**Lab 6 :IPC – 2: Message Queue, Shared Memory**

**Lab Exercises:**

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");

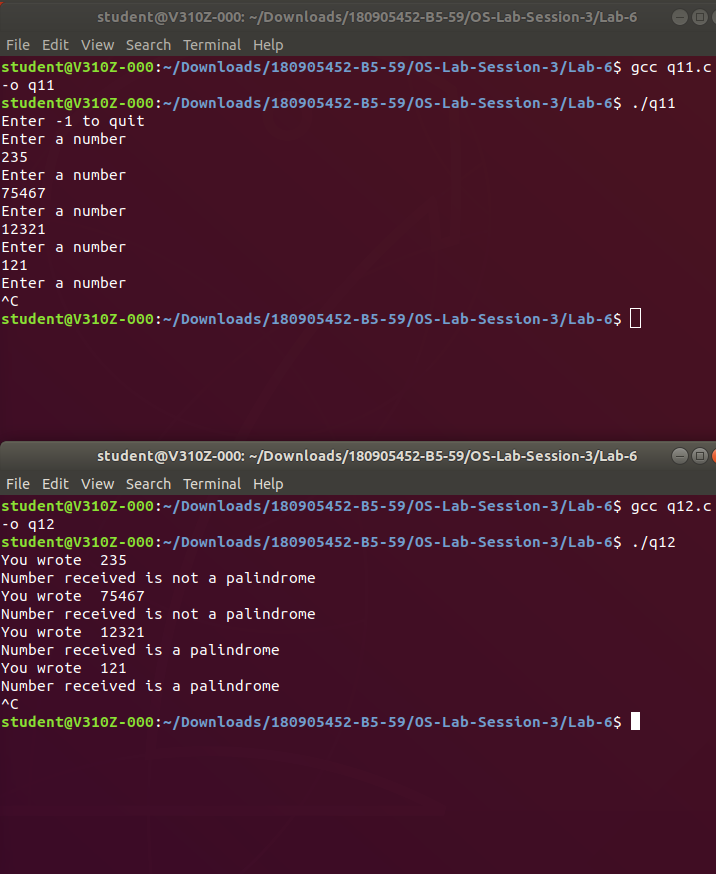
exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

**Output**



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;

}

**Output**

