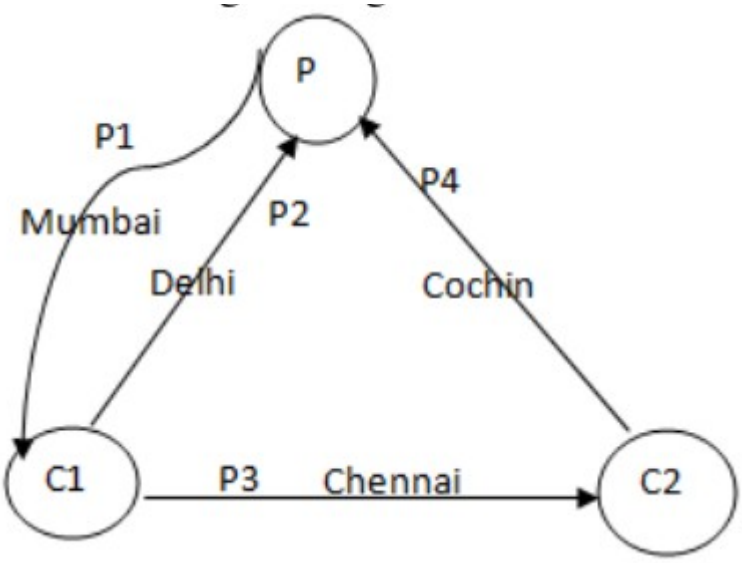
# LAB 10 – AUP Lab

Akash Sarda 111403003
Aditya Malu 111403023
Jassim Rahman 111403019

---

1. A pipe setup is given below that involves three processes. P is the parent process, and C1 and C2 are child processes, spawned from P. The pipes are named p1, p2, p3, and p4. Write a program that establishes the necessary pipe connections, setups, and carries out the reading/writing of the text in the indicated directions.




**Demo :**

```
aditya : que1 $ gcc 1.c
aditya : que1 $ ./a.out
In Child 1: Pipe 1 Read: Mumbai
In Parent: Pipe 2 Read: Delhi
In Child 2: Pipe 3 Read: Chennai
In Parent: Pipe 4 Read: Cochin
aditya : que1 $ _
```

**Code:**

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

int main(){
     pid_t pid1, pid2, pid;
```

```
     int p1[2], p2[2], p3[2], p4[2];
     char buf1[16], buf2[16], buf3[16], buf4[16];
     char cities[4][16] = {"Mumbai", "Delhi", "Cochin",
"Chennai"};
     if(pipe(p1) == -1 || pipe(p2) == -1 || pipe(p3) == -1 ||
pipe(p4) == -1){
          perror("Pipe Error");
          exit(1);
     }

     if((pid1 = fork()) < 0){  // Forking Child 1
          perror("Failed to fork process \n");
          exit(1);
     }
     else if ( pid1 > 0 ){
          // Parent Here //
          close(p1[0]);  // Writing in Pipe 1
          write(p1[1], cities[0], 16); //Writing "Mumbai" on Pipe
1
          close(p1[1]);

          close(p2[1]); // Reading in Pipe 2
          if(read(p2[0], buf3, 16) == -1){
               perror("Pipe 2, Parent: Error:");
          } else {
               printf("In Parent: Pipe 2 Read: %s\n", buf3);
          }
          close(p2[0]);
          // FORKING CHILD 2 //
          if((pid2 = fork()) < 0){
               perror("Failed to fork process \n");
               exit(1);
          } else if (pid2 == 0) {
               //  CHILD 2 //
               close(p3[1]);
               if(read(p3[0], buf2, 16) == -1){
                    perror("Pipe 3, Child 2: Error:");
               } else {
                    printf("In Child 2: Pipe 3 Read: %s\n", buf2);
               }
               close(p3[0]);

               close(p4[0]);
               write(p4[1], cities[2], 16);  //Writing "Cochin" on
Pipe 4
               close(p4[1]);
               // CHILD 2 DONE //
          } else {
               // PARENT HERE //
               close(p4[1]); // Reading in Pipe 4
```

```
                if(read(p4[0], buf4, 16) == -1){
                    perror("Pipe 4, PArent: Error:");
                } else {
                    printf("In Parent: Pipe 4 Read: %s\n", buf4);
                }
                close(p4[0]);
            }
        }
        else { // CHILD 1 (C1) //
            close(p1[1]);
            if(read(p1[0], buf1, 16) == -1){
                perror("Pipe 1, Child 1: Error:");
            } else {
                printf("In Child 1: Pipe 1 Read: %s\n", buf1);
            }
            close(p1[0]);

            close(p2[0]);
            write(p2[1], cities[1], 16); // Writing "Delhi" on Pipe
2
            close(p2[1]);

            close(p3[0]);
            write(p3[1], cities[3], 16); // Writing "Chennai" on
Pipe 3
            close(p3[1]);
        }
}
```

---

2. Let P1 and P2 be two processes alternatively writing numbers from 1 to 100 to a file. Let P1 write odd numbers and p2, even. Implement the synchronization between the processes using FIFO.

**Demo :**

```
aditya : que2 $ gcc p1.c -o p1
aditya : que2 $ gcc p2.c -o p2
aditya : que2 $ ./p1 & ./p2
[1] 5242
P1: 1   P2: 2   P1: 3   P2: 4   P1: 5   P2: 6   P1: 7   P2: 8   P1: 9   P2: 10  P1: 11  P2: 12  P1: 13  P2: 14  P1: 15  P2: 16  P1: 17  P2: 18  P1: 19  P2: 20  P1: 21  P2: 22  P1: 23  P2: 24  P1
: 25    P2: 26  P1: 27  P2: 28  P1: 29  P2: 30  P1: 31  P2: 32  P1: 33  P2: 34  P1: 35  P2: 36  P1: 37  P2: 38  P1: 39  P2: 40  P1: 41  P2: 42  P1: 43  P2: 44  P1: 45  P2: 46  P1: 47  P2: 48  P1
: 49    P2: 50  P1: 51  P2: 52  P1: 53  P2: 54  P1: 55  P2: 56  P1: 57  P2: 58  P1: 59  P2: 60  P1: 61  P2: 62  P1: 63  P2: 64  P1: 65  P2: 66  P1: 67  P2: 68  P1: 69  P2: 70  P1: 71  P2: 72  P1
: 73    P2: 74  P1: 75  P2: 76  P1: 77  P2: 78  P1: 79  P2: 80  P1: 81  P2: 82  P1: 83  P2: 84  P1: 85  P2: 86  P1: 87  P2: 88  P1: 89  P2: 90  P1: 91  P2: 92  P1: 93  P2: 94  P1: 95  P2: 96  P1
: 97    P2: 98  P1: 99  P2: 100 [1]+  Done                    ./p1
aditya : que2 $
aditya : que2 $ _
```

**Code :**

p1.c

---

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

```
int main()
{
     int fd, num = 1;
     char *fifofile = "fifofile";
     mkfifo(fifofile, 0666);
     char buf1[4], buf2[4];
     while (1)
     {
          fd = open(fifofile, O_WRONLY);
          snprintf (buf2, sizeof(buf2), "%d",num);
          write(fd, buf2, sizeof(buf2));
          close(fd);

          fd = open(fifofile, O_RDONLY);
          read(fd, buf1, sizeof(buf1));
          printf("P2: %s\t", buf1);
          fflush(NULL);
          close(fd);
          num = num + 2;
          if(num > 100){
               break;
          }
     }
     return 0;
}
```

p2.c

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

int main()
{
     int fd1, num=2;
     char *fifofile = "fifofile";
     mkfifo(fifofile, 0666);
     char buf1[4], buf2[4];
     while (1)
     {
          fd1 = open(fifofile,O_RDONLY);
          read(fd1, buf1, sizeof(buf1));
          printf("P1: %s\t", buf1);
          fflush(NULL);
          close(fd1);

          fd1 = open(fifofile,O_WRONLY);
          snprintf (buf2, sizeof(buf2), "%d",num);
```

```
          write(fd1, buf2, sizeof(buf2));
          close(fd1);

          num = num + 2;
          if(num > 100){
               break;
          }
     }
     return 0;
}
```

---

3. Implement a producer-consumer setup using shared memory and semaphore. Ensure that data doesn't get over-written by the producer before the consumer reads and displays on the screen. Also ensure that the consumer doesn't read the same data twice.

**Code :**

producer.c

---

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<sys/sem.h>
int main()
{
     key_t key = 5678, skey = IPC_PRIVATE;
     int i;
     int shmid = shmget(key,10 , IPC_CREAT | 0666), semid =
semget(skey, 1, IPC_CREAT | IPC_EXCL | 0666);
     char *shm = shmat(shmid, NULL, 0), *ptr;
     struct sembuf sb;

     ptr = shm;
     if(semid >= 0){

          union semun{
               int val;
               struct semid_dss *buf;
               short *array;
          }arg;

          arg.val = 1;
          semctl(semid, 0, SETVAL, arg);
          i = 0;
          while(1){
               if(i == 9){
                    ptr = shm;
```

```
                    i = 0;
               }
               sb.sem_num = 0;
               sb.sem_op = -1;
               semop(semid, &sb, 1);
               if(*ptr == 'p'){
                    // buffer is full.
                    printf("Buffer is full it seems.\n");
                    sb.sem_num = 0;
                    sb.sem_op = 1;
                    semop(semid, &sb, 1);
                    continue;
               }else{
                    printf("producing.\n");
                    *ptr = 'p';
                    ptr++;
                    i = (i + 1) % 10;
               }
               sb.sem_num = 0;
               sb.sem_op = 1;
               semop(semid, &sb, 1);
          }
     }
     else{
          printf("Semaphore error.\n");
     }
     shm[10] = NULL;
     exit(0);
}
```

consumer.c

```
---
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<sys/sem.h>
int main()
{
     key_t key = 5678, skey = IPC_PRIVATE;
     int i;
     int shmid = shmget(key,10 , IPC_CREAT | 0666), semid =
semget(skey, 1, IPC_CREAT | IPC_EXCL | 0666);
     char *shm = shmat(shmid, NULL, 0), *ptr;
     struct sembuf sb;
     ptr = shm;
     if(semid >= 0){

          union semun{
```

```
            int val;
            struct semid_dss *buf;
            short *array;
        }arg;

        arg.val = 1;
        semctl(semid, 0, SETVAL, arg);
        i = 0;
        while(1){
            if(i == 9){
                i = 0;
                ptr = shm;
            }

            sb.sem_num = 0;
            sb.sem_op = -1;
            semop(semid, &sb, 1);
            if(*ptr == 'c'){
                // buffer is empty.
                printf("Buffer is empty\n");
                sb.sem_num = 0;
                sb.sem_op = 1;
                semop(semid, &sb, 1);
                continue;
            }else{
                // get the semaphore
                printf("Consuming\n");
                *ptr = 'c';
                ptr++;
                i = (i + 1) % 10;
                // release the semaphore
            }

            sb.sem_num = 0;
            sb.sem_op = 1;
            semop(semid, &sb, 1);

        }

    }
    shm[10] = NULL;
    exit(0);
}
```