answers/Code_Q3_4/Q4/posix_mq.c

```c
/*
 * Author: Sam Siewart for posix_mq.c code in Exercise3/Posix_MQ_loop
 * Modified by: Shashank and Parth
 * References:
 * 1. Sam Siewert - 10/14/97 posix_mq.c - vxWorks code
 * 2. posix_mq.c code in Exercise3/Posix_MQ_loop used as the base
 */

#define _GNU_SOURCE
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <mqueue.h>
#include <unistd.h>

// On Linux the file systems slash is needed
#define SNDRCV_MQ "/send_receive_mq"

#define MAX_MSG_SIZE 128
#define ERROR (-1)

#define NUM_CPUS (1)

pthread_t th_receive, th_send; // create threads
pthread_attr_t attr_receive, attr_send;
struct sched_param param_receive, param_send;

struct mq_attr mq_attr;
mqd_t mymq;

static char canned_msg[] = "This is a test, and only a test, in the event of real emergency, you would be instructed...."; // Message to be sent

/* receives pointer to heap, reads it, and deallocate heap memory */
void *receiver(void *arg)
{
  char buffer[MAX_MSG_SIZE];
  int prio;
  int nbytes;

  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);

  printf("receiver - thread entry\n");

  while(1)
  {
    printf("receiver - awaiting message\n");

    if((nbytes = mq_receive(mymq, buffer, MAX_MSG_SIZE, &prio)) == ERROR)
    {
      perror("mq_receive");
```

```
    }
    else
    {
      buffer[nbytes ] = '\0';
      printf("receiver - msg %s received with priority = %d, nbytes = %d\n", buffer,
prio, nbytes);
    }
  }
}

/*send the data in the buffer*/
void *sender(void *arg)
{
  int prio;
  int rc;

  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);

  printf("sender - thread entry\n");

  while(1)
  {
    printf("sender - sending message of size=%d\n", sizeof(canned_msg));

    if((rc = mq_send(mymq, canned_msg, sizeof(canned_msg), 30)) == ERROR)
    {
      perror("mq_send");
    }
    else
    {
      printf("sender - message successfully sent, rc=%d\n", rc);
    }
  }
}

void main(void)
{
  int i=0, rc=0;

  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);
  for(i=0; i < NUM_CPUS; i++)
      CPU_SET(i, &cpuset);

  /* setup common message q attributes */
  mq_attr.mq_maxmsg = 10;
  mq_attr.mq_msgsize = MAX_MSG_SIZE;

  mq_attr.mq_flags = 0;

  mq_unlink(SNDRCV_MQ);  //Unlink if the previous message queue exists

  mymq = mq_open(SNDRCV_MQ, O_CREAT|O_RDWR, S_IRWXU, &mq_attr);
  if(mymq == (mqd_t)ERROR)
  {
```

```
108       perror("mq_open");
109     }
110
111     int rt_max_prio, rt_min_prio;
112     rt_max_prio = sched_get_priority_max(SCHED_FIFO);
113     rt_min_prio = sched_get_priority_min(SCHED_FIFO);
114
115     //initialize  with default atrribute
116     rc = pthread_attr_init(&attr_receive);
117     //specific scheduling for Receiving
118     rc = pthread_attr_setinheritsched(&attr_receive, PTHREAD_EXPLICIT_SCHED);
119     rc = pthread_attr_setschedpolicy(&attr_receive, SCHED_FIFO);
120     rc=pthread_attr_setaffinity_np(&attr_receive, sizeof(cpu_set_t), &cpuset);
121     param_receive.sched_priority = rt_min_prio;
122     pthread_attr_setschedparam(&attr_receive, &param_receive);
123
124     //initialize  with default atrribute
125     rc = pthread_attr_init(&attr_send);
126     //specific scheduling for Sending
127     rc = pthread_attr_setinheritsched(&attr_send, PTHREAD_EXPLICIT_SCHED);
128     rc = pthread_attr_setschedpolicy(&attr_send, SCHED_FIFO);
129     rc=pthread_attr_setaffinity_np(&attr_send, sizeof(cpu_set_t), &cpuset);
130     param_send.sched_priority = rt_max_prio;
131     pthread_attr_setschedparam(&attr_send, &param_send);
132
133     if((rc=pthread_create(&th_send, &attr_send, sender, NULL)) == 0)
134     {
135       printf("\n\rSender Thread Created with rc=%d\n\r", rc);
136     }
137     else
138     {
139       perror("\n\rFailed to Make Sender Thread\n\r");
140       printf("rc=%d\n", rc);
141     }
142
143     if((rc=pthread_create(&th_receive, &attr_receive, receiver, NULL)) == 0)
144     {
145       printf("\n\rReceiver Thread Created with rc=%d\n\r", rc);
146     }
147     else
148     {
149       perror("\n\rFailed Making Reciever Thread\n\r");
150       printf("rc=%d\n", rc);
151     }
152
153     printf("pthread join send\n");
154     pthread_join(th_send, NULL);
155
156     printf("pthread join receive\n");
157     pthread_join(th_receive, NULL);
158
159 }
160
```