190905104
Parth Shukla
Lab 7

**Message Queue, Shared Memory**

**1) Process A wants to send a number to Process B. Once received, Process B has to check whether the number is palindrome or not. Write a C program to implement this interprocess communication using a message queue.**

**// Sender**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<limits.h>
#include<fcntl.h>
#include<sys/msg.h>
#include<sys/stat.h>
#include<string.h>
#include<sys/msg.h>
#include<sys/ipc.h>
#include<errno.h>
#define MAX_TEXT 512

struct my_msg_st
{
   long int my_msg_type;
   int msg;
};

int main(int argc, char const *argv[])
{
   int running=1;
   struct my_msg_st some_data;
   int msgid;
   int num;
   msgid=msgget((key_t)1234,0666|IPC_CREAT);

   if(msgid==-1)
   {
      fprintf(stderr, "msgget failed with error%d\n",errno );
      exit(EXIT_FAILURE);
   }

   printf("Enter -1 to quit\n");

   while(running)
   {
      printf("Enter a number\n");
      scanf("%d",&num);
      some_data.my_msg_type=1;
      some_data.msg=num;
```

```
        if (msgsnd(msgid,(void*)&some_data,MAX_TEXT,0)==-1){
            fprintf(stderr, "msgsnd failed\n" );
            exit(EXIT_FAILURE);
        }
        if(num==-1)
            running=0;
    }

    exit(EXIT_SUCCESS);
}
```

**// Receiver**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<limits.h>
#include<fcntl.h>
#include<sys/msg.h>
#include<sys/stat.h>
#include<string.h>
#include<sys/msg.h>
#include<sys/ipc.h>
#include<errno.h>
#define MAX_TEXT 512

struct my_msg_st
{
    long int my_msg_type;
    int msg;
};

int reverse(int x)
{
    int y = 0;
    while(x > 0)
    {
        y *= 10;
        y += x % 10;
        x /= 10;
    }
    return y;
}

int main(int argc, char const *argv[])
{
    int running=1;
    struct my_msg_st some_data;
    long int msg_to_receive=0;
    int msgid;
    int num;
```

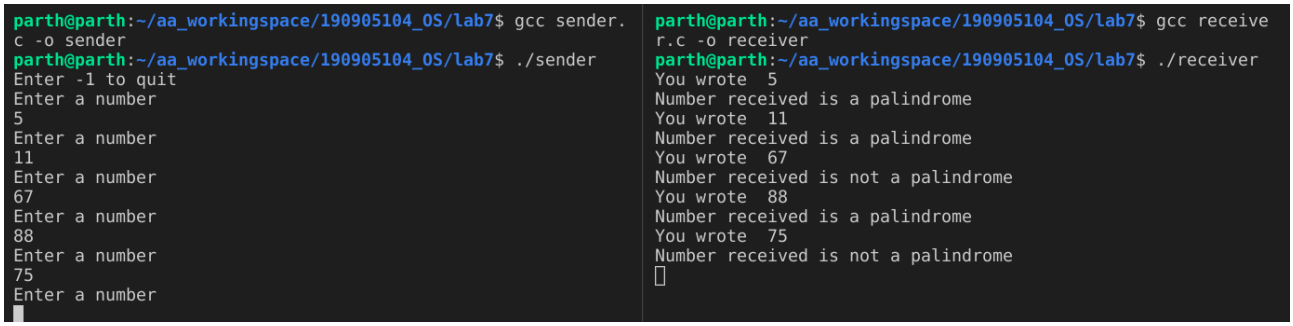
```
    msgid=msgget((key_t)1234,0666|IPC_CREAT);

    if(msgid==-1)
    {
        fprintf(stderr, "msgget failed with error%d\n",errno );
        exit(EXIT_FAILURE);
    }

    while(running)
    {
        if (msgrcv(msgid,(void*)&some_data,BUFSIZ,msg_to_receive,0)==-1)
        {
            fprintf(stderr, "msgrc failedwith error %d\n",errno );
            exit(EXIT_FAILURE);
        }

        printf("You wrote  %d\n",some_data.msg);

        if(some_data.msg == reverse(some_data.msg))
            printf("Number received is a palindrome\n");
        else
            printf("Number received is not a palindrome\n");
        if(some_data.msg==-1)
            running=0;
    }
    if(msgctl(msgid,IPC_RMID,0)==-1){
        fprintf(stderr, "msgctl(IPC_RMID) failed\n");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```




**2) Implement a parent process, which sends an English alphabet to a child process using shared memory. The child process responds with the next English alphabet to the parent. The parent displays the reply from the Child.**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

```
/*
Status codes
0 -> nothing written yet by parent process
1 -> alphabet written by parent process
2 -> answer written by child process
-1 -> exit
*/

struct shared_str
{
   int status;
   char alphabet;
};

int main(int argc, char const *argv[])
{
   int shmid = shmget((key_t)1234,sizeof(struct shared_str),0666|IPC_CREAT);
   pid_t pid = fork();

   if(pid < 0)
   {
      printf("Error in fork()\n");
      exit(-1);
   }

   else if(pid == 0)
   { //child process
      struct shared_str* shared_mem = shmat(shmid,(void*)0,0);

      if(shared_mem == (void*)-1)
      {
         printf("shmat() failed\n");
         exit(-1);
      }

      printf("Memory attached at %p for child process\n",shared_mem);

      while(1)
      {
         if(shared_mem->status < 0)
         {
            // printf("Exit code received %d\n",shared_mem->status);
            if(shmdt(shared_mem) == -1)
            {
               printf("shmdt failed\n");
               exit(-1);
            }

            break;
         }

         if(shared_mem->status == 1)
```

```
        {
            char c = shared_mem->alphabet;
            printf("\n");

            if((int)c >= 65 && (int)c <= 90)
            { //uppercase
                c = ((c - 'A' + 1)%26) + 'A';
            }

            else if((int)c >= 97 && (int)c <= 122)
            { //lowecase
                c = ((c - 'a' + 1)%26) + 'a';
            }

            else
            {
                printf("Non-alphabetic character received\n");
                //do nothing
            }
            shared_mem->alphabet = c; //write to shared memory
            shared_mem->status = 2;
        }
    }
}

else
{ //parent process
    sleep(1);
    struct shared_str* shared_mem = shmat(shmid,(void*)0,0);

    if(shared_mem == (void*)-1)
    {
        printf("shmat() failed\n");
        exit(-1);
    }

    printf("Memory attached at %p for parent process\n",shared_mem);
    shared_mem->status = 0;

    while(1)
    {
        if(shared_mem->status == 1)
        {
            // printf("Waiting for child process\n");
            continue;
        }

        if(shared_mem->status == 2)
        {
            printf("%c\n",shared_mem->alphabet);
        }
```

```
        shared_mem->status = 0;
        char c,nl;
        printf("Enter an alphabet (0 to exit) : \n");
        scanf("%c",&c);
        scanf("%c",&nl);

        if(c == '0')
        {
           shared_mem->status = -1;
           printf("Exiting...\n");

           if(shmdt(shared_mem) == -1)
           {
              printf("shmdt failed\n");
              exit(-1);
           }

           if(shmctl(shmid,IPC_RMID,0) == -1)
           {
              printf("shmctl failed\n");
              exit(-1);
           }
           break;
        }

        shared_mem->alphabet = c;
        shared_mem->status = 1;
     }
   }
   return 0;
}
```

```
parth@parth:~/aa_workingspace/190905104_OS/lab7$ ./q2
Memory attached at 0x7fce992a8000 for child process
Memory attached at 0x7fce992a8000 for parent process
Enter an alphabet (0 to exit) :
f

g
Enter an alphabet (0 to exit) :
a

b
Enter an alphabet (0 to exit) :
z

a
Enter an alphabet (0 to exit) :
t

u
Enter an alphabet (0 to exit) :
0
Exiting...
```