THE UNIVERSITY OF VICTORIA

DEPARTMENT OF ELECTRICAL AND

COMPUTER ENGINEERING

CENG 455: REAL TIME COMPUTER SYSTEMS DESIGN

PROJECTS

March 27th 2017

Project 3 Report

Group:

Clinton Bestmann	V00810641
Jinmin Huang	V00787639
Ngoc Thinh Nguyen	V00817304

Lab TA:
Ahmad Lashgar

Report Mark Distribution (100%):

10%	Document organization
5%	Project description and requirements
5%	List API calls, overview functionality and prototype
5%	Table of task priorities
5%	Interpretation of time (MQX ticks or wallclock)
10%	Diagram to show all tasks and interaction (e.g. messaging and interfacing)
10%	Pseudo code of scheduler and all API calls
10%	Communication mechanisms between API calls and scheduler
5%	Problems, bugs, and lab limitations
25%	Comprehensive Test Strategy, showing how you did test your design to assure it handles corner cases.
10%	Complete project source code as an attachment (ZIP file)

Introduction

The purpose of this project is to write a task scheduler using the deadline driven scheduling algorithm that deals with periodic and aperiodic tasks that have hard deadlines. Showing the statistics of a variety of task loads is also part of the project.

Project description and requirements

There are three main parts of this project: the deadline scheduler task (DD-scheduler), the Task creator that creates the auxiliary tasks, and the API functions. The auxiliary tasks are used to test the DD-scheduler. There are other aspects of the project that are needed for the project to function as a unit. These will be explained below.

DD-scheduler

The scheduler is dynamically able to alter the priorities of the currently active tasks so that a dynamic real time environment can be controlled. The scheduler uses a priority-based task scheduling provided by MQX as a mechanism to achieve fine scheduling control. The scheduler implements the earliest deadline driven scheduling where task with earlier deadline are given higher priority. The scheduler is implemented as a task in MQX, and we gave it a priority of 13, which is the second highest priority. The scheduler is the task have responsible for scheduling user tasks.

The scheduler is woken every time either a scheduling request message arrives (such as during task create or task delete) or when a scheduled task is past its deadline. The message arrives: a) at task creation time so that correct priority could be given to the new task by using link list, b) at task deletion time so that the task with the earliest deadline will be assigned to the highest priority, c) at the next deadline.

The DD-scheduler maintains a list of active tasks with their priorities and deadlines. We used a double link list to keep a list of current user tasks. This list is sorted by the user task’s deadlines. The DD-scheduler is able to adjust the priorities of the user task to ensure that the user task with the soonest deadline is running first. The DD-scheduler is able to create a new user tasks, and delete a task depend on the content of the message it has received.

User tasks

Users tasks are tasks that user has created and wants scheduled. These task must have fixed execution time and hard deadlines. We also assumed that these tasks are independent of each other meaning that, a user task would not depend on a message from another user task in order to complete its execution time.

Generator task

The generator task was used to generate periodic and aperiodic user tasks. We generated user tasks by calling dd-create(). The user tasks were created with a fixed execution time and a fixed deadline. The tasks deletes itself by using dd-delete when it has finished its execution.

Idle task

Idle task is the task running whenever no other user tasks currently running. The purpose of this task is to prevent MQX from scheduling tasks that we do not want scheduled.

Monitor task

The monitor task is a task with the lowest priority that will run when no other tasks are running. This task is used to calculate and report processor utilization and system overhead.

Initial Design

This project was quite large in scope, and we felt that we would be easily overwhelmed. So in order to mitigate this issue, we started the project by laying out a top level design of the entirety of the system based on our current understanding of the project specifications. This design diagram is shown in figure 1 and 2.

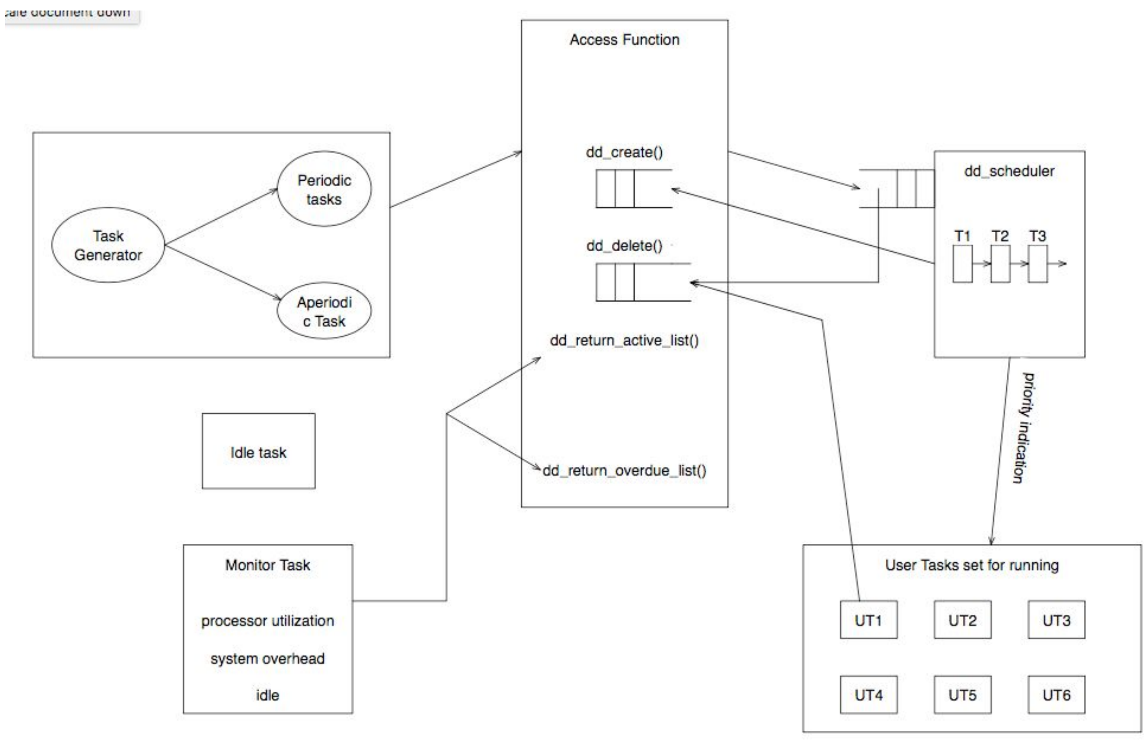


Figure 1: System Top level Diagram

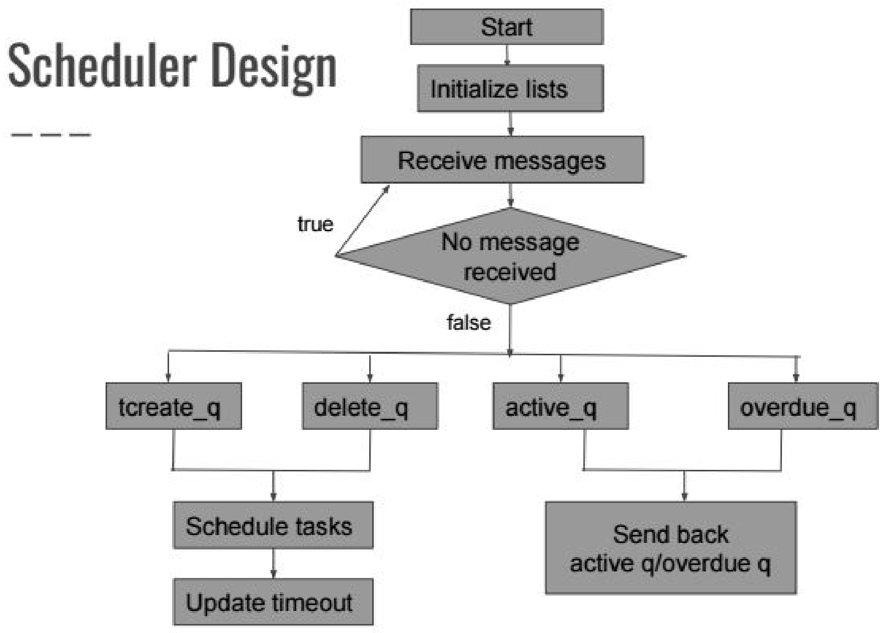


Figure 2: design of scheduler

Table of priorities

Name of tasks	Priorities	status
Generator	10	Auto
Scheduler	13	Auto
Running task	15	
Idle task	17	Auto
Monitor	19	Auto
Created Tasks	20	

Global structures

All structures and global variables are defined in the global_variables.h.

- *t_list*: is a double link list structure store information for a list of tasks.
- *start_t*: check the start time of each task
- *message*: a struct stores header and data for message pointer.
- *is_running*: is an integer .used to check if a task is running or not
- *interrupt_occurs*: is boolean type used to check if the interrupt has occured or not
- *running_task*: get the task id of which task is currently running.
- *msg_pool*: initialize a message pool.
- *dd_id, tcreate_qid, delete_qid*: these are queue id of tcreate, dd_id, and delete queues.
- num_aperiod: return the number of aperiodic tasks
- sys_time: used to calculate the system time
- total_idle_time :used to calculate total idle time
- total_running_time: used to calculate total running time

We attempted to model the scheduler based on the initial design given in the project specifications (see figure 2). Based on the pseudo codes for each module of the project, we see that the system can be broken down into parts which include: the dd_scheduler, the generator, Access Functions, Global Structures, the user tasks, idle task, and monitor task.

Pseudo codes

dd_Scheduler pseudo code

```
Create dd_active_list_head;
Create dd_active_list_tail;
Create dd_active_list_current;
Create dd_overdue_list_head;
Create dd_overdue_list_current;

Create old_priority;
Create new_priority;
Create_check_priority;
Change priority of running task to 16

Create queue for dd_scheduler
Create message pool

Wait for message from access functions
If ( message.source_qid == tcreate_qid)
        Put this task into link list
        Sort the link list
        Display the active task
        Send the message back to dd_create
If (message.source_qid == delete_qid) {
        Go to the link list
        Delete indicated task in the link list
        Display link list again
        Send message back to dd_delete
}
if(message.source_qid == return_active_list_qid) {

        Initialize an active task list
        Store all the task in current link list into the active task list
        Send the message contain the active list back  to the requestor
}
if(message.source_qid == return_active_list_qid){
        Initialize an overdue task list
        Store all the overdue task  into the overdue task list
        Send the message contain the overdue task  list back  to the requestor
}
```

Access Functions

The Access functions were designed to abstract the functionality of the dd_scheduler from the various user tasks. This means that the task do not manipulate any global structures or flags. The flags also use the access functions as their only means of communication with the d_scheduler tasks by sending messages using dd_tcreate, dd_delete, dd_return_active_list, and dd_return_overdue_list functions. As per the specifications, four access functions were created; their functionality and design is further explained in the following sections.

dd_tcreate

Dd_create is used to create a user task, the scheduler put the task id and other information into the task list and sort the list.

```
task_id dd_tcreate(uint_32 tempplate_index, uint_32 deadline) {
        Open a queue
        Create the task specified and assigns it the minimum priority possible
        Compose a create_task_message and send it to the dd_scheduler
        Wait for a reply from dd_scheduler at the queue it created above
        Obtain the reply
        Destroy the queue
        Return to the task id of the created task or error
}
```

dd_delete

This function sends the message to dd_scheduler and request scheduler to delete a indicated task from the task list. This structure is parallel with dd_tcreate.

```
Uint_32 dd_delete(uint_32 task_id) {
        Open a queue
        Create a delete_qid, and get_id from dd_scheduler
        Compose a delete_message and send it to scheduler
        Waits from a reply at the queue
        One the reply is received, it obtain
        Destroys the queue
        Return to the invoking task: task id or error
}
```

dd_return_active_list

This function is responsible to request the dd_scheduler for the list of active tasks and return this information to the requestor. Our design, the list was copied and send to the requestor.

Uint_32 **dd_return_active_list**(struct task_list **list) {

```
        Initialize a dd_return_active_list_qid
        Open a queue
        Open message pool
        Compose the dd_return_active_task_list and send to scheduler
        Wait for a reply at the queue
        Obtain the rely
        Destroy the queue
        Return the list of active task
}
```

dd_return_overdue_list

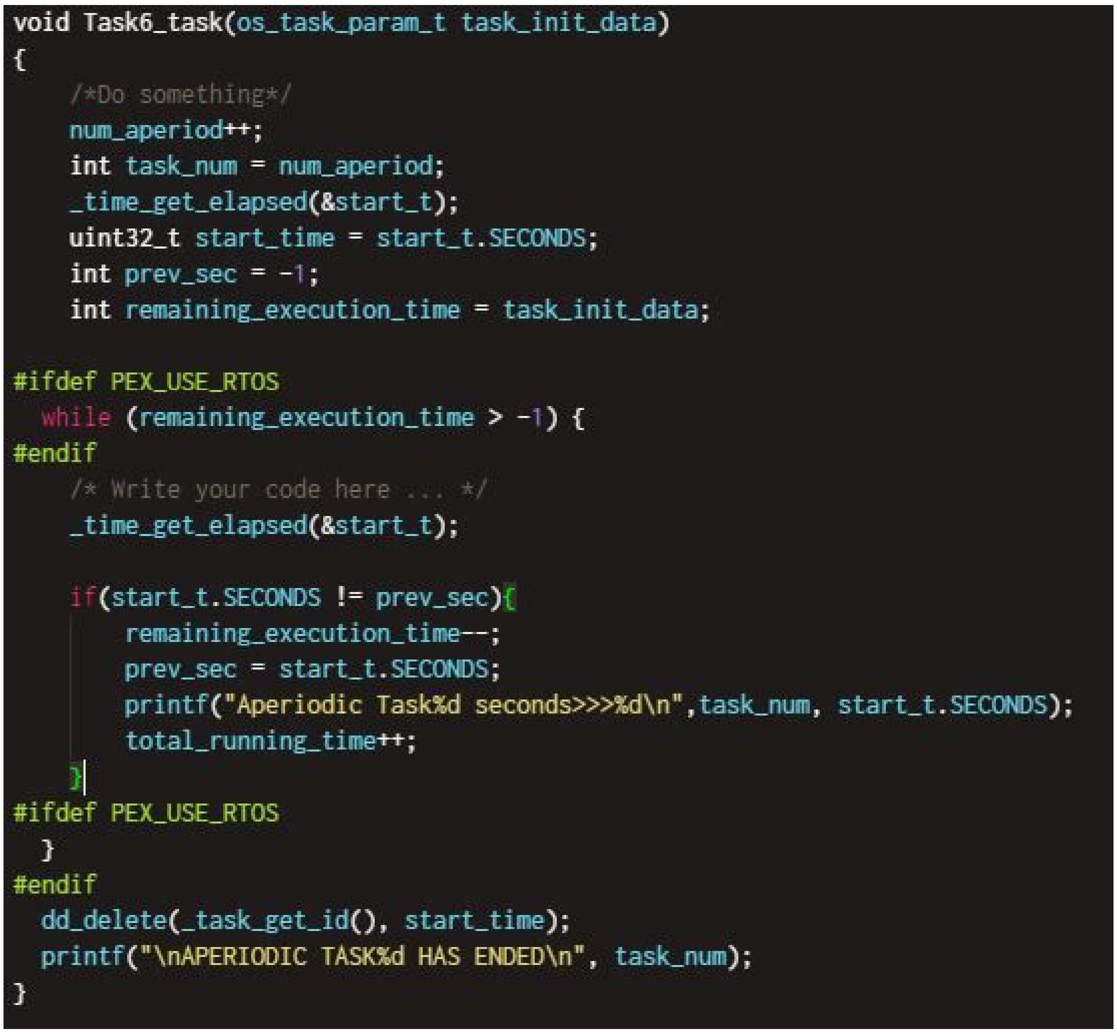
This function is responsible to request the dd_scheduler for the list of overdue tasks and return this information to the requestor. Our design, the list was copied and send to the requestor.

Uint_32 **dd_return_active_list**(struct task_list **list) {

```
        Initialize a dd_return_overdue_list_qid
        Open a queue
        Open message pool
        Compose the dd_return_overdue_task_list and send to scheduler
        Wait for a reply at the queue
        Obtain the rely
        Destroy the queue
        Return the list of active task
}
```

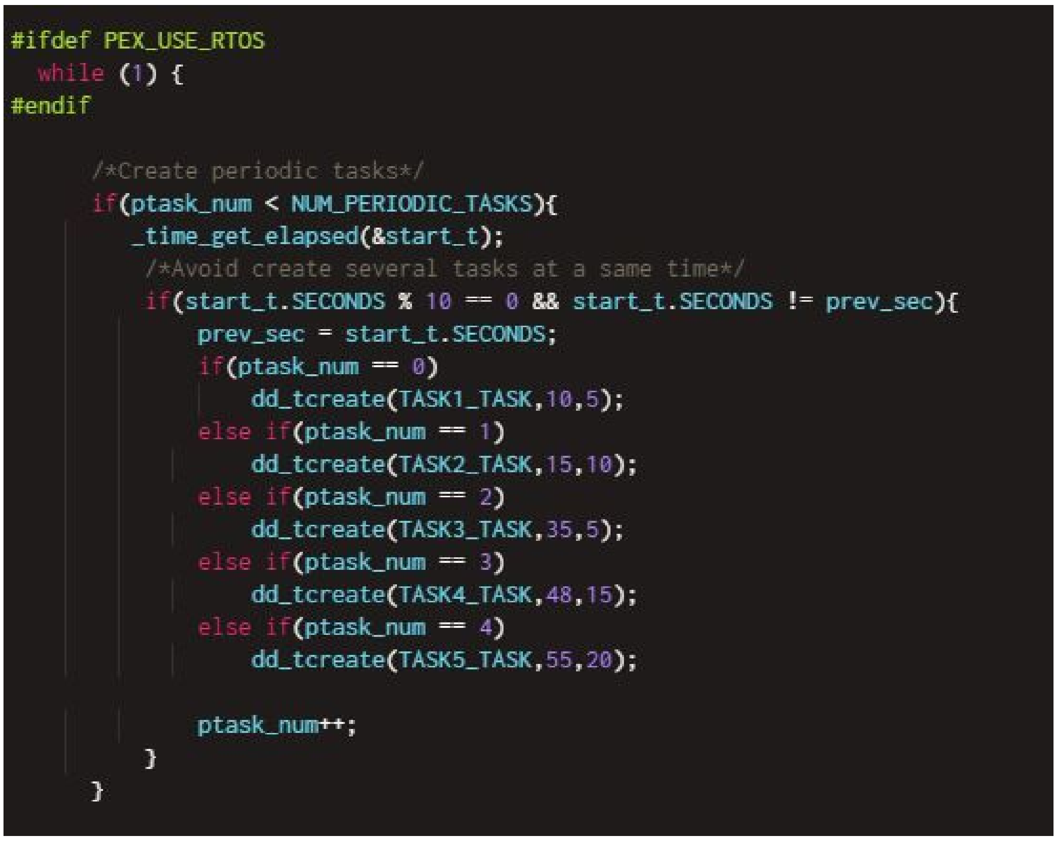
Testing

The system was tested using both periodic and aperiodic tasks. Five periodic tasks were created by the Task creator, and we were able to modify the execution time and deadline of each created task. We also used a random number generator function to generate the Aperiodic tasks. Aperiodic tasks with random arrival time, execution time and deadline.



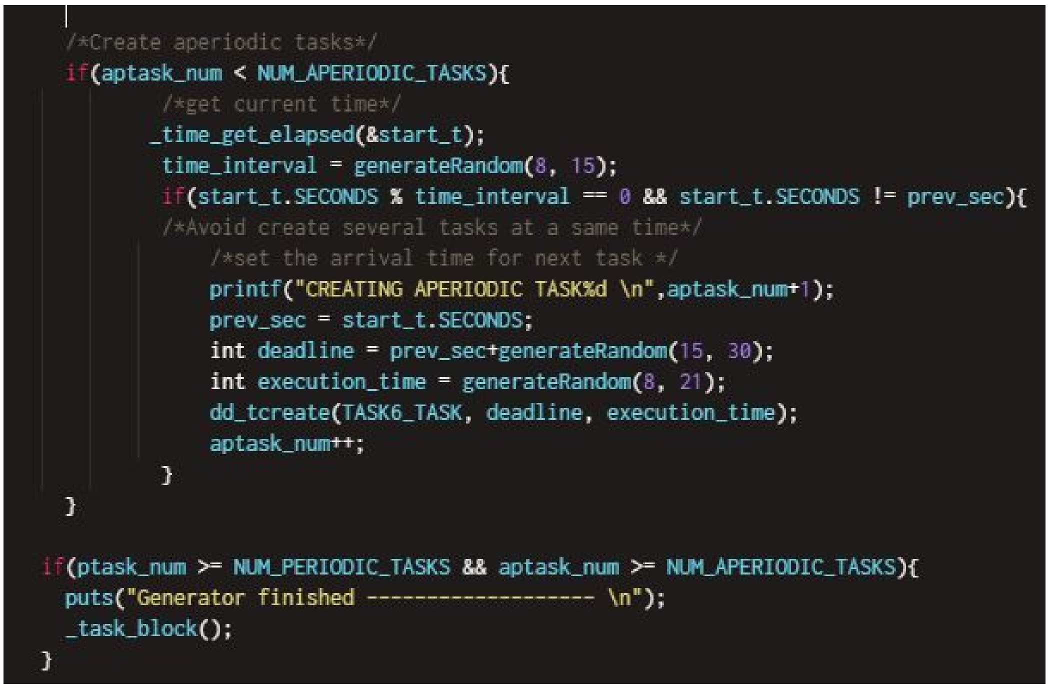
```c
void Task6_task(os_task_param_t task_init_data)
{
    /*Do something*/
    num_aperiod++;
    int task_num = num_aperiod;
    _time_get_elapsed(&start_t);
    uint32_t start_time = start_t.SECONDS;
    int prev_sec = -1;
    int remaining_execution_time = task_init_data;

#ifdef PEX_USE_RTOS
  while (remaining_execution_time > -1) {
#endif
    /* Write your code here ... */
    _time_get_elapsed(&start_t);

    if(start_t.SECONDS != prev_sec){
        remaining_execution_time--;
        prev_sec = start_t.SECONDS;
        printf("Aperiodic Task%d seconds>>>%d\n",task_num, start_t.SECONDS);
        total_running_time++;
    }
#ifdef PEX_USE_RTOS
  }
#endif
  dd_delete(_task_get_id(), start_time);
  printf("\nAPERIODIC TASK%d HAS ENDED\n", task_num);
}
```

Figure 3: code of task template



```
#ifdef PEX_USE_RTOS
  while (1) {
#endif

      /*Create periodic tasks*/
      if(ptask_num < NUM_PERIODIC_TASKS){
         _time_get_elapsed(&start_t);
          /*Avoid create several tasks at a same time*/
          if(start_t.SECONDS % 10 == 0 && start_t.SECONDS != prev_sec){
              prev_sec = start_t.SECONDS;
              if(ptask_num == 0)
                  dd_tcreate(TASK1_TASK,10,5);
              else if(ptask_num == 1)
                  dd_tcreate(TASK2_TASK,15,10);
              else if(ptask_num == 2)
                  dd_tcreate(TASK3_TASK,35,5);
              else if(ptask_num == 3)
                  dd_tcreate(TASK4_TASK,48,15);
              else if(ptask_num == 4)
                  dd_tcreate(TASK5_TASK,55,20);

              ptask_num++;
          }
      }
```

Figure 4: code of creating periodic tasks



```
    /*Create aperiodic tasks*/
    if(aptask_num < NUM_APERIODIC_TASKS){
            /*get current time*/
           _time_get_elapsed(&start_t);
            time_interval = generateRandom(8, 15);
            if(start_t.SECONDS % time_interval == 0 && start_t.SECONDS != prev_sec){
            /*Avoid create several tasks at a same time*/
                /*set the arrival time for next task */
                printf("CREATING APERIODIC TASK%d \n",aptask_num+1);
                prev_sec = start_t.SECONDS;
                int deadline = prev_sec+generateRandom(15, 30);
                int execution_time = generateRandom(8, 21);
                dd_tcreate(TASK6_TASK, deadline, execution_time);
                aptask_num++;
            }
    }

  if(ptask_num >= NUM_PERIODIC_TASKS && aptask_num >= NUM_APERIODIC_TASKS){
    puts("Generator finished ------------------- \n");
    _task_block();
  }
```

Figure 5: code of creating aperiodic tasks

Strategy

- Deadline-Driven testing:
    1. Create 5 tasks with same arrival time and different deadlines. Task with shortest deadline should always run first.
    2. Create several tasks such as (three numbers are arrival time, execution time and deadline respectively)   T1(0, 8, 10), T2(3, 4, 20), T3(4, 2, 17), T4(5, 3, 18), T5(6, 3, 21). And see whether those tasks run in right order.

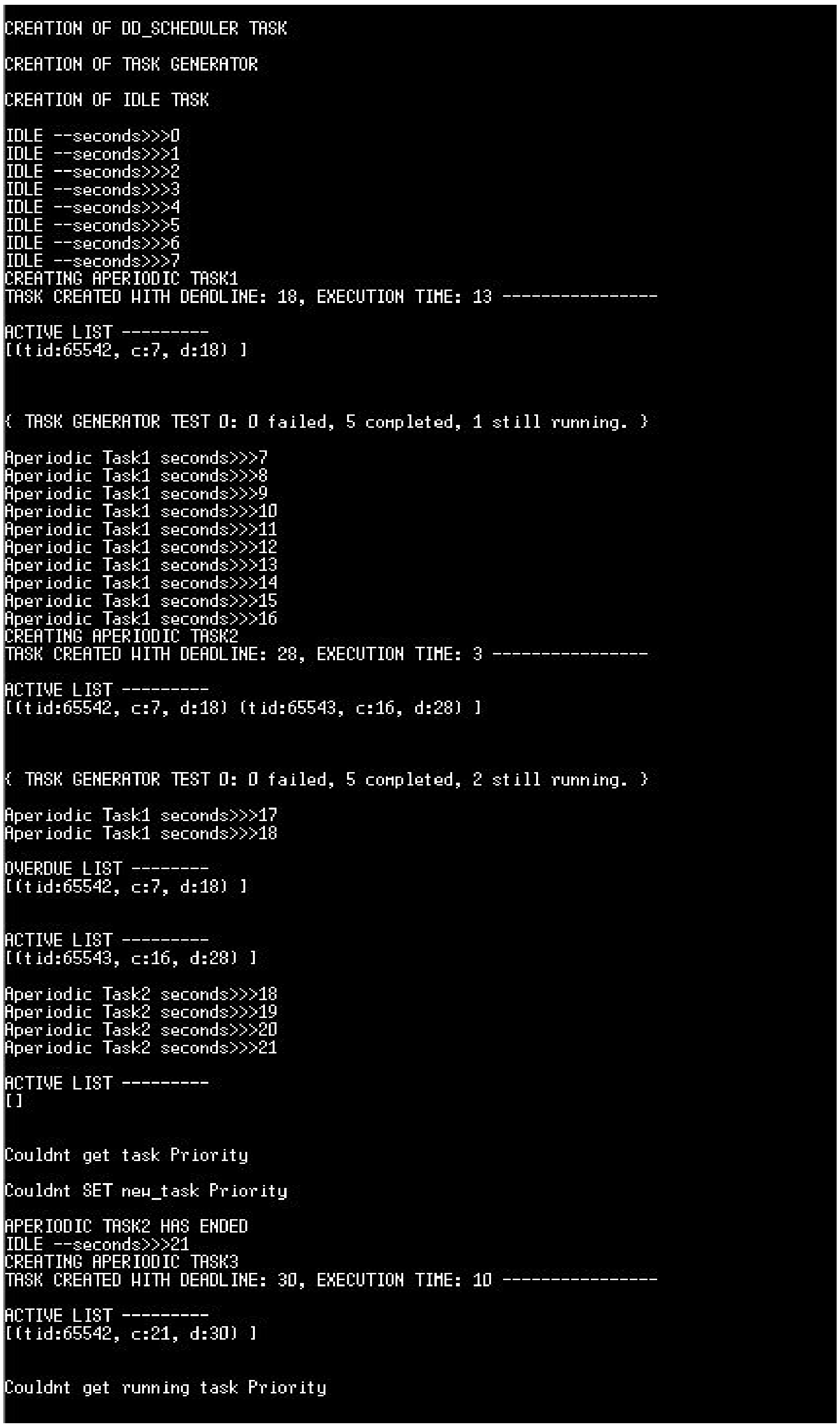
- Preemption testing:
    1. Create two periodic tasks and let the second one to preempt the first task.
    2. Create five periodic tasks and let every next task to preempt the previous task.
    3. Create two task with same deadline but different arrival time. The second task should not preempt the first task
    4. Create several periodic tasks with different deadlines(no order). Execute the program to see if tasks running follow the gantt chart.
    5. Add aperiodic task to test

- Overdue testing:
    1. Create one task and let it overdue.
    2. Create two tasks and let the both of them overdue.
    3. Create two tasks and let the second one arrival time equals to deadline.
    4. Create two tasks and let the second one arrival time greater than deadline.
    5. Create ten tasks and let four tasks overdue, four tasks run normally and two tasks arrival time greater than deadline.

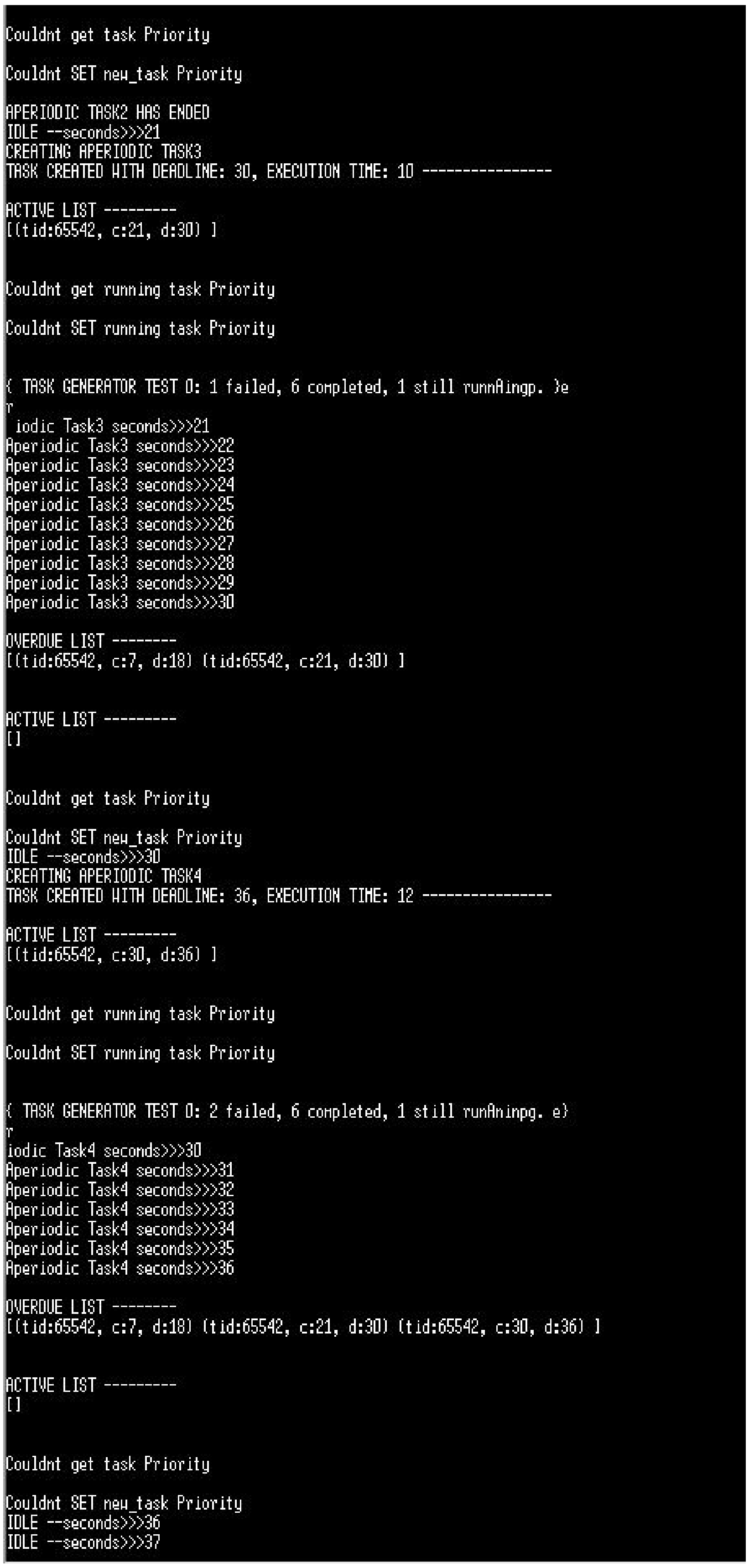
- Delete testing:
    1. Create one task running without overdue. Task should be deleted after finishing.
    2. Combine the test with overdue test. Overdue task should be deleted in time.

Testing Result

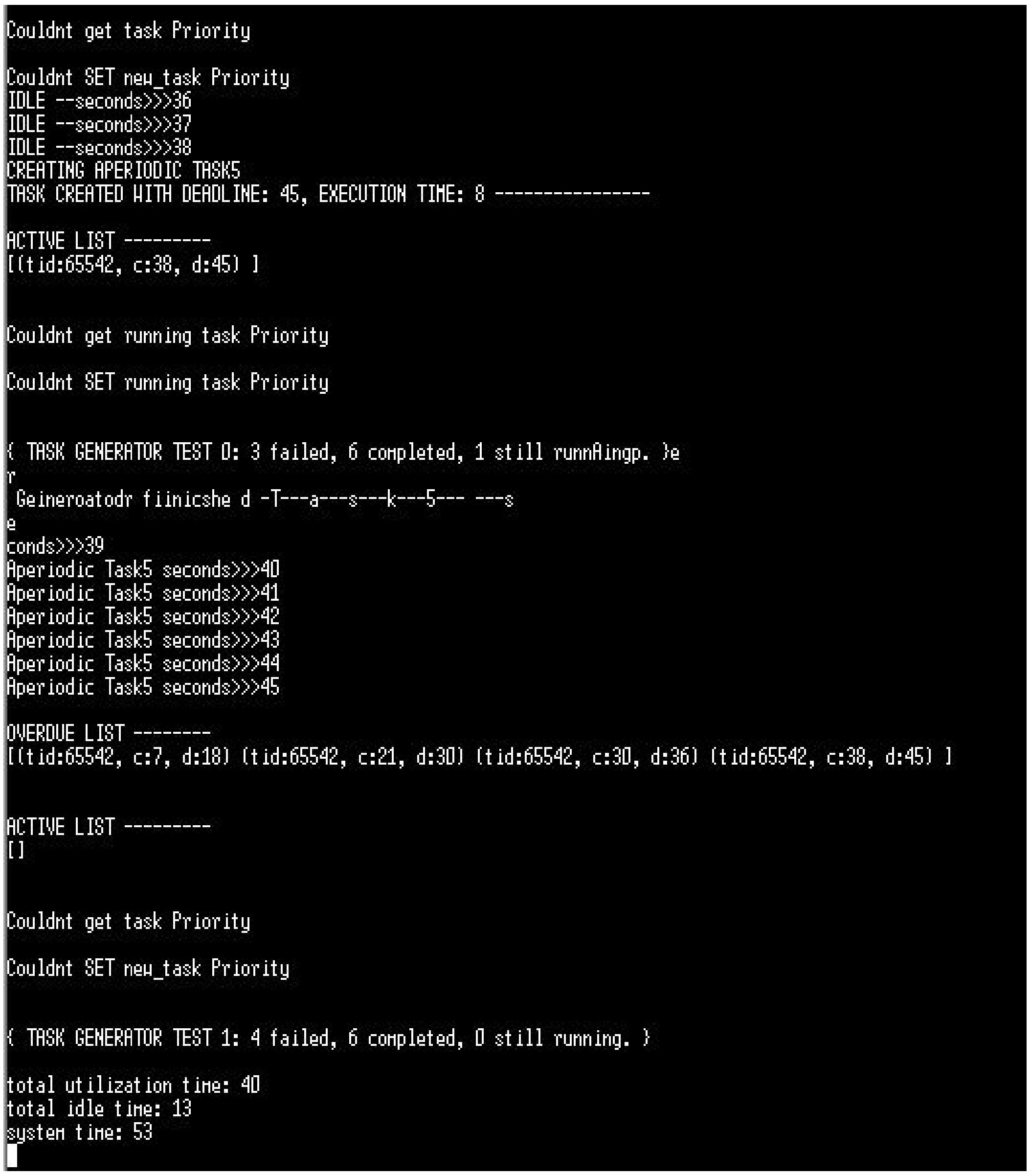
Aperiodic tasks



```
CREATION OF DD_SCHEDULER TASK

CREATION OF TASK GENERATOR

CREATION OF IDLE TASK

IDLE --seconds>>>0
IDLE --seconds>>>1
IDLE --seconds>>>2
IDLE --seconds>>>3
IDLE --seconds>>>4
IDLE --seconds>>>5
IDLE --seconds>>>6
IDLE --seconds>>>7
CREATING APERIODIC TASK1
TASK CREATED WITH DEADLINE: 18, EXECUTION TIME: 13 ----------------

ACTIVE LIST ---------
[(tid:65542, c:7, d:18) ]


{ TASK GENERATOR TEST 0: 0 failed, 5 completed, 1 still running. }

Aperiodic Task1 seconds>>>7
Aperiodic Task1 seconds>>>8
Aperiodic Task1 seconds>>>9
Aperiodic Task1 seconds>>>10
Aperiodic Task1 seconds>>>11
Aperiodic Task1 seconds>>>12
Aperiodic Task1 seconds>>>13
Aperiodic Task1 seconds>>>14
Aperiodic Task1 seconds>>>15
Aperiodic Task1 seconds>>>16
CREATING APERIODIC TASK2
TASK CREATED WITH DEADLINE: 28, EXECUTION TIME: 3 ----------------

ACTIVE LIST ---------
[(tid:65542, c:7, d:18) (tid:65543, c:16, d:28) ]


{ TASK GENERATOR TEST 0: 0 failed, 5 completed, 2 still running. }

Aperiodic Task1 seconds>>>17
Aperiodic Task1 seconds>>>18

OVERDUE LIST --------
[(tid:65542, c:7, d:18) ]


ACTIVE LIST ---------
[(tid:65543, c:16, d:28) ]

Aperiodic Task2 seconds>>>18
Aperiodic Task2 seconds>>>19
Aperiodic Task2 seconds>>>20
Aperiodic Task2 seconds>>>21

ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority

APERIODIC TASK2 HAS ENDED
IDLE --seconds>>>21
CREATING APERIODIC TASK3
TASK CREATED WITH DEADLINE: 30, EXECUTION TIME: 10 ----------------

ACTIVE LIST ---------
[(tid:65542, c:21, d:30) ]


Couldnt get running task Priority
```

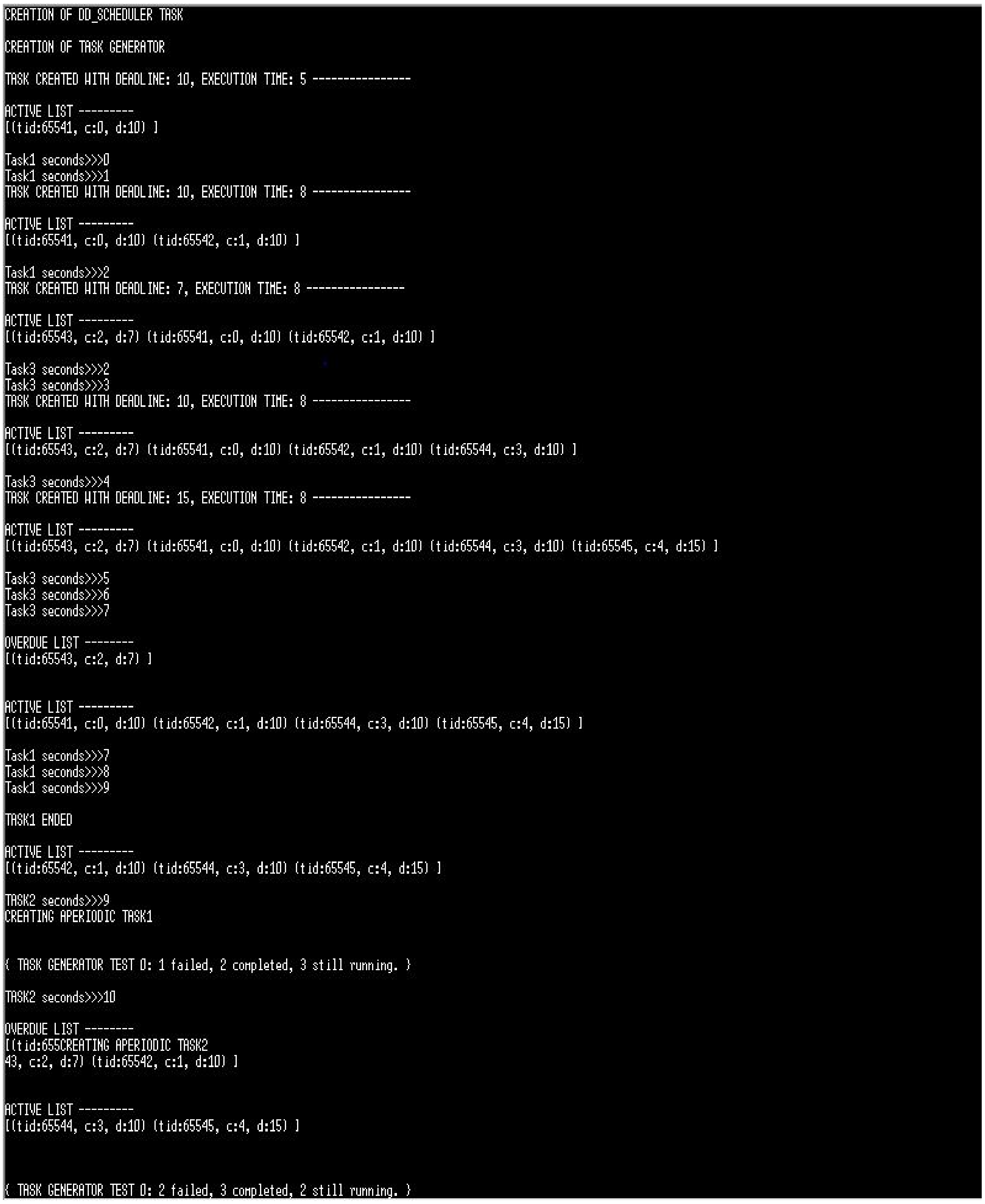


```
Couldnt get task Priority

Couldnt SET new_task Priority

APERIODIC TASK2 HAS ENDED
IDLE --seconds>>>21
CREATING APERIODIC TASK3
TASK CREATED WITH DEADLINE: 30, EXECUTION TIME: 10 ----------------

ACTIVE LIST ---------
[(tid:65542, c:21, d:30) ]


Couldnt get running task Priority

Couldnt SET running task Priority


{ TASK GENERATOR TEST 0: 1 failed, 6 completed, 1 still runnAingp. }e
r
 iodic Task3 seconds>>>21
Aperiodic Task3 seconds>>>22
Aperiodic Task3 seconds>>>23
Aperiodic Task3 seconds>>>24
Aperiodic Task3 seconds>>>25
Aperiodic Task3 seconds>>>26
Aperiodic Task3 seconds>>>27
Aperiodic Task3 seconds>>>28
Aperiodic Task3 seconds>>>29
Aperiodic Task3 seconds>>>30

OVERDUE LIST --------
[(tid:65542, c:7, d:18) (tid:65542, c:21, d:30) ]


ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority
IDLE --seconds>>>30
CREATING APERIODIC TASK4
TASK CREATED WITH DEADLINE: 36, EXECUTION TIME: 12 ----------------

ACTIVE LIST ---------
[(tid:65542, c:30, d:36) ]


Couldnt get running task Priority

Couldnt SET running task Priority


{ TASK GENERATOR TEST 0: 2 failed, 6 completed, 1 still runAninpg. e}
r
iodic Task4 seconds>>>30
Aperiodic Task4 seconds>>>31
Aperiodic Task4 seconds>>>32
Aperiodic Task4 seconds>>>33
Aperiodic Task4 seconds>>>34
Aperiodic Task4 seconds>>>35
Aperiodic Task4 seconds>>>36

OVERDUE LIST --------
[(tid:65542, c:7, d:18) (tid:65542, c:21, d:30) (tid:65542, c:30, d:36) ]


ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority
IDLE --seconds>>>36
IDLE --seconds>>>37
```

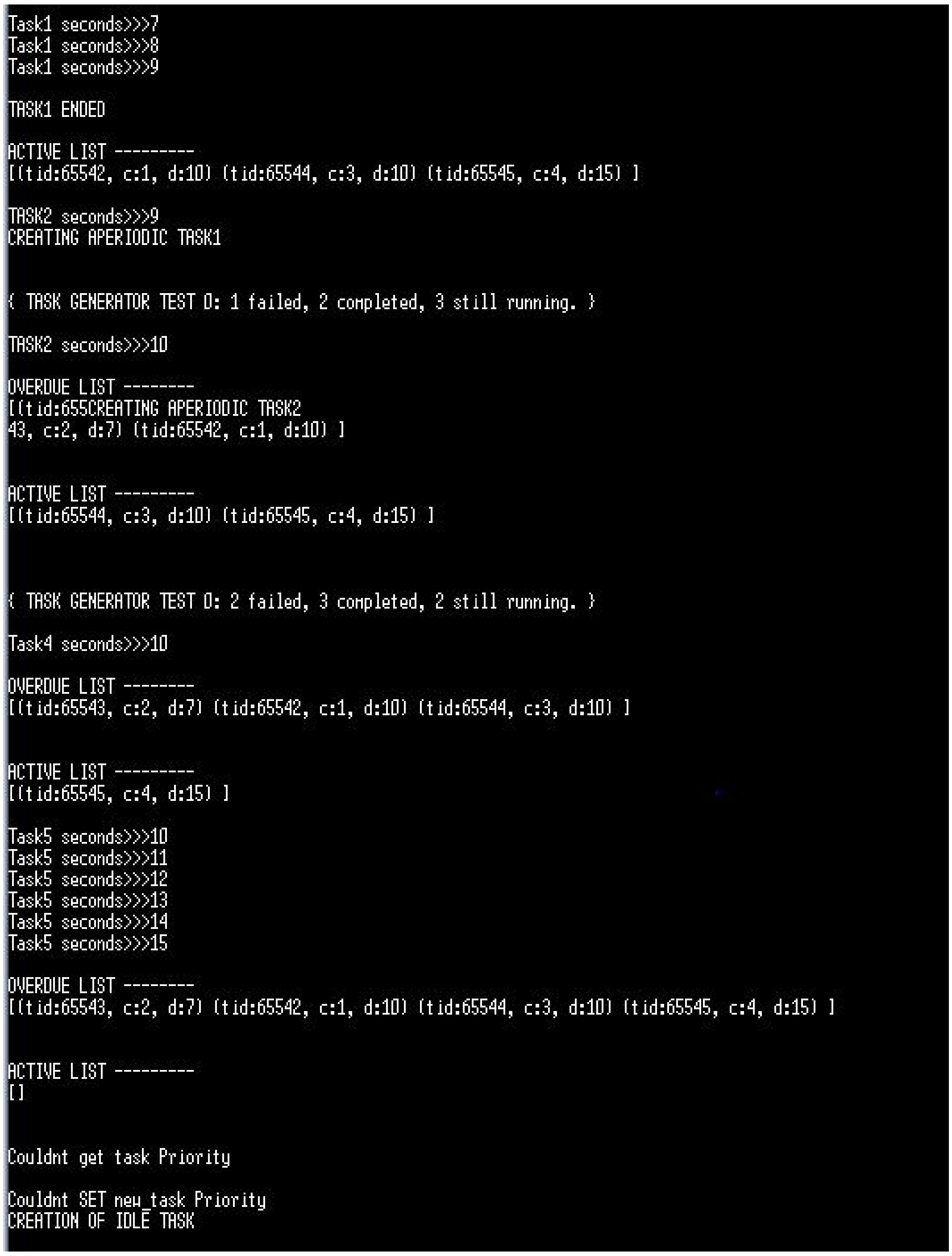


```
Couldnt get task Priority

Couldnt SET new_task Priority
IDLE --seconds>>>36
IDLE --seconds>>>37
IDLE --seconds>>>38
CREATING APERIODIC TASK5
TASK CREATED WITH DEADLINE: 45, EXECUTION TIME: 8 ----------------

ACTIVE LIST ---------
[(tid:65542, c:38, d:45) ]


Couldnt get running task Priority

Couldnt SET running task Priority


{ TASK GENERATOR TEST 0: 3 failed, 6 completed, 1 still runnAingp. }e
r
 Geineroatodr fiinicshe d -T---a---s---k---5--- ---s
e
conds>>>39
Aperiodic Task5 seconds>>>40
Aperiodic Task5 seconds>>>41
Aperiodic Task5 seconds>>>42
Aperiodic Task5 seconds>>>43
Aperiodic Task5 seconds>>>44
Aperiodic Task5 seconds>>>45

OVERDUE LIST --------
[(tid:65542, c:7, d:18) (tid:65542, c:21, d:30) (tid:65542, c:30, d:36) (tid:65542, c:38, d:45) ]


ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority


{ TASK GENERATOR TEST 1: 4 failed, 6 completed, 0 still running. }

total utilization time: 40
total idle time: 13
system time: 53
```

Periodic tasks

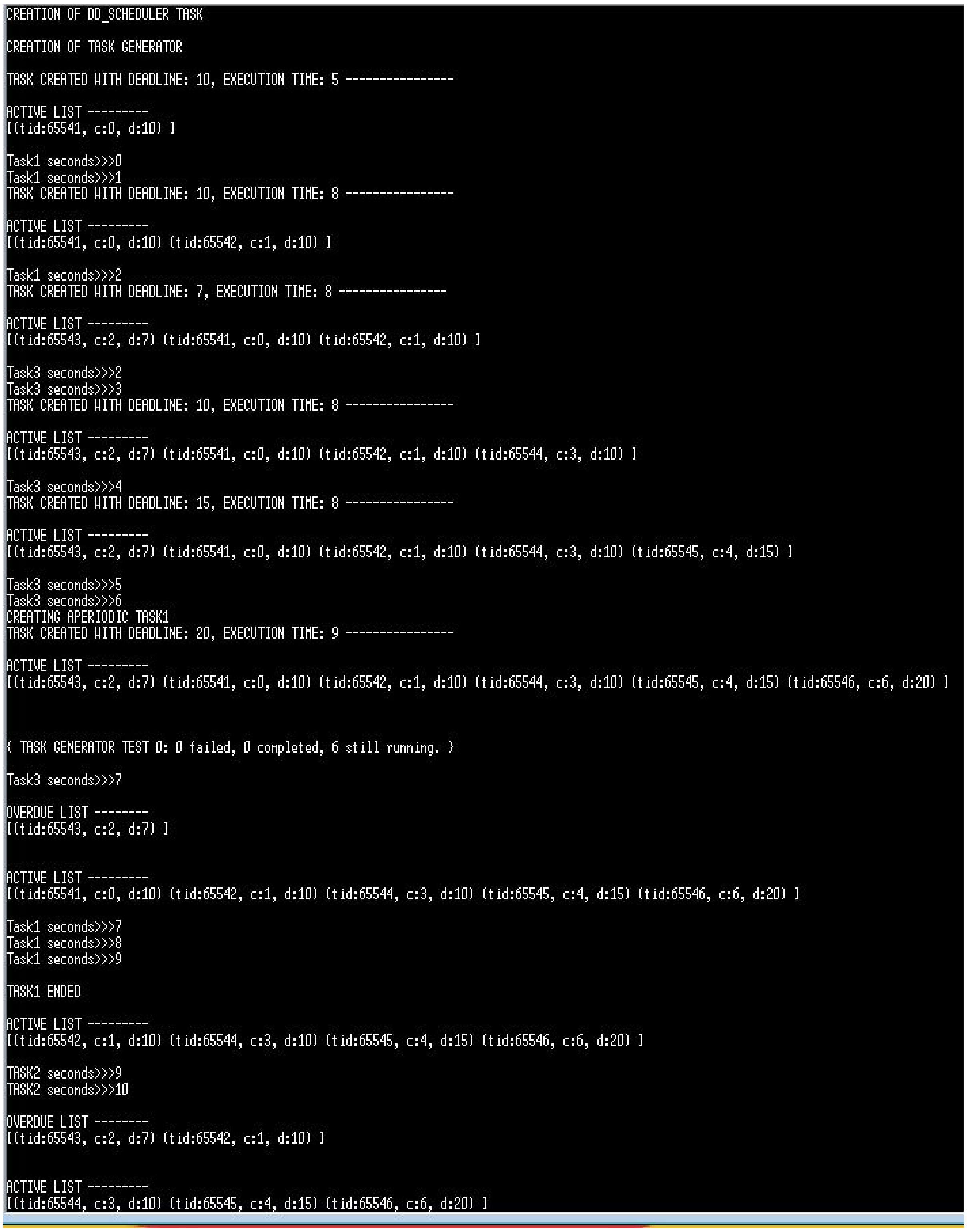


```
CREATION OF DD_SCHEDULER TASK

CREATION OF TASK GENERATOR

TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 5 ---------------

ACTIVE LIST ---------
[(tid:65541, c:0, d:10) ]

Task1 seconds>>>0
Task1 seconds>>>1
TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 8 ---------------

ACTIVE LIST ---------
[(tid:65541, c:0, d:10) (tid:65542, c:1, d:10) ]

Task1 seconds>>>2
TASK CREATED WITH DEADLINE: 7, EXECUTION TIME: 8 ---------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) ]

Task3 seconds>>>2
Task3 seconds>>>3
TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 8 ---------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) ]

Task3 seconds>>>4
TASK CREATED WITH DEADLINE: 15, EXECUTION TIME: 8 ---------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]

Task3 seconds>>>5
Task3 seconds>>>6
Task3 seconds>>>7

OVERDUE LIST --------
[(tid:65543, c:2, d:7) ]


ACTIVE LIST ---------
[(tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]

Task1 seconds>>>7
Task1 seconds>>>8
Task1 seconds>>>9

TASK1 ENDED

ACTIVE LIST ---------
[(tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]

TASK2 seconds>>>9
CREATING APERIODIC TASK1


{ TASK GENERATOR TEST 0: 1 failed, 2 completed, 3 still running. }

TASK2 seconds>>>10

OVERDUE LIST --------
[(tid:655CREATING APERIODIC TASK2
43, c:2, d:7) (tid:65542, c:1, d:10) ]


ACTIVE LIST ---------
[(tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]



{ TASK GENERATOR TEST 0: 2 failed, 3 completed, 2 still running. }
```

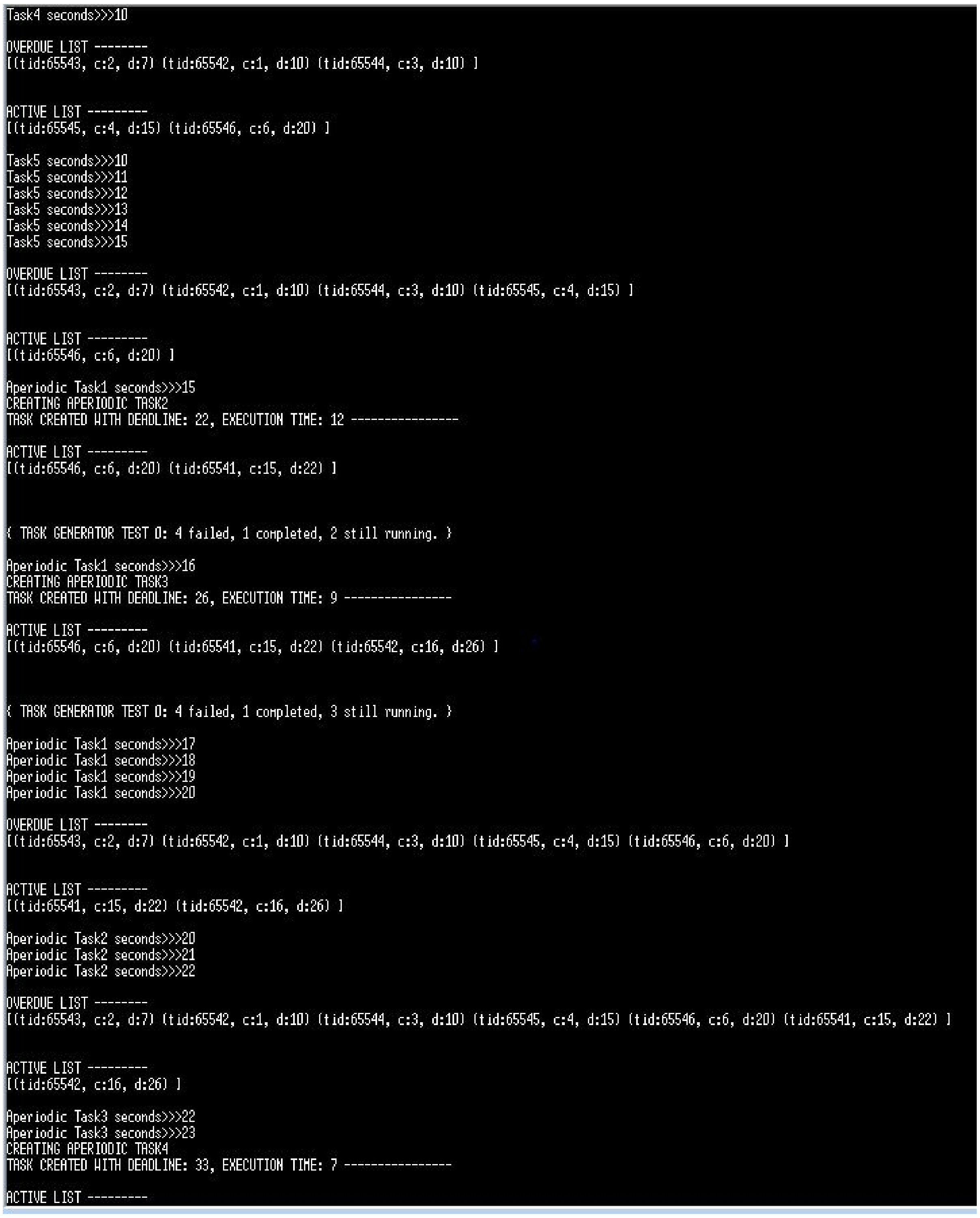


```
Task1 seconds>>>7
Task1 seconds>>>8
Task1 seconds>>>9

TASK1 ENDED

ACTIVE LIST ---------
[(tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]

TASK2 seconds>>>9
CREATING APERIODIC TASK1


{ TASK GENERATOR TEST 0: 1 failed, 2 completed, 3 still running. }

TASK2 seconds>>>10

OVERDUE LIST --------
[(tid:655CREATING APERIODIC TASK2
43, c:2, d:7) (tid:65542, c:1, d:10) ]


ACTIVE LIST ---------
[(tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]



{ TASK GENERATOR TEST 0: 2 failed, 3 completed, 2 still running. }

Task4 seconds>>>10

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) ]


ACTIVE LIST ---------
[(tid:65545, c:4, d:15) ]

Task5 seconds>>>10
Task5 seconds>>>11
Task5 seconds>>>12
Task5 seconds>>>13
Task5 seconds>>>14
Task5 seconds>>>15

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]


ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority
CREATION OF IDLE TASK
```

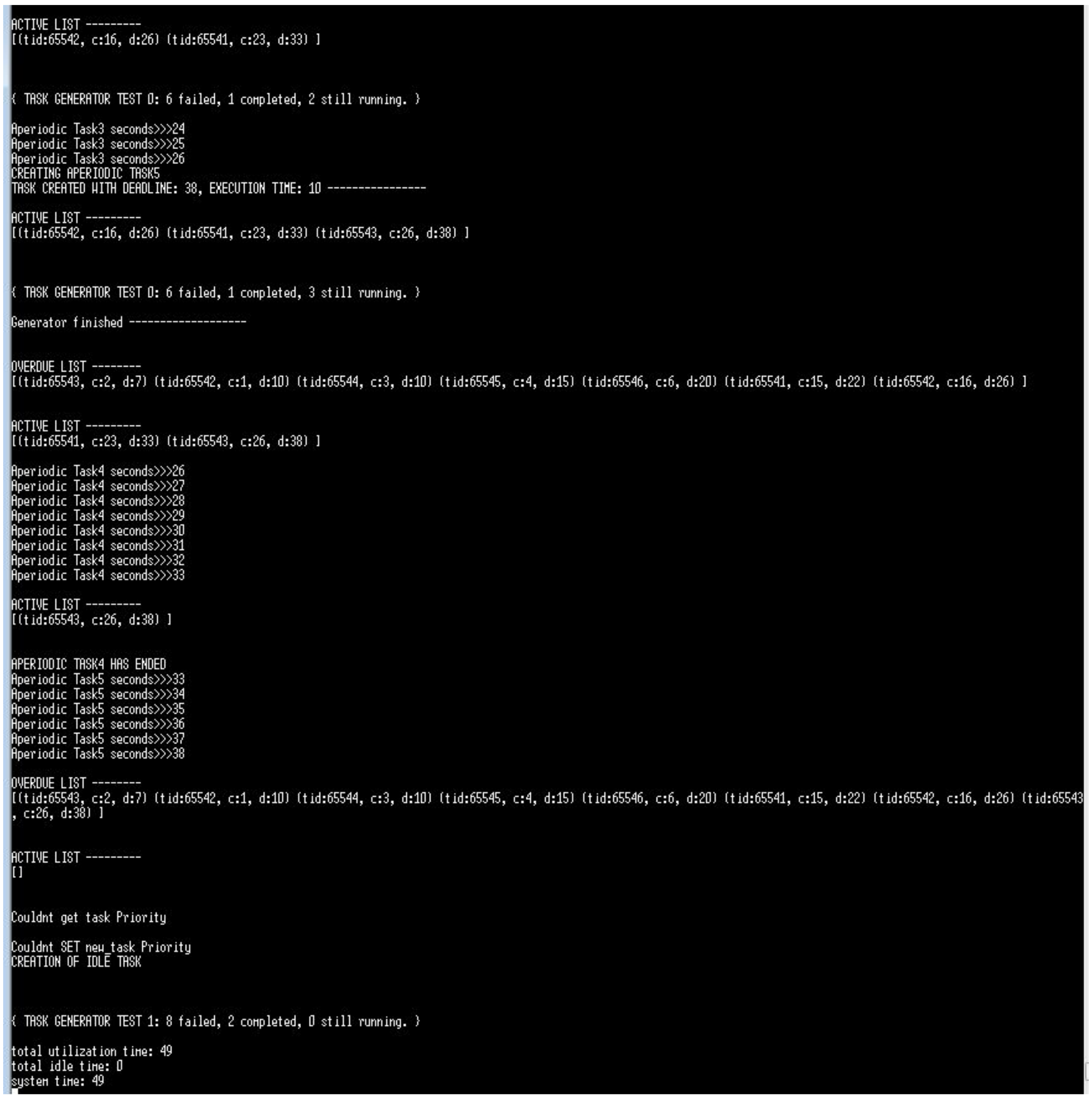
Periodic and aperiodic tasks



```
CREATION OF DD_SCHEDULER TASK

CREATION OF TASK GENERATOR

TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 5 ----------------

ACTIVE LIST ---------
[(tid:65541, c:0, d:10) ]

Task1 seconds>>>0
Task1 seconds>>>1
TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 8 ----------------

ACTIVE LIST ---------
[(tid:65541, c:0, d:10) (tid:65542, c:1, d:10) ]

Task1 seconds>>>2
TASK CREATED WITH DEADLINE: 7, EXECUTION TIME: 8 ----------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) ]

Task3 seconds>>>2
Task3 seconds>>>3
TASK CREATED WITH DEADLINE: 10, EXECUTION TIME: 8 ----------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) ]

Task3 seconds>>>4
TASK CREATED WITH DEADLINE: 15, EXECUTION TIME: 8 ----------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]

Task3 seconds>>>5
Task3 seconds>>>6
CREATING APERIODIC TASK1
TASK CREATED WITH DEADLINE: 20, EXECUTION TIME: 9 ----------------

ACTIVE LIST ---------
[(tid:65543, c:2, d:7) (tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]


( TASK GENERATOR TEST 0: 0 failed, 0 completed, 6 still running. )

Task3 seconds>>>7

OVERDUE LIST --------
[(tid:65543, c:2, d:7) ]


ACTIVE LIST ---------
[(tid:65541, c:0, d:10) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]

Task1 seconds>>>7
Task1 seconds>>>8
Task1 seconds>>>9

TASK1 ENDED

ACTIVE LIST ---------
[(tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]

TASK2 seconds>>>9
TASK2 seconds>>>10

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) ]


ACTIVE LIST ---------
[(tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]
```



```
Task4 seconds>>>10

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) ]


ACTIVE LIST ---------
[(tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]

Task5 seconds>>>10
Task5 seconds>>>11
Task5 seconds>>>12
Task5 seconds>>>13
Task5 seconds>>>14
Task5 seconds>>>15

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) ]


ACTIVE LIST ---------
[(tid:65546, c:6, d:20) ]

Aperiodic Task1 seconds>>>15
CREATING APERIODIC TASK2
TASK CREATED WITH DEADLINE: 22, EXECUTION TIME: 12 ----------------

ACTIVE LIST ---------
[(tid:65546, c:6, d:20) (tid:65541, c:15, d:22) ]



{ TASK GENERATOR TEST 0: 4 failed, 1 completed, 2 still running. }

Aperiodic Task1 seconds>>>16
CREATING APERIODIC TASK3
TASK CREATED WITH DEADLINE: 26, EXECUTION TIME: 9 ----------------

ACTIVE LIST ---------
[(tid:65546, c:6, d:20) (tid:65541, c:15, d:22) (tid:65542, c:16, d:26) ]



{ TASK GENERATOR TEST 0: 4 failed, 1 completed, 3 still running. }

Aperiodic Task1 seconds>>>17
Aperiodic Task1 seconds>>>18
Aperiodic Task1 seconds>>>19
Aperiodic Task1 seconds>>>20

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) ]


ACTIVE LIST ---------
[(tid:65541, c:15, d:22) (tid:65542, c:16, d:26) ]

Aperiodic Task2 seconds>>>20
Aperiodic Task2 seconds>>>21
Aperiodic Task2 seconds>>>22

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) (tid:65541, c:15, d:22) ]


ACTIVE LIST ---------
[(tid:65542, c:16, d:26) ]

Aperiodic Task3 seconds>>>22
Aperiodic Task3 seconds>>>23
CREATING APERIODIC TASK4
TASK CREATED WITH DEADLINE: 33, EXECUTION TIME: 7 ----------------

ACTIVE LIST ---------
```



```
ACTIVE LIST ---------
[(tid:65542, c:16, d:26) (tid:65541, c:23, d:33) ]


{ TASK GENERATOR TEST 0: 6 failed, 1 completed, 2 still running. }

Aperiodic Task3 seconds>>>24
Aperiodic Task3 seconds>>>25
Aperiodic Task3 seconds>>>26
CREATING APERIODIC TASK5
TASK CREATED WITH DEADLINE: 38, EXECUTION TIME: 10 ----------------

ACTIVE LIST ---------
[(tid:65542, c:16, d:26) (tid:65541, c:23, d:33) (tid:65543, c:26, d:38) ]


{ TASK GENERATOR TEST 0: 6 failed, 1 completed, 3 still running. }

Generator finished -------------------


OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) (tid:65541, c:15, d:22) (tid:65542, c:16, d:26) ]


ACTIVE LIST ---------
[(tid:65541, c:23, d:33) (tid:65543, c:26, d:38) ]

Aperiodic Task4 seconds>>>26
Aperiodic Task4 seconds>>>27
Aperiodic Task4 seconds>>>28
Aperiodic Task4 seconds>>>29
Aperiodic Task4 seconds>>>30
Aperiodic Task4 seconds>>>31
Aperiodic Task4 seconds>>>32
Aperiodic Task4 seconds>>>33

ACTIVE LIST ---------
[(tid:65543, c:26, d:38) ]


APERIODIC TASK4 HAS ENDED
Aperiodic Task5 seconds>>>33
Aperiodic Task5 seconds>>>34
Aperiodic Task5 seconds>>>35
Aperiodic Task5 seconds>>>36
Aperiodic Task5 seconds>>>37
Aperiodic Task5 seconds>>>38

OVERDUE LIST --------
[(tid:65543, c:2, d:7) (tid:65542, c:1, d:10) (tid:65544, c:3, d:10) (tid:65545, c:4, d:15) (tid:65546, c:6, d:20) (tid:65541, c:15, d:22) (tid:65542, c:16, d:26) (tid:65543
, c:26, d:38) ]


ACTIVE LIST ---------
[]


Couldnt get task Priority

Couldnt SET new_task Priority
CREATION OF IDLE TASK



{ TASK GENERATOR TEST 1: 8 failed, 2 completed, 0 still running. }

total utilization time: 49
total idle time: 0
system time: 49
```

Problem Areas

While programming the functionality of the system, we encountered several issues. One of them had to do with the tasks not having the correct priority at run-time. This meant that our entire system didn't work properly. We were able to fix this issue by using the MQX functions get_task_priority() and set_task_priority(). These functions were placed at the beginning of the auto initialized tasks. Based on our system design, we used the system time to measure the timing of the tasks and also to account for their deadlines. While this worked for the project, we believe that this is an edge case that could fail if the system time was a very large number. Because we stored the system time in an (int) data type, there is a very good possibility that with a very large system time (which could occur if the system were left running for days), the value stored in the int could overflow and thus the entire system would become invalid. A way to fix this issue would be to perform timing of the tasks using timers, and have these timers reset and every scheduled task.

Summary and Conclusion

In conclusion, the project was a success i regards to the specifications. We were able to accomplish the project requirements which meant creating a scheduler that can schedule user tasks based on a earliest deadline first algorithm. The system is also able to handle both periodic and aperiodic tasks. The only issue not accounted for was with tracking the execution times of the tasks using the system time, which could overflow if the system time is very very big.

References

[1] *Ceng 455 Lab Manual 2017*, University of Victoria., Victoria., BC, 2017, pp. 14-33.

Appendix

Code files were emailed to the TA.