Peer-graded Assignment: Assignment 5

Submit by November 5, 11:59 PM PST

Important Information
It is especially important to submit this assignment before the deadline, November 5, 11:59 PM PST, because it must be graded by others. If you submit late, there may not be enough classmates around to review your work. This makes it difficult - and in some cases, impossible - to produce a grade. Submit on time to avoid these risks.

Development of Real-Time Systems – Assignment 5

Overview

In the last assignment you will **implement your own scheduler in SimSo**. This is done by adding a new multi-core scheduler plug-in as a python file and integrating this into SimSo. We will learn that it is possible to implement any kind of scheduler in this environment which can be useful for your future research! Then we will also **implement inter-task communication in practice** in FreeRTOS by using message queues.

Pre-requisite

A normal PC capable of executing Python programs

It is recommended to use the free software Visual Studio Express for this assignment. The FreeRTOS system you are about to setup will execute in Visual Studio Express, and output from the system is available from its console. More help Here about setting up Visual Studio Express with the FreeRTOS project.

It is also possible to use the free software Eclipse for this assignment. Notes on setting up Eclipse with the FreeRTOS project Here.

After completing this assignment, you will be able to:

- Implement a custom multi-core or single-core scheduler in SimSo
- Implement inter-task communication in FreeRTOS

Simulation assignment

The assignment is to modify a real-time simulator to verify feasibility of a set of tasks.

-Download the SimSo scheduler Here!

-The example has been run under Windows 7 but other platforms are also supported by the scheduler.

-Install SimSo and familiarize yourself with the tool. More information is found in This document.

-Modify the python code in P_RM.py to use firstfit instead of the current algorithm. Please follow the steps in This document.

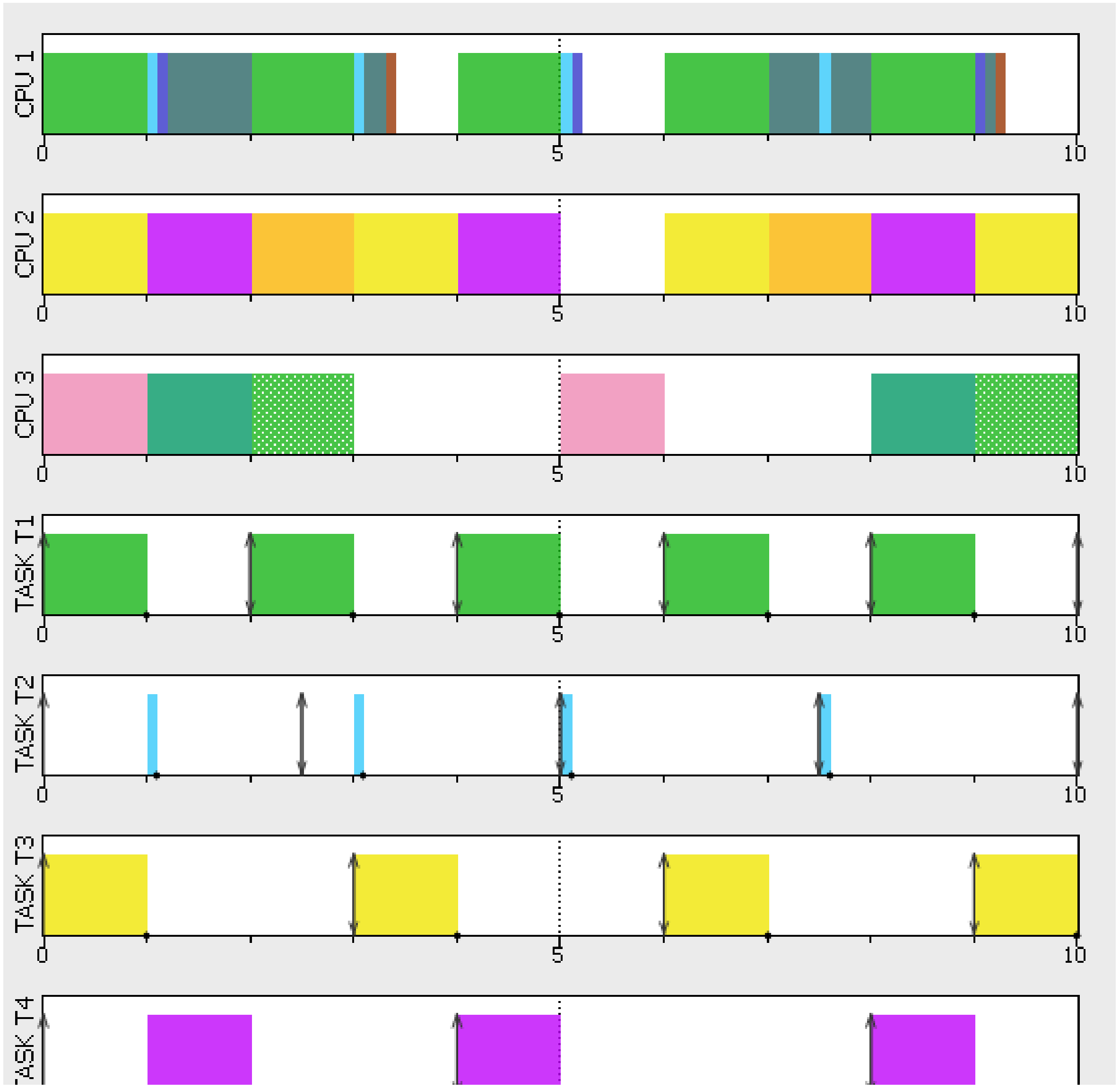
-Hint: Instead of scheduling the task to the CPU with the lowest utilization chose the first one which has a lower utilization than Urm(x+1) where x is the already scheduled tasks on the CPU

-Hint2: have a look at the def packer(self) function in the file P_RM.py

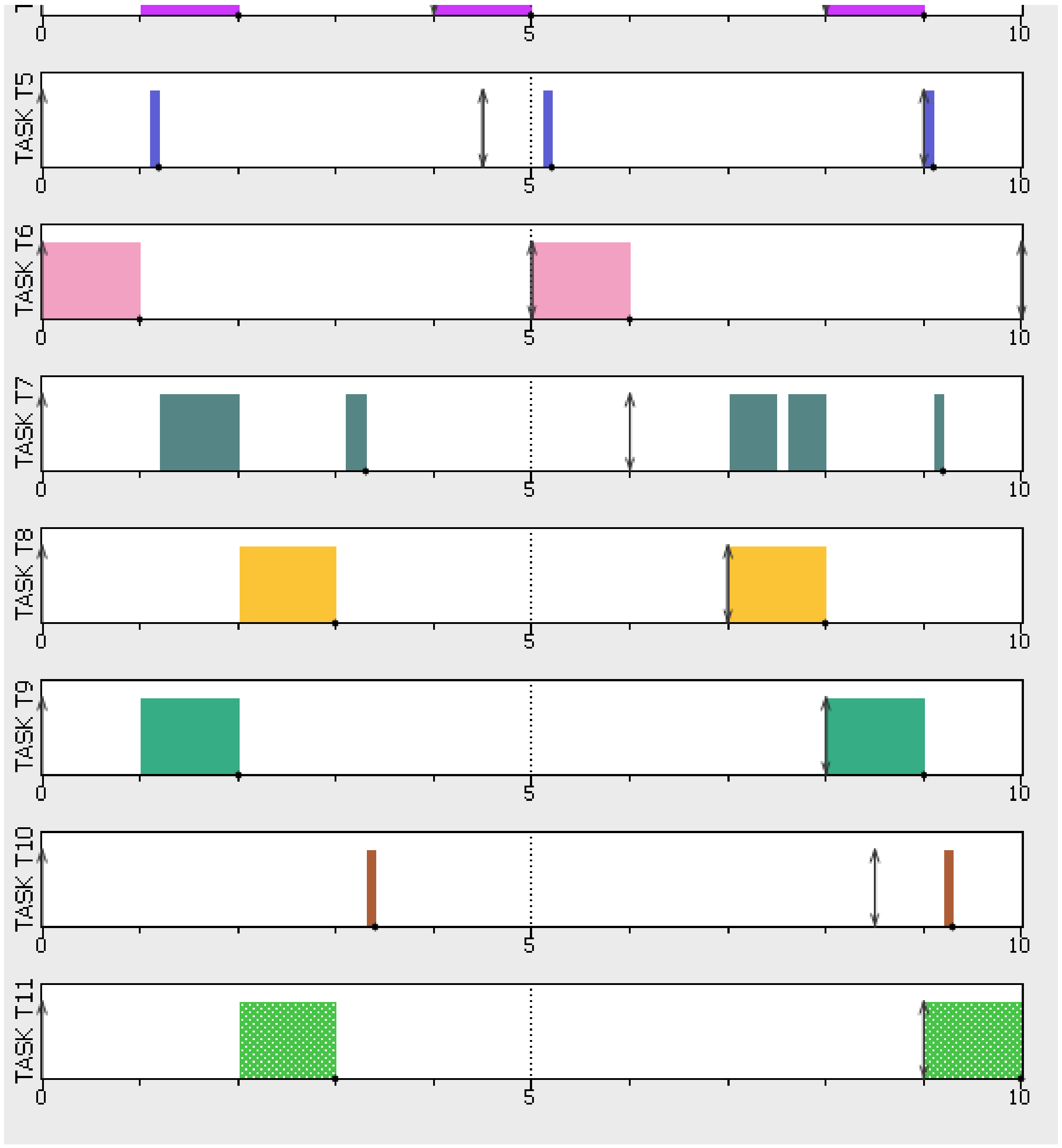
-Schedule the following task set on three processors using your modified algorithm.

T1(2,1) T2(2.5,0.1) T3(3,1) T4(4,1) T5(4.5,0.1) T6(5,1) T7(6,1) T8(7,1) T9(8,1) T10(8.5,0.1) T11(9,1)

-The schedule should look like the following image:



Help Center



The following should be included in a written report:

- A picture of the final schedule using the modified scheduler

Help Center

- The source code of the scheduling algorithm

Programming assignment

-Download the FreeRTOS project Here

-Import the project into Visual Studio Express 2015. More information from our Documentation.

-In Visual Studio click the “build” menu and “Build solution”. This will compile the project. More information from our Documentation.

-Now run the project. More information from our Documentation

-Familiarize yourself with the FreeRTOS API and locate to the main() function in the FreeRTOS project in Visual Studio Express

-Create a task "matrixtask" containing the following functionality:

```c
#define SIZE 10
#define ROW SIZE
#define COL SIZE
static void matrix_task()
{
  int i;
  double **a = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) a[i] = (double *)pvPortMalloc(COL * sizeof(double));
  double **b = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) b[i] = (double *)pvPortMalloc(COL * sizeof(double));
  double **c = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) c[i] = (double *)pvPortMalloc(COL * sizeof(double));

  double sum = 0.0;
  int j, k, l;

  for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
      a[i][j] = 1.5;
      b[i][j] = 2.6;
    }
  }

  while (1) {
    /*
    * In an embedded systems, matrix multiplication would block the CPU for a long time
    * but since this is a PC simulator we must add one additional dummy delay.
    */
    long simulationdelay;
    for (simulationdelay = 0; simulationdelay<1000000000; simulationdelay++)
      ;
    for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
        c[i][j] = 0.0;
      }
    }

    for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
        sum = 0.0;
        for (k = 0; k < SIZE; k++) {
          for (l = 0; l<10; l++) {
                sum = sum + a[i][k] * b[k][j];
            }
        }
        c[i][j] = sum;
      }
    }
    vTaskDelay(100);
  }
}
```

-Create a queue and send the content of (double **)c to the queue in matrixtask with before the vTaskDelay() call (hint: place the c variable in a struct). (More information Here).

-Create a reader task which reads the content of the queue in case there is something in the queue.

-In case the queue has some content it should save the data in a local (double **) variable.

-Print out the content of the (double **)c variable in case the content is updated. The data transferred from c should be a 10x10 matrix with the value 390 in each slot.

The following should be provided in a written report:

- A screenshot of the execution

Review criteria

less

Everyone enrolled in this course must review at least three other submissions to ensure everyone receives a grade.

Help Center

Help Center