HO CHI MINH UNIVERSITY OF TECHNOLOGY

EMBEDDED SYSTEM

LAB 4 REPORT

Author:
Tran Nhut Quang

Teacher:
Dr. Pham Hoang Anh

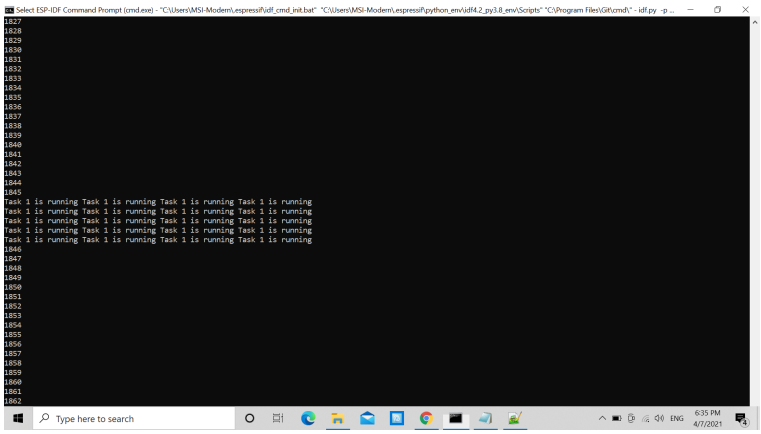
Apr 7, 2021

1 Exercise

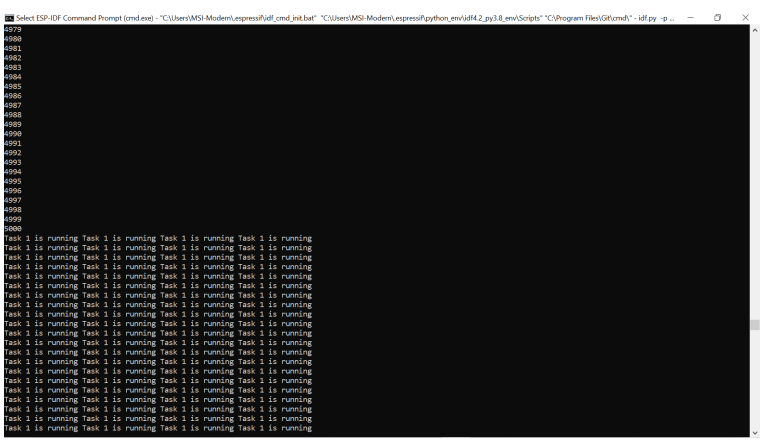
2 Prioritized Pre-emptive Scheduling with Time Slicing

Despite the Time Slicing is the default setting, I do not know why I could not implement it.

3 Prioritized Pre-emptive Scheduling (without Time Slicing)



4 Co-operative Scheduling



5 Impletation

```c
#include <stdio.h>
#include <time.h>
#include "sdkconfig.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/FreeRTOSconfig.h"

#include "esp_system.h"
#include "esp_spi_flash.h"

#if CONFIG_FREERTOS_UNICORE
  static const BaseType_t app_cpu = 0;
#else
  static const BaseType_t app_cpu = 1;
#endif

void vTaskPreempty(void* pvParameters)
{
  char * pcTaskName;
  const TickType_t xDelay250ms = pdMS_TO_TICKS(250);

  /* The string to print out is passed in via the parameter .
  Cast this to a character pointer . */
  pcTaskName = ( char * ) pvParameters;
  /* As per most tasks , this task is implemented in
  an infinite loop . */
  for ( ;; )
  {
    /* Print out the name of this task . */
    for (int i=0; i<5; i++) {
      printf(pcTaskName);
    }
    //printf(" ---- Preempty Task Begin: %ld\n", clock());
    /* Delay for a period . This time a call to vTaskDelay ()
    is used which places the task into the Blocked state
    until the delay period has expired . The parameter takes
    a time specified in " ticks " , and the pdMS_TO_TICKS () macro
    is used ( where the xDelay250ms constant is declared ) to
    convert 250 milliseconds into an equivalent time in ticks .*/
    //printf(" --- Preempty Task End: %ld\n", clock());
    vTaskDelay (250 / portTICK_PERIOD_MS);
  }
}

void vTaskFunction(void* pvParameters)
{
  while (1) {
    /*clock_t begin = clock();
    printf("Long Task Begin: %ld\n", begin);
    while (clock() < begin+2000) {
    }
    printf("Long Task End: %ld\n", clock());*/
    for (int i = 0; i <= 5000; i++) {
      printf("%d\n", i);
    }
    vTaskDelay (2000 / portTICK_PERIOD_MS);
  }
}

```

```c
void vIdleTask(void* pvParameters) {
  while (1) {
    printf("IDLE IDLE IDLE IDLE IDLE IDLE IDLE IDLE IDLE IDLE\n");
  }
}

static const char * pcTextForTask1 = "Task 1 is running Task 1 is running Task 1 is running Task 1 is running\r\n";
static const char * pcTextForTask2 = "Task 2 is running \r\n";

void app_main(void)
{
  printf("TIME SLICING: %d\n", configUSE_TIME_SLICING);
  printf("PREEMPTION: %d\n", configUSE_PREEMPTION);

  xTaskCreatePinnedToCore(vTaskPreempty, "Task PreEmpty", 2048, (void*)pcTextForTask1, 2, NULL, app_cpu);
  xTaskCreatePinnedToCore(vTaskFunction, "Task 1", 2048, NULL, 1, NULL, app_cpu);
  //xTaskCreate(vIdleTask, "Task Idle", 2048, (void*)pcTextForTask2, 1, app_cpu);
  //vTaskStartScheduler();

}
```