number;
    int fibonacciNumbers[10];
    int count = 0;

    while (read(pipefd[0], &number, sizeof(int)) > 0) {
        fibonacciNumbers[count] = number;
        count++;
    }
    close(pipefd[0]);

    printf("Even numbers: ");
    for (int i = 0; i < count; i++) {
        if (fibonacciNumbers[i] % 2 == 0) {
            printf("%d ", fibonacciNumbers[i]);
        }
    }
    printf("\n");

    printf("Odd numbers: ");
    for (int i = 0; i < count; i++) {
        if (fibonacciNumbers[i] % 2 != 0) {
            printf("%d ", fibonacciNumbers[i]);
        }
    }
    printf("\n");
}


int main() {
    int pipefd[2];
    if (pipe(pipefd) == -1) {
        perror("Pipe creation failed");
        return 1;
    }

    if (fork() == 0) {
        // Child process
        close(pipefd[0]);
        generateFibonacci(10, pipefd);
    } else {
        // Parent process
        close(pipefd[1]);
        separateOddEven(pipefd);
    }

    return 0;
```

}

```
rcoem@rcoem-Veriton-M200-H310:~/A69_vedantbhutada$ gcc fibo.c
rcoem@rcoem-Veriton-M200-H310:~/A69_vedantbhutada$ ./a.out
Even numbers: 0 2 8 34
Odd numbers: 1 1 3 5 13 21
rcoem@rcoem-Veriton-M200-H310:~/A69_vedantbhutada$
```

Program 4:

CREATION OF A ONEWAY PIPE BETWEEN TWO PROCESS */

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
 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);
 }
 }
```

```
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ ./a.out

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

 size of data12
 data received from child throughpipe:hello world
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$
```

Program 5:

/*CREATION OF A TWOWAY PIPE BETWEEN TWO PROCESS*/

PROGRAM

```
#include<stdio.h>
 #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");
}
 }
```

```
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ ./a.out

 readfds=3 5

 writefds=4 6

 parent process receiving data

 CHILD PROCESS SENDING DATA

 data received from child through pipe:where is GEC

 reply send
 reply from parent: in gudlavalleru
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$
```

Program-6:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
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());
  }
}}
```

```
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ ./a.out
         PROCESS 1.ENTER THE STRINGvedant

         PROCESS 2:it splits the given string
p4 5003 p2 5002
         PROCESS 4:sub string    ved    no of char=3
p5 5004 p2 5002
         PROCESS 5:sub string    ant    no of char=3
p2 5002 p1 5001
p1 5001 kernal 3199
p3 5005 p1 5001
         PROCESS 3: the given string is not palindrome
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$
```

Program- 7:

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 .

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

#define BUFSIZE 20
#define WINNING_CONDITION 10

static int stop = 0;

// Custom signal handler to handle the termination of game
void end_process(int sig) {
  if (sig == SIGUSR1) {
    stop = 1;
  }
  else if (sig == SIGUSR2) {
    stop = 2;
  }
}

int main() {
  pid_t pid1, pid2;

  // Create the pipe descriptors
  int fd1[2], fd2[2];
  pipe(fd1);
  pipe(fd2);

  // Assign the custom signal handler to SIGUSR1 signal
  signal(SIGUSR1, end_process);
  signal(SIGUSR2, end_process);

  if ((pid1 = fork()) == 0) {
    // Child 1
    srand((unsigned int)time(NULL) ^ getpid());
    close(fd1[0]);  // Close the read end as the child will not read

    // Generate a random number and place it on the pipe
    while(!stop) {
      int n = rand() % 100;
      char line[BUFSIZE];
      sprintf(line, "%d", n);
      write(fd1[1], line, BUFSIZE);
    }

    // Handle the exit of the child process by displaying proper message
```

```
  if (stop == 1) {
    printf("C: I am the winner\n");
  } else if (stop == 2) {
    printf("C: D is the winner\n");
  }
  printf("Exiting from child C\n\n");

} else {
  if ((pid2 = fork()) == 0) {
    // Child 2
    srand((unsigned int)time(NULL) ^ getpid());
    close(fd2[0]);  // Close the read end as the child will not read

    // Generate a random number and place it on the pipe
    while(!stop) {
      int n = rand() % 100;
      char line[BUFSIZE];
      sprintf(line, "%d", n);
      write(fd2[1], line, BUFSIZE);
    }

    // Handle the exit of the child process by displaying proper message
    if (stop == 1) {
      printf("D: C is the winner\n");
    } else if (stop == 2) {
      printf("D: I am the winner\n");
    }
    printf("Exiting from child D\n\n");

  } else {
    // Parent
    close(fd1[1]);
    close(fd2[1]);  // Close the write ends as the parent will not read

    srand((unsigned int)time(NULL) ^ getpid());
    int round = 1;
    int score1 = 0, score2 = 0;
    char line[BUFSIZE];
    char winner;
    int num1, num2, choice;

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

      choice = rand() % 2;  // P chooses a flag: 0 for MAX and 1 for MIN

      // Read and print the first integer from the pipe
      read(fd1[0], line, BUFSIZE);
      sscanf(line, "%d", &num1);
      printf("Integer received from C: %d\n", num1);
```

```
      // Read and print the second integer from the pipe
      read(fd2[0], line, BUFSIZE);
      sscanf(line, "%d", &num2);
      printf("Integer received from D: %d\n", num2);

      // Choose the winner for the round and add to its point
      if (choice) {
        printf("P's choice of flag: MIN\n");
      } else {
        printf("P's choice of flag: MAX\n");
      }
      if (num1 == num2)
        printf("This round is a tie!\n");
      else if ((num1 < num2 && choice) || (num1 > num2 && !choice)) {
        printf("C gets the point\n");
        score1++;
      } else {
        printf("D gets the point\n");
        score2++;
      }
      printf("Updated scores: C = %d, D = %d\n\n", score1, score2);

      // Break when score of any child becomes 10
      if (score1 == WINNING_CONDITION || score2 == WINNING_CONDITION)
        break;
    }

    // Send appropriate singal to the children to let them know who the winner was
    if (score1 > score2) {
      kill(pid1, SIGUSR1);
      kill(pid2, SIGUSR1);
    } else {
      kill(pid1, SIGUSR2);
      kill(pid2, SIGUSR2);
    }

    // Wait for the children to terminate and then terminate the parent
    waitpid(pid1, &num1, 0);
    waitpid(pid2, &num2, 0);
    printf("Exiting from parent P\n");
  }
 }
 return 0;
}
```

```
rcoem@rcoem-Veriton-M200-H510:~$ mkdir A69_vedantbhutada
rcoem@rcoem-Veriton-M200-H510:~$ cd A69_vedantbhutada
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ gedit game.c
^C
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ gcc game.c
game.c: In function ‘main’:
game.c:137:7: warning: implicit declaration of function ‘waitpid’ [-Wimplicit-function-declaration]
  137 |       waitpid(pid1, &num1, 0);
      |       ^~~~~~~
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$ ./a.out
Round number: 1
Integer received from C: 90
Integer received from D: 50
P's choice of flag: MAX
C gets the point
Updated scores: C = 1, D = 0

Round number: 2
Integer received from C: 26
Integer received from D: 34
P's choice of flag: MAX
D gets the point
Updated scores: C = 1, D = 1

Round number: 3
Integer received from C: 79
Integer received from D: 59
P's choice of flag: MAX
C gets the point
Updated scores: C = 2, D = 1

Round number: 4
Integer received from C: 14
Integer received from D: 0
P's choice of flag: MIN
D gets the point
Updated scores: C = 2, D = 2

Round number: 5
Integer received from C: 45
Integer received from D: 61
P's choice of flag: MAX
D gets the point
Updated scores: C = 2, D = 3
```

```
Round number: 12
Integer received from C: 3
Integer received from D: 90
P's choice of flag: MAX
D gets the point
Updated scores: C = 4, D = 8

Round number: 13
Integer received from C: 20
Integer received from D: 70
P's choice of flag: MIN
C gets the point
Updated scores: C = 5, D = 8

Round number: 14
Integer received from C: 5
Integer received from D: 50
P's choice of flag: MIN
C gets the point
Updated scores: C = 6, D = 8

Round number: 15
Integer received from C: 1
Integer received from D: 37
P's choice of flag: MAX
D gets the point
Updated scores: C = 6, D = 9

Round number: 16
Integer received from C: 2
Integer received from D: 4
P's choice of flag: MAX
D gets the point
Updated scores: C = 6, D = 10

C: D is the winner
D: I am the winner
Exiting from child C
Exiting from child D


Exiting from parent P
rcoem@rcoem-Veriton-M200-H510:~/A69_vedantbhutada$
```

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