answers/Code_Q3_4/Q4/heap_mq.c

```c
/*
 * Author: Sam Siewart for heap_mq.c code in Exercise3/Posix_MQ_loop
 * Modified by: Shashank and Parth
 * References:
 * 1. Sam Siewert - 10/14/97 heap_mq.c - vxWorks code
 * 2. heap_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 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;

static char imagebuff[4096];
struct mq_attr mq_attr;
mqd_t mymq;

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

  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);

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

  /* read oldest, highest priority msg from the message queue until empty */
  while(1)
  {
    printf("receiver - awaiting message\n");

    if((nbytes = mq_receive(mymq, buffer, (size_t)(sizeof(void *)+sizeof(int)), &
prio)) == ERROR)
    {
```

```
53        perror("mq_receive");
54      }
55      else
56      {
57        memcpy(&buffptr, &buffer, sizeof(void *));
58        memcpy((void *)&id, &(buffer[sizeof(void *)]), sizeof(int));
59        printf("receiver - ptr msg 0x%p received with priority = %d, length = %d, id =
    %d\n", buffptr, prio, nbytes, id);
60        printf("receiver - Contents of ptr = \n%s\n", (char *)buffptr);
61
62        free(buffptr);
63        printf("receiver - heap space memory freed\n");
64      }
65    }
66  }
67
68  /*send pointer to heap which points to the data in imagebuff*/
69  void *sender(void *arg)
70  {
71    char buffer[sizeof(void *)+sizeof(int)];
72    void *buffptr;
73    int prio;
74    int nbytes;
75    int id = 999;
76
77    cpu_set_t cpuset;
78    CPU_ZERO(&cpuset);
79
80    printf("sender - thread entry\n");
81
82    while(1)
83    {
84      buffptr = (void *)malloc(sizeof(imagebuff));
85      strcpy(buffptr, imagebuff);
86      printf("sender - Message to send = %s\n", (char *)buffptr);
87      printf("sender - Sending message of size=%d\n", sizeof(buffptr));
88
89      memcpy(buffer, &buffptr, sizeof(void *));
90      memcpy(&(buffer[sizeof(void *)]), (void *)&id, sizeof(int));
91
92        if((nbytes = mq_send(mymq, buffer, (size_t)(sizeof(void *)+sizeof(int)), 30)) =
    = ERROR)
93        {
94          perror("mq_send");
95        }
96        else
97        {
98          printf("sender - message ptr 0x%p successfully sent\n", buffptr);
99        }
100   }
101
102 }
103
104 /*Fills imagebuff with ASCII data */
105 void fillbuffer(void)
106 {
107   int i, j;
```

```
108   char pixel = 'A';
109
110   for(i=0;i<4096;i+=64)
111   {
112     pixel = 'A';
113     for(j=i;j<i+64;j++)
114     {
115       imagebuff[j] = (char)pixel++;
116     }
117     imagebuff[j-1] = '\n';
118   }
119   imagebuff[4095] = '\0';
120   imagebuff[63] = '\0';
121 }
122
123
124 void main(void)
125 {
126   int i=0, rc=0;
127
128   cpu_set_t cpuset;
129   CPU_ZERO(&cpuset);
130   for(i=0; i < NUM_CPUS; i++)
131       CPU_SET(i, &cpuset);
132
133   fillbuffer();
134
135   /* setup common message q attributes */
136   mq_attr.mq_maxmsg = 10;
137   mq_attr.mq_msgsize = sizeof(void *)+sizeof(int);
138
139   mq_attr.mq_flags = 0;
140
141   mq_unlink(SNDRCV_MQ);  //Unlink if the previous message queue exists
142
143   mymq = mq_open(SNDRCV_MQ, O_CREAT|O_RDWR, S_IRWXU, &mq_attr);
144   if(mymq == (mqd_t)ERROR)
145   {
146     perror("mq_open");
147   }
148
149   int rt_max_prio, rt_min_prio;
150   rt_max_prio = sched_get_priority_max(SCHED_FIFO);
151   rt_min_prio = sched_get_priority_min(SCHED_FIFO);
152
153   //creating prioritized thread
154
155   //initialize  with default atrribute
156   rc = pthread_attr_init(&attr_receive);
157   //specific scheduling for Receiving
158   rc = pthread_attr_setinheritsched(&attr_receive, PTHREAD_EXPLICIT_SCHED);
159   rc = pthread_attr_setschedpolicy(&attr_receive, SCHED_FIFO);
160   rc=pthread_attr_setaffinity_np(&attr_receive, sizeof(cpu_set_t), &cpuset);
161   param_receive.sched_priority = rt_min_prio;
162   pthread_attr_setschedparam(&attr_receive, &param_receive);
163
```

```
164   //initialize  with default atrribute
165   rc = pthread_attr_init(&attr_send);
166   //specific scheduling for Sending
167   rc = pthread_attr_setinheritsched(&attr_send, PTHREAD_EXPLICIT_SCHED);
168   rc = pthread_attr_setschedpolicy(&attr_send, SCHED_FIFO);
169   rc=pthread_attr_setaffinity_np(&attr_send, sizeof(cpu_set_t), &cpuset);  //SC Added
170   param_send.sched_priority = rt_max_prio;
171   pthread_attr_setschedparam(&attr_send, &param_send);
172
173   if((rc=pthread_create(&th_send, &attr_send, sender, NULL)) == 0)
174   {
175     printf("\n\rSender Thread Created with rc=%d\n\r", rc);
176   }
177   else
178   {
179     perror("\n\rFailed to Make Sender Thread\n\r");
180     printf("rc=%d\n", rc);
181   }
182
183   if((rc=pthread_create(&th_receive, &attr_receive, receiver, NULL)) == 0)
184   {
185     printf("\n\r Receiver Thread Created with rc=%d\n\r", rc);
186   }
187   else
188   {
189     perror("\n\r Failed Making Reciever Thread\n\r");
190     printf("rc=%d\n", rc);
191   }
192
193   printf("pthread join send\n");
194   pthread_join(th_send, NULL);
195
196   printf("pthread join receive\n");
197   pthread_join(th_receive, NULL);
198 }
199
```