NAME: AMIT KUMAR SHAW
ENROLL: 510819012
ASSIGNMENT: 05

1. Write a C program where two processes that have no parent-child relationship, will communicate between themselves using PIPE. Make sure both the process may send and receive data from others.

p1_make_pipe.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
        int key;
        key=mkfifo("pipe1", 0777);
        printf("Pipe created\n");
}
```

p1_sender.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
        int fd=open("pipe1", O_NONBLOCK, O_RDWR);
        write(fd, "Sending data from process1", 27);
        printf("Data sent from process1 with pid: %d\n", getpid());

        char buff[100];
        read(fd, buff, 100);
        printf("Data received from process2 is: %s\n", buff);
}
```

p1_receiver.c:

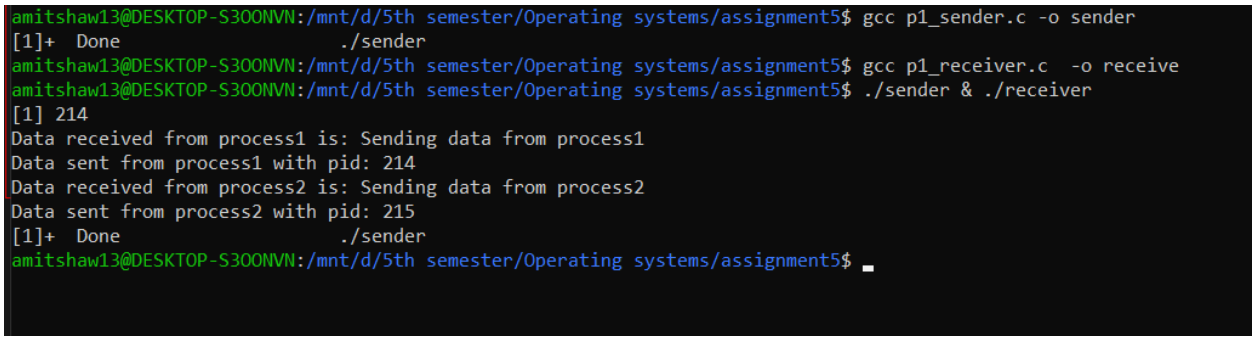
```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

int main(){
        int fd=open("pipe1", O_NONBLOCK, O_RDWR);
        char buff[100];
        read(fd, buff, 100);
        printf("Data received from process2 is: %s\n", buff);

        write(fd, "Sending data from process2", 27);
        printf("Data sent from process2 with pid: %d\n", getpid());
}
```

OUTPUT:



2. Write a C program to implement the banker's algorithm that is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

p2.c:

```
#include<stdio.h>
#include<stdbool.h>
#include<stdlib.h>
const int P = 5;
const int R = 3;
void calc(int arr[P][R], int m[P][R],
        int allot[P][R]){
        for (int i = 0 ; i < P ; i++)
        for (int j = 0 ; j < R ; j++)
        arr[i][j] = m[i][j] - allot[i][j];
}
bool safe(int processes[], int avail[], int max[][R],int alloted[][R]){
        int arr[P][R];
        calc(arr, max, alloted);
```

```
        bool finish[3] = {0,0,0};
        int safeSeq[P];
        int work[R];
        for (int i = 0; i < R ; i++)
        work[i] = avail[i];
        int count = 0;
        while (count < P){
                bool found = false;
                for (int p = 0; p < P; p++){
                        if (finish[p] == 0){
                                int j;
                                for (j = 0; j < R; j++)
                                        if (arr[p][j] > work[j])
                                        break;
                                if (j == R){
                                        for (int k = 0 ; k < R ; k++)
                                                work[k] += alloted[p][k];
                                        safeSeq[count++] = p;
                                        finish[p] = 1;
                                        found = true;
                                }
                        }
                }
                if (found == false){
                        printf("NO Safe State\n");
                        return false;
                }
        }
        printf("Safe State Detected.\n");
        for (int i = 0; i < 5 ; i++)
                printf("%d, ",safeSeq[i]);
        exit(0);
}
int main(){
        int processes[5] = {0, 1, 2, 3, 4};
        int avail[3] = {3, 3, 2};
        bool a;
        int max[5][3] = {{7, 5, 3},
        {3, 2, 2},
        {2, 0, 2},
        {7, 2, 2},
        {4, 3, 3}};
        int alloted[5][3] = {{0, 1, 0},
        {2, 0, 0},
```

```
        {1, 0, 3},
        {2, 1, 1},
        {0, 0, 2}};
        a = safe(processes, avail, max, alloted);
        return 0;
}
```

OUTPUT: