A cartoon of a person sitting on a chair

Description automatically generatedA white circle with red and black text

Description automatically generated**Department of Electronics and Communications Engineering**

**Advanced Embedded Systems**

**ELC4030**

Cairo University

Faculty of Engineering

Embedded ASSIGNMENT 1

|  |  |  |  |
| --- | --- | --- | --- |
| BN | ID | SEC |  |
| 4 | ***9210023*** | 1 | **أحمد اسامه سعد ياسين** |

Under supervision of: Dr. Hany El-Sayed

# **Rate-Monotonic Scheduling**

## **Part 1**

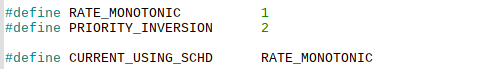
Task 1 (100ms) has the highest priority

Task 2 (200ms) has the medium priority

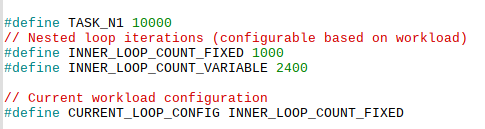
Task 3 (300ms) has the lowest priority

Setting Tasks outer loop = 10000 and inner loop = 1000, So, The Multiplication of them equals as required.

First, CURRENT\_USING\_SCHD has been set into RATE\_MONOTONIC.



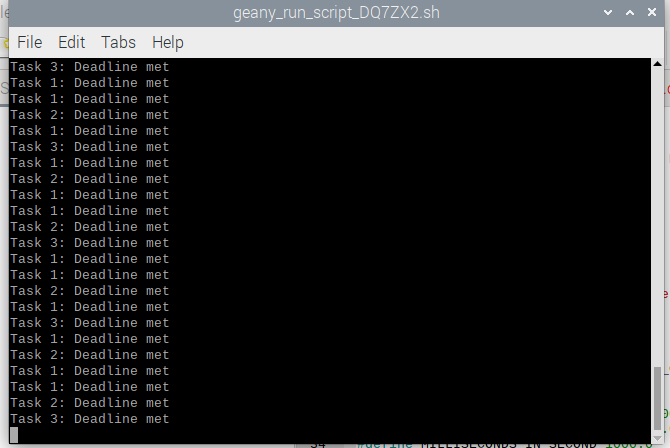
TASK\_N1 defines as 10000, and CURRENT\_LOOP\_CONFIG defines as INNER\_LOOP\_COUNT\_FIXED defines as either INNER\_LOOP\_COUNT\_FIXED as 1000 and INNER\_LOOP\_COUNT\_VARIABLE as 2400



TASK\_N1 equals to outer loop in the for loop, and CURRENT\_LOOP\_CONFIG equals into inner for loop.



### **Results**

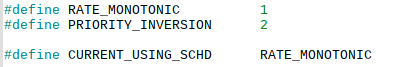


#### **Discussion**

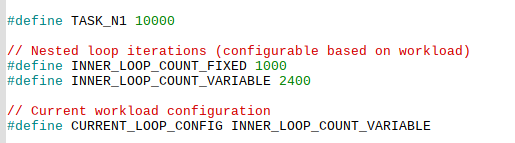
* All tasks met the deadlines as required successfully.

## **Part 2**

First, CURRENT\_USING\_SCHD has been set into RATE\_MONOTONIC.



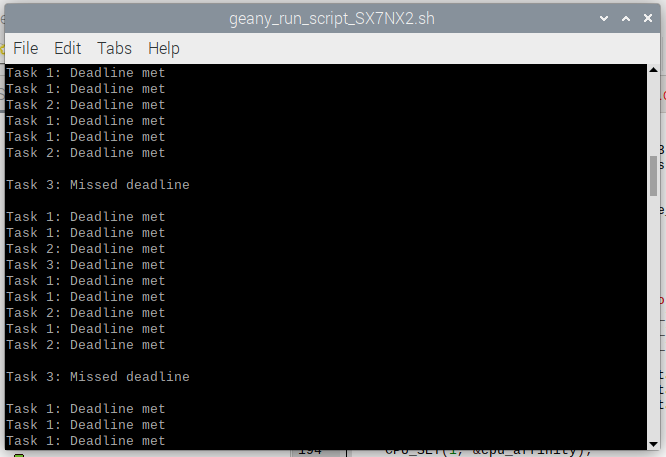
TASK\_N1 defines as 10000, and CURRENT\_LOOP\_CONFIG defines as INNER\_LOOP\_COUNT\_VARIABLE defines as either INNER\_LOOP\_COUNT\_FIXED as 2400



TASK\_N1 equals to outer loop in the for loop, and CURRENT\_LOOP\_CONFIG equals into inner for loop.



### **Results**



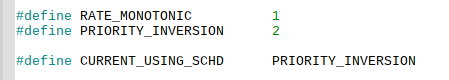
#### **Results Discussion**

* As we increase the execution time (increasing the counts of Nested Loops), Task 3 is the first task misses its deadline first as expected

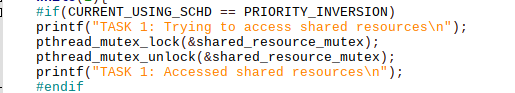
# **Priority Inversion**

## **Part 1**

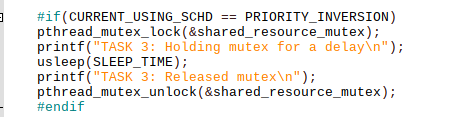
First, CURRENT\_USING\_SCHD has been set into PRIORITY\_INVERSION.



We start adding a mutex lock and unlock in the highest priority task (task 1)

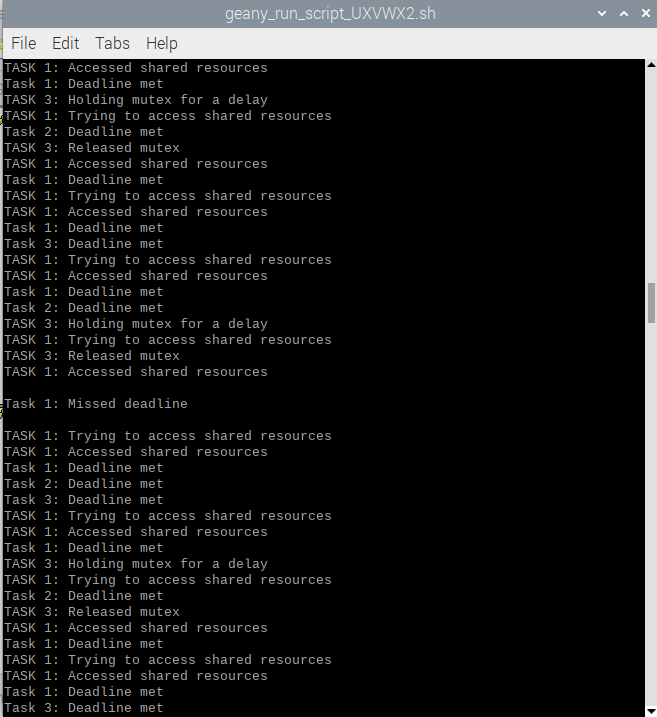


We added the mutex lock and sleep time then mutex unlock in the lowest priority task, and between them we let lowest priority task waits for a long delay 120ms so we use usleep(SLEEP\_TIME) 🡪 SLEEP\_TIME is defined as 120000





### **Results**



Single Core Sleep time 120ms – 10 million alliteration.

#### **Results Discussion**

* First all tasks are meeting their deadline but most of the time task 3 locks the mutex for a shared resource for 120ms and task 1 wants to access the shared resource at a time task 1 will miss its deadline if task 3 starts before task 1 and holds the mutex for more than task 1 period this will due to a deadline miss.

## **Part 2**

First, I will set the SLEEP\_TIME into half (60ms).

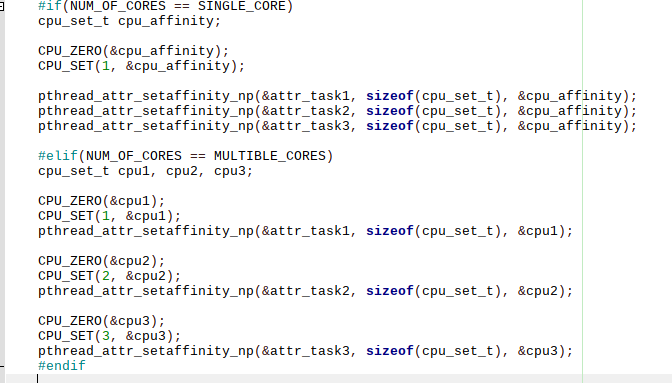


Second, I will set the SLEEP\_TIME into double (240ms).

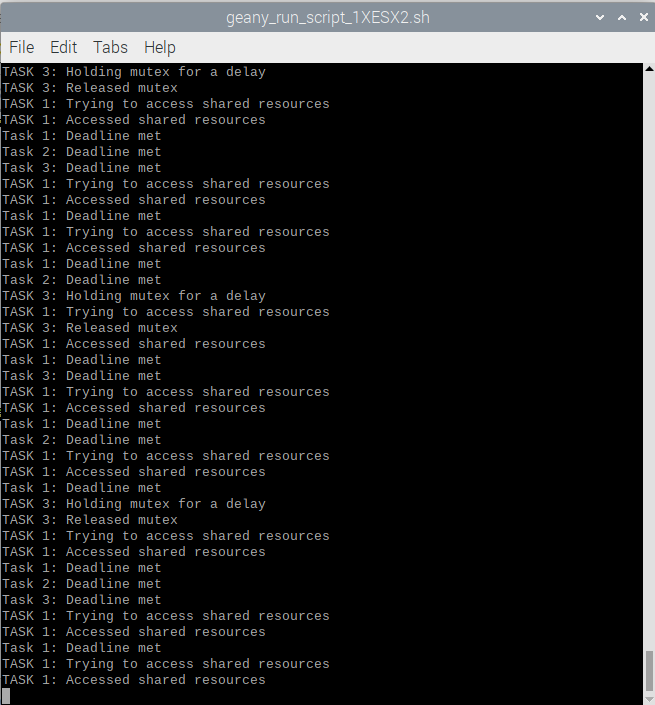


At the second part of this part, I allow the tasks to run on different cores.





### **Results 1**



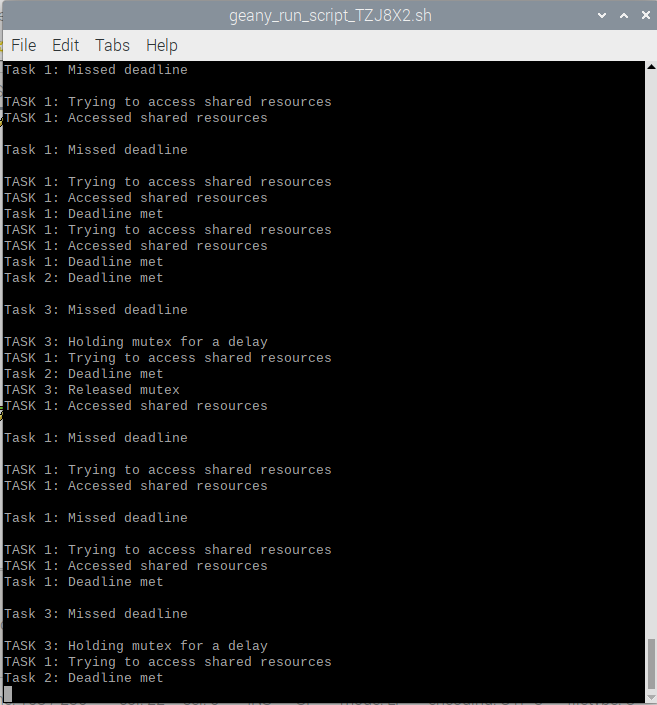
Single Core Sleep time is halved into 60ms – 10 million alliteration.

#### **Discussion**

* As the Sleep time decreases the less time lowest priority task taking while having the mutex so it’s kind rare that task 1 highest priority will miss its deadline.

**there’s no task misses its deadline, tasks met their deadlines.**

### **Result 2**

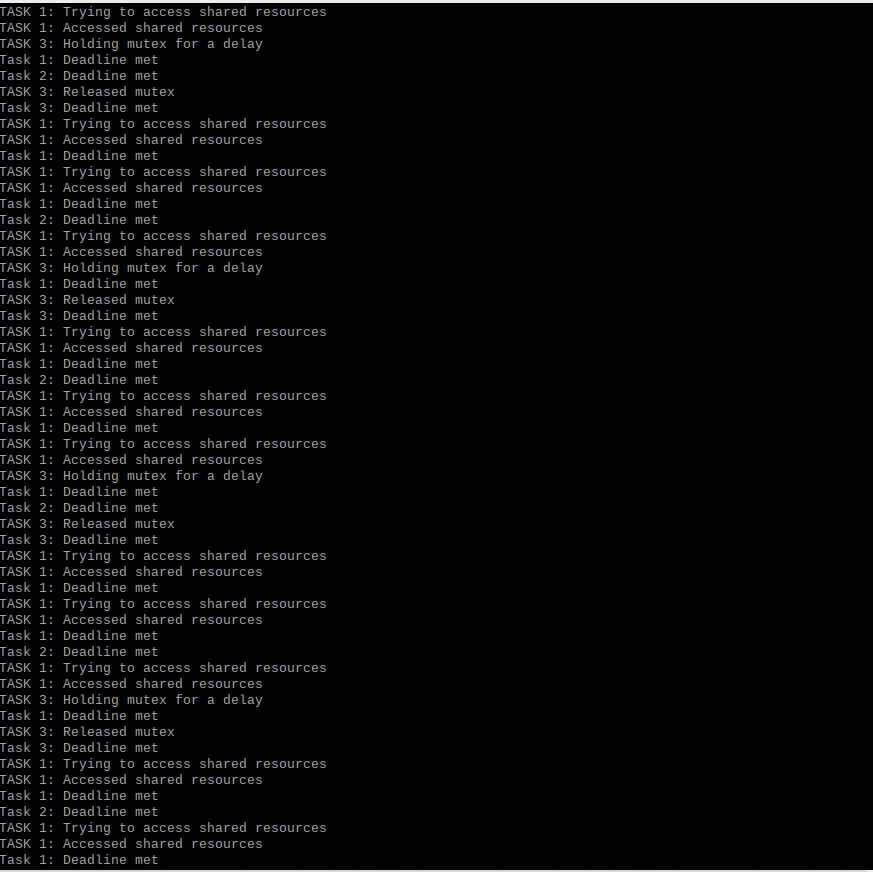


Single Core Sleep time is doubled into 240ms – 10 million alliteration.

#### **Discussion**

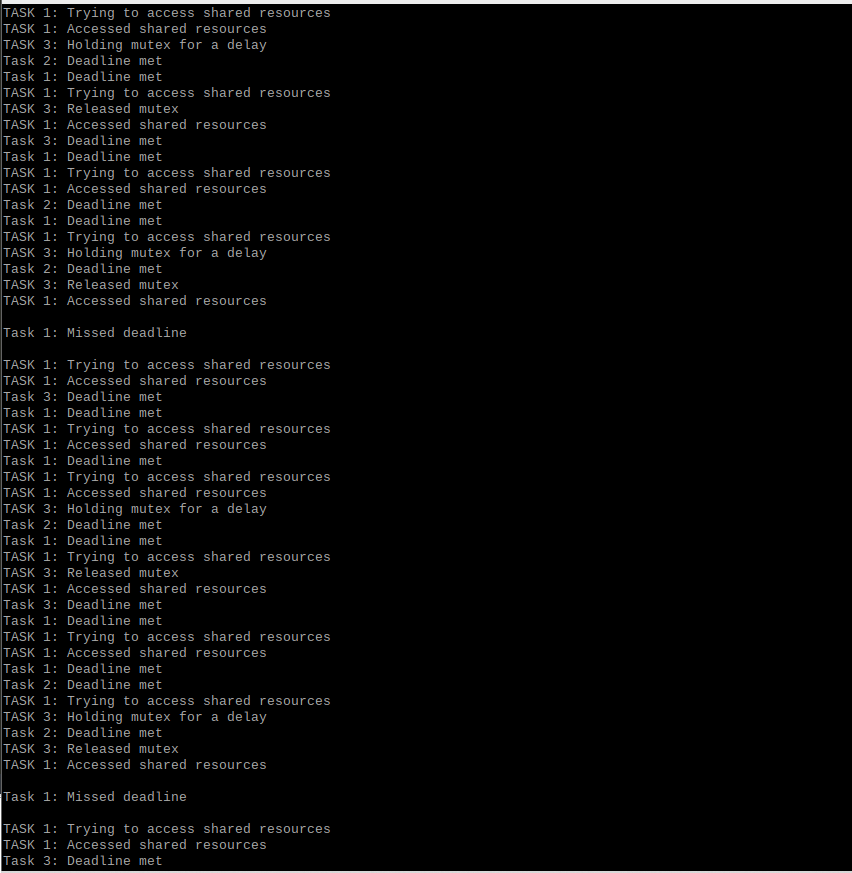
* As the Sleep time doubles the more time lowest priority task taking while having the mutex so it’s kind common that task 1 highest priority will miss its deadline. So, the other tasks might miss like task 3.

### **Results 3**



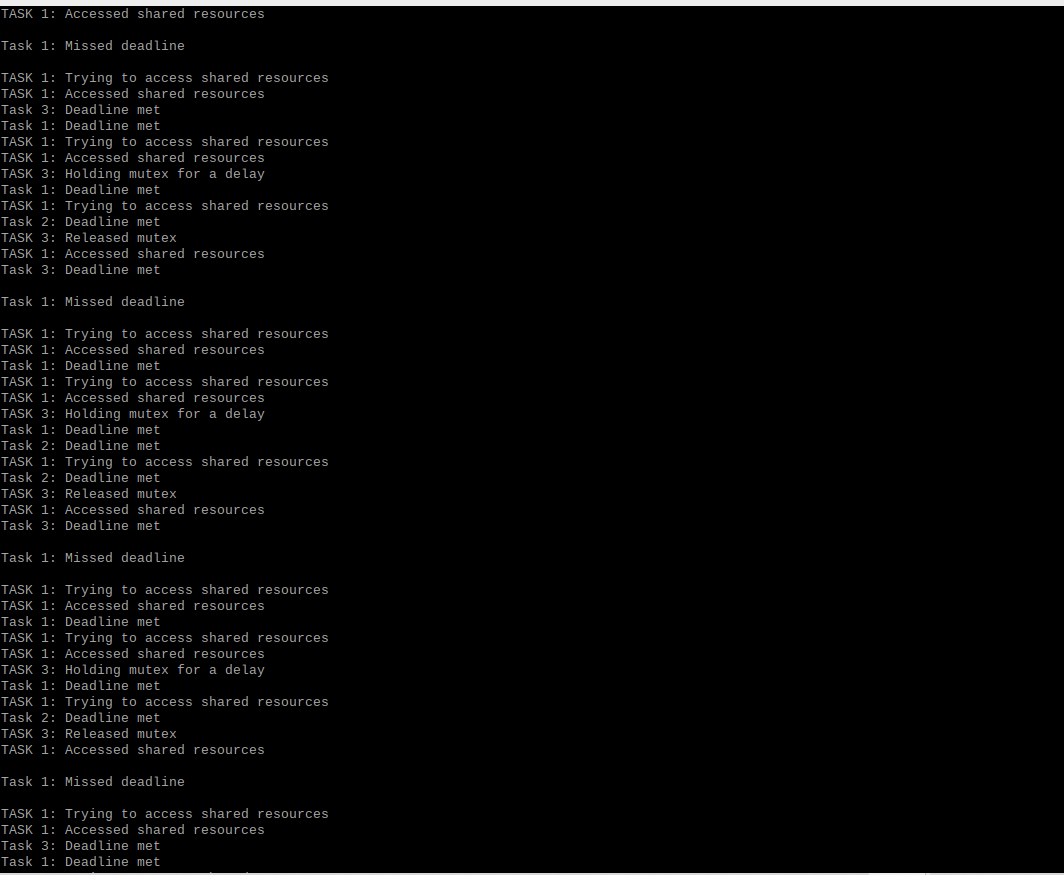
Multi cores – 60ms Sleep – 10 million alliteration.

* There is no difference between the signal core and multi-cores so we are not able to decide which one is the better one. But at this case multi cores it’s divided into three cores so the utilization decreases of each one.



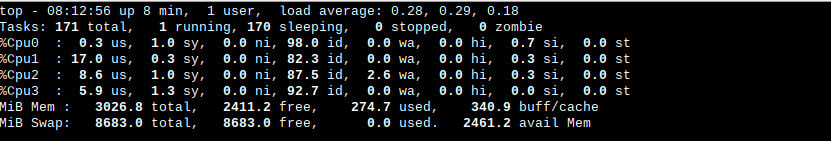
Multi cores – 120ms Sleep – 10 million alliteration.

* there is no big difference between the signal core and multi-cores so we are not able to decide which one is the better one. But at this case multi cores it’s divided into three cores so the utilization decreases of each one.



Multi cores – 240ms Sleep – 10 million alliteration.

* In this case multi cores is much better than single core because its not missing the task 3, because task 3 is running on another cpu so the shared resource is the only problem in this case not the cpu aswell.



CPU Load – Multi cores.

#### **Discussion**

* You can notice that the cpu1 has the highest CPU load because it’s highest priority and the most period of all tasks then task 2 medium priority in cpu2 and task 3 lowest priority as cp3 lowest CPU load.

# **The Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Author: Ahmed Osama Saad Yassin \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Includes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sched.h>

#include <time.h>

#include <math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Macros \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define RATE\_MONOTONIC 1

#define PRIORITY\_INVERSION 2

#define CURRENT\_USING\_SCHD RATE\_MONOTONIC

#define SINGLE\_CORE 1

#define MULTIBLE\_CORES 2

#define NUM\_OF\_CORES SINGLE\_CORE

#define TASK\_N1 10000

// Nested loop iterations (configurable based on workload)

#define INNER\_LOOP\_COUNT\_FIXED 1000

#define INNER\_LOOP\_COUNT\_VARIABLE 2400

// Current workload configuration

#define CURRENT\_LOOP\_CONFIG INNER\_LOOP\_COUNT\_FIXED

// Constants for time calculations

#define NANOSECONDS\_IN\_SECOND 1000000000.0

#define MICROSECONDS\_IN\_SECOND 1000000.0

#define MILLISECONDS\_IN\_SECOND 1000.0

// Task periodic intervals in microseconds

#define TASK1\_INTERVAL\_US 100000 // 100ms

#define TASK2\_INTERVAL\_US 200000 // 200ms

#define TASK3\_INTERVAL\_US 300000 // 300ms

// Task priorities

#define TASK1\_PRIORITY 3

#define TASK2\_PRIORITY 2

#define TASK3\_PRIORITY 1

#define MISSED\_TIME 1

#define SLEEP\_TIME 120000

pthread\_mutex\_t shared\_resource\_mutex;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function Prototypes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Utility functions for timespec manipulation

void timespec\_add\_us(struct timespec \*t, long us);

int timespec\_cmp(const struct timespec \*a, const struct timespec \*b);

double subtract\_timespecs(const struct timespec \*a, const struct timespec \*b);

// Task functions

void \*task1(void \*);

void \*task2(void \*);

void \*task3(void \*);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Utility Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Adds microseconds to a given timespec structure

void timespec\_add\_us(struct timespec \*t, long us) {

t->tv\_nsec += us \* 1000;

if (t->tv\_nsec >= NANOSECONDS\_IN\_SECOND) {

t->tv\_nsec -= NANOSECONDS\_IN\_SECOND;

t->tv\_sec += 1;

}

}

// Compare two timespec structures (-1: earlier, 0: equal, 1: later)

int timespec\_cmp(const struct timespec \*a, const struct timespec \*b){

if (a->tv\_sec > b->tv\_sec)

return 1;

else if (a->tv\_sec < b->tv\_sec)

return -1;

else if (a->tv\_nsec > b->tv\_nsec)

return 1;

else if (a->tv\_nsec < b->tv\_nsec)

return -1;

else

return 0;

}

// Calculate the difference in seconds between two timespec structures

double subtract\_timespecs(const struct timespec \*a, const struct timespec \*b){

double t1 = a->tv\_sec + (a->tv\_nsec/NANOSECONDS\_IN\_SECOND);

double t2 = b->tv\_sec + (b->tv\_nsec/NANOSECONDS\_IN\_SECOND);

if(t1>t2)

return t1-t2;

else

return t2-t1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Task Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Task 1: Periodic execution with resource locking

void \*task1(void \*args) {

struct timespec time\_current\_task, time\_next\_task = {0};

int \_\_attribute\_\_((unused)) a =0;

clock\_gettime(CLOCK\_REALTIME, &time\_next\_task);

while(1){

#if(CURRENT\_USING\_SCHD == PRIORITY\_INVERSION)

printf("TASK 1: Trying to access shared resources\n");

pthread\_mutex\_lock(&shared\_resource\_mutex);

pthread\_mutex\_unlock(&shared\_resource\_mutex);

printf("TASK 1: Accessed shared resources\n");

#endif

for (int i=0; i< TASK\_N1; i++) {

for (int j=0; j<CURRENT\_LOOP\_CONFIG; j++) a=j/2;

}

timespec\_add\_us(&time\_next\_task, (long)TASK1\_INTERVAL\_US);

clock\_gettime(CLOCK\_REALTIME, &time\_current\_task);

if (timespec\_cmp(&time\_current\_task, &time\_next\_task) == MISSED\_TIME)

printf("\nTask 1: Missed deadline\n\n");

else

printf("Task 1: Deadline met\n");

clock\_nanosleep(CLOCK\_REALTIME, TIMER\_ABSTIME,&time\_next\_task, NULL);

}

return NULL;

}

// Task 2: Periodic execution without resource locking

void \*task2(void \*args) {

struct timespec time\_current\_task, time\_next\_task = {0};

int \_\_attribute\_\_((unused)) a =0;

clock\_gettime(CLOCK\_REALTIME, &time\_next\_task);

while(1){

for (int i=0; i< TASK\_N1; i++) {

for (int j=0; j<CURRENT\_LOOP\_CONFIG; j++) a=j/2;

}

timespec\_add\_us(&time\_next\_task, (long)TASK2\_INTERVAL\_US);

clock\_gettime(CLOCK\_REALTIME, &time\_current\_task);

if (timespec\_cmp(&time\_current\_task, &time\_next\_task) == MISSED\_TIME)

printf("\nTask 2: Missed deadline\n\n");

else

printf("Task 2: Deadline met\n");

clock\_nanosleep(CLOCK\_REALTIME, TIMER\_ABSTIME,&time\_next\_task, NULL);

}

return NULL;

}

// Task 3: Periodic execution with resource locking and delay simulation

void \*task3(void \*args) {

struct timespec time\_current\_task, time\_next\_task = {0};

int \_\_attribute\_\_((unused)) a;

clock\_gettime(CLOCK\_REALTIME, &time\_next\_task);

while(1){

#if(CURRENT\_USING\_SCHD == PRIORITY\_INVERSION)

pthread\_mutex\_lock(&shared\_resource\_mutex);

printf("TASK 3: Holding mutex for a delay\n");

usleep(SLEEP\_TIME);

printf("TASK 3: Released mutex\n");

pthread\_mutex\_unlock(&shared\_resource\_mutex);

#endif

for (int i=0; i< TASK\_N1; i++) {

for (int j=0; j<CURRENT\_LOOP\_CONFIG; j++) a=j/2;

}

timespec\_add\_us(&time\_next\_task, (long)TASK3\_INTERVAL\_US);

clock\_gettime(CLOCK\_REALTIME, &time\_current\_task);

if (timespec\_cmp(&time\_current\_task, &time\_next\_task) == MISSED\_TIME)

printf("\nTask 3: Missed deadline\n\n");

else

printf("Task 3: Deadline met\n");

clock\_nanosleep(CLOCK\_REALTIME, TIMER\_ABSTIME,&time\_next\_task, NULL);

}

return NULL;

}

int main(){

pthread\_t thread1, thread2, thread3;

pthread\_attr\_t attr\_task1, attr\_task2, attr\_task3;

pthread\_mutex\_init(&shared\_resource\_mutex, NULL);

pthread\_attr\_init(&attr\_task1);

pthread\_attr\_init(&attr\_task2);

pthread\_attr\_init(&attr\_task3);

// Set scheduling policies and priorities

pthread\_attr\_setinheritsched(&attr\_task1, PTHREAD\_EXPLICIT\_SCHED);

pthread\_attr\_setinheritsched(&attr\_task2, PTHREAD\_EXPLICIT\_SCHED);

pthread\_attr\_setinheritsched(&attr\_task3, PTHREAD\_EXPLICIT\_SCHED);

pthread\_attr\_setschedpolicy(&attr\_task1, SCHED\_FIFO);

pthread\_attr\_setschedpolicy(&attr\_task2, SCHED\_FIFO);

pthread\_attr\_setschedpolicy(&attr\_task3, SCHED\_FIFO);

#if(NUM\_OF\_CORES == SINGLE\_CORE)

cpu\_set\_t cpu\_affinity;

CPU\_ZERO(&cpu\_affinity);

CPU\_SET(1, &cpu\_affinity);

pthread\_attr\_setaffinity\_np(&attr\_task1, sizeof(cpu\_set\_t), &cpu\_affinity);

pthread\_attr\_setaffinity\_np(&attr\_task2, sizeof(cpu\_set\_t), &cpu\_affinity);

pthread\_attr\_setaffinity\_np(&attr\_task3, sizeof(cpu\_set\_t), &cpu\_affinity);

#elif(NUM\_OF\_CORES == MULTIBLE\_CORES)

cpu\_set\_t cpu1, cpu2, cpu3;

CPU\_ZERO(&cpu1);

CPU\_SET(1, &cpu1);

pthread\_attr\_setaffinity\_np(&attr\_task1, sizeof(cpu\_set\_t), &cpu1);

CPU\_ZERO(&cpu2);

CPU\_SET(2, &cpu2);

pthread\_attr\_setaffinity\_np(&attr\_task2, sizeof(cpu\_set\_t), &cpu2);

CPU\_ZERO(&cpu3);

CPU\_SET(3, &cpu3);

pthread\_attr\_setaffinity\_np(&attr\_task3, sizeof(cpu\_set\_t), &cpu3);

#endif

// Set task priorities

struct sched\_param priority\_task1 = {.sched\_priority = TASK1\_PRIORITY};

struct sched\_param priority\_task2 = {.sched\_priority = TASK2\_PRIORITY};

struct sched\_param priority\_task3 = {.sched\_priority = TASK3\_PRIORITY};

pthread\_attr\_setschedparam(&attr\_task1, &priority\_task1);

pthread\_attr\_setschedparam(&attr\_task2, &priority\_task2);

pthread\_attr\_setschedparam(&attr\_task3, &priority\_task3);

// Create tasks

pthread\_create(&thread1, &attr\_task1, &task1, NULL);

pthread\_create(&thread2, &attr\_task2, &task2, NULL);

pthread\_create(&thread3, &attr\_task3, &task3, NULL);

pthread\_attr\_destroy(&attr\_task1);

pthread\_attr\_destroy(&attr\_task2);

pthread\_attr\_destroy(&attr\_task3);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

pthread\_join(thread3, NULL);

pthread\_mutex\_destroy(&shared\_resource\_mutex);

return 0;

}

# README FILE

## [GitHub Repo](https://github.com/ahmedoyassin/rtos-raspbrian-uni-proj) 🡪 [README.md](https://github.com/ahmedoyassin/rtos-raspbrian-uni-proj/blob/main/README.md)

NOTE: click on GitHub Repo to get into the github repo link. Or click on README.md to get into the readme file directly.