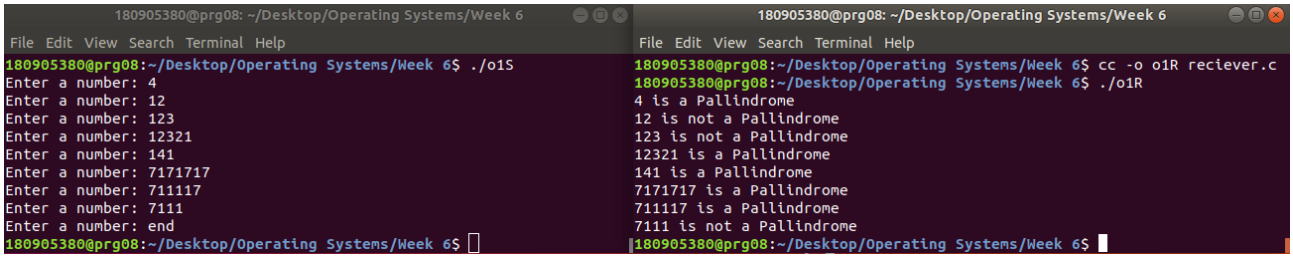
Q1. Process A wants to send a number to Process B. Once recieved, Process B has to check whether the number is palindrome or not. Write a C program to implement this interprocess communication using a message queue.

```
//sender.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT 512

struct my_msg_st
{
	long int my_msg_type;
	char some_text[BUFSIZ];
};

int main()
{
	int running = 1;
	int msgid;
	struct my_msg_st some_data;
	char buffer[BUFSIZ];
	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)
	{
		printf("Enter a number:\t");
		fgets(buffer,BUFSIZ,stdin);
		some_data.my_msg_type = 1;
		strcpy(some_data.some_text, buffer);
		if (msgsnd(msgid,(void *)&some_data,MAX_TEXT,0) == -1)
		{
			fprintf(stderr, "msgsnd failed\n");
			exit(EXIT_FAILURE);
		}
		if (strncmp(buffer, "end",3) == 0)
		{
			running = 0;
		}
	}
	exit(EXIT_SUCCESS);
}

//reciever.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
struct my_msg_st
{
	long int my_msg_type;
	char some_text[BUFSIZ];
};

int main()
{
	int running = 1;
	int msgid;
	struct my_msg_st some_data;
	long int msg_to_recieve = 0;
	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_recieve,0) == -1)
		{
			fprintf(stderr,"msgrcv failed with error: %d\n", errno);
			exit(EXIT_FAILURE);
		}
		char rev[BUFSIZ];
		int i;
		for (i = 0; some_data.some_text[i] != '\n'; i++);
		for (int j = 0; j < i; j++)
			rev[j] = some_data.some_text[i-j-1];
		rev[i] = '\0';
		some_data.some_text[i] = '\0';
		if (strncmp(some_data.some_text,"end",3) == 0)
			running = 0;
		else if (strcmp(some_data.some_text,rev) == 0)
			printf("%s is a Pallindrome\n",some_data.some_text);
		else
			printf("%s is not a Pallindrome\n",some_data.some_text);
	}
	if (msgctl(msgid,IPC_RMID,0) == -1)
	{
		fprintf(stderr, "msgctl(IPC_RMID) failed\n");
		exit(EXIT_FAILURE);
	}
	exit(EXIT_SUCCESS);
}
```



Q2. 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 <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

struct shared_use
{
	char c;
	char next_char;
};

int main()
{
	void *shared_memory = (void *)0;
	int shmid;
	struct shared_use *stuff;
	char buffer;
	pid_t p;
	p = fork();
	if (p == -1)
	{
		fprintf(stderr,"fork failed!!\n");
		exit(EXIT_FAILURE);
	}
	else if (p == 0)
	{
		shmid = shmget((key_t)1234,sizeof(struct shared_use),0666|IPC_CREAT);
		if(shmid==-1)
		{
			fprintf(stderr,"shmget failed!!\n");
			exit(EXIT_FAILURE);
		}
		shared_memory = shmat(shmid,(void *)0,0);
		if(shared_memory == (void *)-1)
		{
			fprintf(stderr, "shmat failed\n");
			exit(EXIT_FAILURE);
		}
		stuff = (struct shared_use *)shared_memory;
		sleep(10);
		printf("current char: %c\n",stuff->c);
		stuff->c++;
		if(shmdt(shared_memory)==-1)
		{
			fprintf(stderr, "shmdt failed\n");
			exit(EXIT_FAILURE);
		}
		if(shmctl(shmid,IPC_RMID,0)==-1)
		{
			fprintf(stderr, "shmctl(IP_RMID) failed\n");
			exit(EXIT_FAILURE);
		}
		exit(EXIT_SUCCESS);
```

```
	}
	else
	{
		shmid = shmget((key_t)1234,sizeof(struct shared_use),0666|IPC_CREAT);
		if(shmid==-1)
		{
			fprintf(stderr,"shmget failed!!\n");
			exit(EXIT_FAILURE);
		}
		shared_memory = shmat(shmid,(void *)0,0);
		if(shared_memory == (void *)-1)
		{
			fprintf(stderr, "shmat failed\n");
			exit(EXIT_FAILURE);
		}
		stuff = (struct shared_use *)shared_memory;
		char ch;
		printf("Enter a character: \n");
		scanf("%c",&ch);
		stuff->c = ch;
		printf("current char: %c\n",stuff->c);
		printf("Waiting for child process to change...\n");
		wait(NULL);
		printf("new char: %c\n",stuff->c);
		if(shmdt(shared_memory)==-1)
		{
			fprintf(stderr, "shmdt failed\n");
			exit(EXIT_FAILURE);
		}
		exit(EXIT_SUCCESS);
	}
}
```