Real Time Linux Programming using Xenomai

Prepared by

Dr. Ramakrishnan Maharajan

Department of Electronics and Communication Engineering
School of Engineering Sciences
SRM University, AP
Amaravati – 522502

Exercise 1: Task Management -I

Aim: To create the RT_TASK using Xenomai 3.0.8 using Alchemy API.

Source Code:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

#include <alchemy/task.h>

RT_TASK hello_task,task1;

// function to be executed by task
void helloWorld(void *arg)
{
  RT_TASK_INFO curtaskinfo;

  rt_printf("Hello World!\n");

  // inquire current task
  rt_task_inquire(NULL,&curtaskinfo);

  // print task name
  rt_printf("Task name : %s \n", curtaskinfo.name);
}



// function to be executed by task
void task_1(void *arg)
{
  RT_TASK_INFO curtaskinfo1;

  // inquire current task
  rt_task_inquire(NULL,&curtaskinfo1);

  rt_printf("I am in task1\r\n");
  // print task name
  rt_printf("Task name : %s \n", curtaskinfo1.name);
}

int main(int argc, char* argv[])
{
  char  str[10],str1[10] ;

  printf("start task\n");
  sprintf(str,"hello");
   sprintf(str1,"task1");
  /* Create task
   * Arguments: &task,
   *          name,
   *          stack size (0=default),
   *          priority,
   *          mode (FPU, start suspended, ...)
   */
  rt_task_create(&hello_task, str, 0, 50, 0);
  rt_task_create(&task1, str1, 0, 40, 0);
  /*  Start task
   * Arguments: &task,
   *          task function,
   *          function argument
   */
  rt_task_start(&hello_task, &helloWorld, 0);
  rt_task_start(&task1, &task_1, 0);
rt_printf("End of main\r\n");
}
```

Output:

```
Hello World!
Task name : hello
I am in task1
Task name : task1
End of main
```

Inference: Though from main, we have created two tasks, with priority 30 and 40, as the main is considered as a Linux user space process, which has the lowest priority, it is preempted by the hello task . It runs first, Then the task1 will run. Then finally it comes to the main again.

Exercise 2: Task Management -II

Aim: Create the RT_TASKs inside the higher priority task “root”, to demonstrate ready state wait of the tasks created until root completes its execution.

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

#include <alchemy/task.h>

RT_TASK rt, hello_task,task1;

// function to be executed by task
void helloWorld(void *arg)
{
  RT_TASK_INFO curtaskinfo;

  rt_printf("Hello World!\n");

  // inquire current task
  rt_task_inquire(NULL,&curtaskinfo);

  // print task name
  rt_printf("Task name : %s \n", curtaskinfo.name);
}

// function to be executed by task
void task_1(void *arg)
{
  RT_TASK_INFO curtaskinfo1;

  // inquire current task
  rt_task_inquire(NULL,&curtaskinfo1);

  rt_printf("I am in task1\r\n");
  // print task name
  rt_printf("Task name : %s \n", curtaskinfo1.name);
}

// function to be executed by task
void root(void *arg)
{
  RT_TASK_INFO curtaskinfo;

  rt_printf("Root Task!\n");

  rt_task_create(&hello_task, "hello", 0, 30, 0);
  rt_task_create(&task1, "task1", 0, 40, 0);
```

```
   /*  Start task
    * Arguments: &task,
    *           task function,
    *           function argument
    */
  rt_task_start(&hello_task, &helloWorld, 0);
  rt_task_start(&task1, &task_1, 0);
  // inquire current task
  rt_task_inquire(NULL,&curtaskinfo);

  // print task name
  rt_printf("Task name : %s \n", curtaskinfo.name);
}

int main(int argc, char* argv[])
{

  printf("start task\n");
  /* Create task
    * Arguments: &task,
    *           name,
    *           stack size (0=default),
    *           priority,
    *           mode (FPU, start suspended, ...)
    */
  rt_task_create(&rt, "root", 0, 99, 0);
  rt_task_start(&rt, &root, 0);
rt_printf("End of main\r\n");
}
```

Output:

```
start task
Root Task!
Task name : root
I am in task1
Task name : task1
Hello World!
Task name : hello
End of main
```

Inference: Here, as the task is created from root, which has the highest priority, the tasks can not preempt the root. So both “hello_task” and “task1” waits and once root completes. Once the root execution is completed, task1 starts running as it has the higher priority. Then “Hello_task” will run. Then it comes back to the lower priority Linux main process.

Exercise 3: Task Management -III

Aim: Create the RT_TASKs which runs in the round robin fashion.

Code:

```
 #include <stdio.h>
   #include <signal.h>
   #include <unistd.h>
   #include <alchemy/task.h>
   #include <alchemy/sem.h>
   #include <alchemy/timer.h>

// Number of task to be created is 3
   #define NTASKS 3
// Task Ids
   RT_TASK demo_task[NTASKS];
// sem Id
   RT_SEM mysync;

   #define EXECTIME   2e8   // execution time in ns //50ms
   #define SPINTIME   1e7   // spin time in ns      //10ms

void demo(void *arg)
{
        RTIME starttime, runtime;
        int num=*(int *)arg;
        rt_printf("Task  : %d\n",num);
        rt_sem_p(&mysync,TM_INFINITE);

        // let the task run RUNTIME ns in steps of SPINTIME ns
        runtime = 0;
        while(runtime < EXECTIME)
        {
                rt_timer_spin(SPINTIME);  // spin cpu doing nothing
                runtime = runtime + SPINTIME;
                printf("Running Task  : %d  at time : %d\n",num,runtime);
        }
        printf("End Task  : %d\n",num);
}

//startup code
void startup()
{
        int i;
        char  str[20] ;
        // semaphore to sync task startup on
        rt_sem_create(&mysync,"MySemaphore",0,S_FIFO);

        for(i=0; i < NTASKS; i++)
        {
                printf("start task  : %d\n",i);
                sprintf(str,"task%d",i);
                rt_task_create(&demo_task[i], str, 0, 50, 0);
                rt_task_slice(&demo_task[i], SPINTIME);
                rt_task_start(&demo_task[i], &demo, &i);
        }
        printf("wake up all tasks\n");
        rt_sem_broadcast(&mysync);
}

int main(int argc, char* argv[])
{
        startup();
        printf("\nType CTRL-C to end this program\n\n" );
        pause();
}
```

Output:

```
start task  : 0
Task  : 0
start task  : 1
Task  : 1
start task  : 2
Task  : 2
wake up all tasks
Running Task  : 0  at time : 10000000
Running Task  : 1  at time : 10000000
Running Task  : 2  at time : 10000000
Running Task  : 0  at time : 20000000
Running Task  : 1  at time : 20000000
Running Task  : 2  at time : 20000000
Running Task  : 0  at time : 30000000
Running Task  : 1  at time : 30000000
Running Task  : 2  at time : 30000000
Running Task  : 0  at time : 40000000
Running Task  : 1  at time : 40000000
Running Task  : 2  at time : 40000000
Running Task  : 0  at time : 50000000
End Task  : 0
Running Task  : 1  at time : 50000000
End Task  : 1
Running Task  : 2  at time : 50000000
End Task  : 2

Type CTRL-C to end this program

^C
```

Exercise 4: Task Management -IV

Aim: Create the RT_TASKs which runs in the round robin fashion incrementing and decrementing the same global variable.

Code:

```
 #include <stdio.h>
 #include <signal.h>
 #include <unistd.h>
   #include <alchemy/task.h>
   #include <alchemy/sem.h>
   #include <alchemy/timer.h>

// Number of task to be created is 3
   #define ITER 1000000
// Task Ids
   RT_TASK t1,t2,rt;
   int global =0;

   #define EXECTIME   5e7   // execution time in ns //50ms
   #define SPINTIME   1e7   // spin time in ns      //10ms

void taskOne(void *arg)

{
    int i;
    RTIME runtime = 0;
    while(runtime < EXECTIME)
    {
            rt_timer_spin(SPINTIME);  // spin cpu doing nothing
            runtime = runtime + SPINTIME;
            rt_printf("I am taskOne and global = %d................\n", ++global);
  }
}


void taskTwo(void *arg)
{
    int i;
    RTIME runtime = 0;
   while(runtime < EXECTIME)
    {
            rt_timer_spin(SPINTIME);  // spin cpu doing nothing
            runtime = runtime + SPINTIME;
           rt_printf("I am taskTwo and global = %d---------------\n", --global);
  }
}
```

```
//startup code
void root()
{
        int i;

        printf("root task\n");
   /* create the two tasks */

   rt_task_create(&t1, "task1", 0, 1, 0);
   rt_task_slice(&t1, SPINTIME);
   rt_task_create(&t2, "task2", 0, 1, 0);
   rt_task_slice(&t2, SPINTIME);

   /* start the two tasks */

   rt_task_start(&t1, &taskOne, 0);

   rt_task_start(&t2, &taskTwo, 0);
printf("root task ends\n");

}

int main(int argc, char* argv[])
{
   rt_task_create(&rt, "root", 0, 99, 0);
   rt_task_start(&rt, &root, 0);
        printf("\nType CTRL-C to end this program\n\n" );
        pause();
}
```

Output:

```
root task
root task ends
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------

Type CTRL-C to end this program

^C
```

Exercise 5: Inter Task Synchronization

Aim: Synchronize the two tasks accessing the shared variable such that the task execution alternate between each other using semaphore.

Code:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

#include <alchemy/task.h>
#include <alchemy/timer.h>
#include <alchemy/sem.h>

#define ITER 10

static RT_TASK  t1;
static RT_TASK  t2;

int global = 0;
// sem Id
  RT_SEM producer, consumer;
void taskOne(void *arg)
{
   int i;
   for (i=0; i < ITER; i++) {
      rt_sem_p(&producer,TM_INFINITE);
      printf("I am taskOne and global = %d................\n", ++global);
         rt_sem_v(&consumer);

   }
}

void taskTwo(void *arg)
{
   int i;
   for (i=0; i < ITER; i++) {
         rt_sem_p(&consumer,TM_INFINITE);
      printf("I am taskTwo and global = %d----------------\n", --global);
         rt_sem_v(&producer);
   }
}

int main(int argc, char* argv[]) {
         // semaphore to sync tasks
   rt_sem_create(&producer,"Producer",1,S_FIFO);
   rt_sem_create(&consumer,"Consumer",0,S_FIFO);
   rt_task_create(&t1, "task1", 0, 1, 0);
```

```
    rt_task_create(&t2, "task2", 0, 1, 0);
    rt_task_start(&t1, &taskOne, 0);
    rt_task_start(&t2, &taskTwo, 0);
    return 0;
}
```

Output:

```
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
I am taskOne and global = 1................
I am taskTwo and global = 0----------------
```

Exercise 6: Priority Inversion Problem and Solution

Aim: Write a program to demonstrate the priority inversion problem.

Code:

```
  #include <stdio.h>
  #include <signal.h>
  #include <unistd.h>
  #include <alchemy/task.h>
  #include <alchemy/sem.h>
  #include <alchemy/timer.h>

// Task Ids
  RT_TASK lpt,mpt,hpt;

// sem Id
  RT_SEM mysync;

  #define EXECTIME   5e7   // execution time in ns //50ms
  #define SPINTIME   1e7   // spin time in ns      //10ms
void mp_task(void *arg);
void hp_task(void *arg);

void lp_task(void *arg)
{
        RTIME runtime;
        rt_printf("lp_task: Running\n");
        rt_printf("lp_task: Trying to take semaphore -mysync\n");
        rt_sem_p(&mysync,TM_INFINITE);
        rt_printf("lp_task: Took semaphore -mysync\n");
        rt_printf("lp_task: About to start Higher Priority Task\n");
        rt_task_start(&hpt, &hp_task, 0);
        rt_printf("lp_task: About to start Medium Priority Task\n");
        rt_task_start(&mpt, &mp_task, 0);
        rt_printf("lp_task: About to release semaphore mysync\n");
        rt_sem_v(&mysync);
        rt_printf("lp_task: Execution completes\n");
}
void mp_task(void *arg)
{
        RTIME runtime;
        rt_printf("mp_task: Started\n");

        // let the task run RUNTIME ns in steps of SPINTIME ns
        runtime = 0;
        while(runtime < EXECTIME)
        {
                rt_timer_spin(SPINTIME);  // spin cpu doing nothing
                runtime = runtime + SPINTIME;
                rt_printf("mp_task: Running\n");
        }
        rt_printf("mp_task: Execution completes\n");
}


void hp_task(void *arg)
{
        RTIME runtime;
        rt_printf("hp_task: Running\n");
        rt_printf("hp_task: Trying to take semaphore -mysync which is held by lp_task\n");
        rt_sem_p(&mysync,TM_INFINITE);
        rt_printf("hp_task: Took semaphore -mysync\n");

        // let the task run RUNTIME ns in steps of SPINTIME ns
        runtime = 0;
        while(runtime < EXECTIME)
        {
                rt_timer_spin(SPINTIME);  // spin cpu doing nothing
                runtime = runtime + SPINTIME;
                rt_printf("hp_task: Running\n");
        }
        rt_printf("hp_task: Execution completes\n");
}
//startup code
void startup()
{
        // semaphore to sync task startup on
        rt_sem_create(&mysync,"MySemaphore",1,S_FIFO);
        rt_task_create(&lpt, "lp_task", 0, 50, 0);
        rt_task_create(&mpt, "mp_task", 0, 60, 0);
        rt_task_create(&hpt, "hp_task", 0, 70, 0);
       rt_printf("startup(): About to start Lower Priority Task\n");
        rt_task_start(&lpt, &lp_task, 0);
}

int main(int argc, char* argv[])
{
        startup();
        printf("\nType CTRL-C to end this program\n\n" );
        pause();
}
```

Description: Priority inversion is the problem happens when the high priority task is waiting for the resource held by low priority task and the low priority task is preempted by medium priority tasks. In this case, the higher priority task waiting for the low priority task is acceptable as the low priority task has the resource. But the higher priority task is made to wait for the medium priority tasks which is the priority inversion. To avoid this Xenomai provides the mutex which has the solution for the priority inversion. When we use the mutex services of Xenomai, it automatically elevates the priority of the task holding the resource to the priority of the task seeking the resource. This solution is called priority inheritance.

Priority Inversion Solution: Priority Inheritance (with Mutex)

Code:

```
 #include <stdio.h>
   #include <signal.h>
   #include <unistd.h>
   #include <alchemy/task.h>
   #include <alchemy/mutex.h>
   #include <alchemy/timer.h>

// Task Ids
   RT_TASK lpt,mpt,hpt;

// sem Id
   RT_MUTEX mymutex;

   #define EXECTIME   5e7   // execution time in ns //50ms
   #define SPINTIME   1e7   // spin time in ns      //10ms
void mp_task(void *arg);
void hp_task(void *arg);

void lp_task(void *arg)
{
        RTIME runtime;
        rt_printf("lp_task: Running\n");
        rt_printf("lp_task: Trying to take mutex -mymutex\n");
        rt_mutex_acquire(&mymutex,TM_INFINITE);
        rt_printf("lp_task: Took mutex -mymutex\n");
        rt_printf("lp_task: About to start Higher Priority Task\n");
        rt_task_start(&hpt, &hp_task, 0);
        rt_printf("lp_task: About to start Medium Priority Task\n");
        rt_task_start(&mpt, &mp_task, 0);
        rt_printf("lp_task: About to release mutex mymutex\n");
        rt_mutex_release(&mymutex);
        rt_printf("lp_task: Execution completes\n");
}
void mp_task(void *arg)
{
        RTIME runtime;
        rt_printf("mp_task: Started\n");

        // let the task run RUNTIME ns in steps of SPINTIME ns
        runtime = 0;
        while(runtime < EXECTIME)
        {
                rt_timer_spin(SPINTIME);  // spin cpu doing nothing
                runtime = runtime + SPINTIME;
                rt_printf("mp_task: Running\n");
        }
        rt_printf("mp_task: Execution completes\n");
}


void hp_task(void *arg)
{
        RTIME runtime;
        rt_printf("hp_task: Running\n");
        rt_printf("hp_task: Trying to take mutex -mymutex which is held by lp_task\n");
        rt_mutex_acquire(&mymutex,TM_INFINITE);
        rt_printf("hp_task: Took mutex -mymutex\n");

        // let the task run RUNTIME ns in steps of SPINTIME ns
        runtime = 0;
        while(runtime < EXECTIME)
        {
                rt_timer_spin(SPINTIME);  // spin cpu doing nothing
                runtime = runtime + SPINTIME;
                rt_printf("hp_task: Running\n");
        }
        rt_mutex_release(&mymutex);
        rt_printf("hp_task: released the mutex\n");
        rt_printf("hp_task: Execution completes\n");
}
//startup code
void startup()
{
        // mutex to sync task startup on
        rt_mutex_create(&mymutex,"Mymutex");
        rt_task_create(&lpt, "lp_task", 0, 50, 0);
        rt_task_create(&mpt, "mp_task", 0, 60, 0);
        rt_task_create(&hpt, "hp_task", 0, 70, 0);
        rt_printf("startup(): About to start Lower Priority Task\n");
        rt_task_start(&lpt, &lp_task, 0);
}

int main(int argc, char* argv[])
{
        startup();
        printf("\nType CTRL-C to end this program\n\n" );
        pause();
}
```

output:

```
startup(): About to start Lower Priority Task
lp_task: Running
lp_task: Trying to take mutex -mymutex
lp_task: Took mutex -mymutex
lp_task: About to start Higher Priority Task
hp_task: Running
hp_task: Trying to take mutex -mymutex which is held by lp_task
lp_task: About to start Medium Priority Task
lp_task: About to release mutex mymutex
hp_task: Took mutex -mymutex
hp_task: Running
hp_task: Running
hp_task: Running
hp_task: Running
hp_task: Running
hp_task: released the mutex
hp_task: Execution completes
mp_task: Started
mp_task: Running
mp_task: Running
mp_task: Running
mp_task: Running
mp_task: Running
mp_task: Execution completes
lp_task: Execution completes

Type CTRL-C to end this program

^C
```

Exercise 7: Inter Process Communication

Aim: Write the program to use message queues for the communication between the two tasks.

Source code:

```
/* ex05example.c */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>

#include <alchemy/task.h>
#include <alchemy/timer.h>
#include <alchemy/queue.h>

#define NTASKS 2

#define QUEUE_SIZE 255
#define MAX_MESSAGE_LENGTH 40

RT_TASK task_struct[NTASKS];

#define QUEUE_SIZE 255
RT_QUEUE myqueue;

void taskOne(void *arg)
{
   int retval;
   char message[] = "Message from taskOne";

   /* send message */
   retval = rt_queue_write(&myqueue,message,sizeof(message),Q_NORMAL);

   if (retval < 0 ) {
     rt_printf("Sending error\n");
   } else {
     rt_printf("taskOne sent message to mailbox\n");
   }
}

void taskTwo(void *arg)
{
   int retval;
   char msgBuf[MAX_MESSAGE_LENGTH];

   /* receive message */
   retval = rt_queue_read(&myqueue,msgBuf,sizeof(msgBuf),TM_INFINITE);
   if (retval < 0 ) {
         rt_printf("Receiving error\n");
    } else {
        rt_printf("taskTwo received message: %s\n",msgBuf);
        rt_printf("with length %d\n",retval);
    }
}

//startup code
void startup()
{
  int i;
  char  str[10] ;

  void (*task_func[NTASKS]) (void *arg);
  task_func[0]=taskOne;
  task_func[1]=taskTwo;

  rt_queue_create(&myqueue,"myqueue",QUEUE_SIZE,10,Q_FIFO);

  for(i=0; i < NTASKS; i++) {
    rt_printf("start task  : %d\n",i);
    sprintf(str,"task%d",i);
    rt_task_create(&task_struct[i], str, 0, 50, 0);
    rt_task_start(&task_struct[i], task_func[i], &i);
  }
}

int main(int argc, char* argv[])
{
  printf("\nType CTRL-C to end this program\n\n" );


  //startup code
  startup();

  pause();
}
```

Output:

```
root@xenomai308:/home/des/Desktop/xenomai/IPC# ./ipc
start task  : 0
taskOne sent message to mailbox
start task  : 1
taskTwo received message: Message from taskOne
with length 21

Type CTRL-C to end this program

^C
```