Embedded System Software Design

Project 2

Problem Definition:

By parallelizing some regions in a program, we can reduce the execution duration, but the data dependency might damage the parallelism. In this project, we will protect the sharing resource and synchronize thread in multithread mode, using different schedulers (FIFO, Round-Robin) in the Linux system.

Experimental Environment:

- ✓ PC: at least 4 cores
- ✓ RAM: at least 4GB
- ✓ OS: Ubuntu 16.04 or version above
- ✓ Compiler: G++ 5.4.0

POSIX Thread Mutex

The mutex object is locked by calling pthread_mutex_lock(). If the mutex is already locked, the calling thread shall be blocked until the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

```
#include <sched.h>

pthread_mutex_t count_mutex= PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_lock( &count_mutex );
pthread_mutex_unlock( &count_mutex );
```

POSIX Thread Barrier

POSIX threads specify a synchronization object called a barrier, along with barrier functions. The functions create the barrier, specifying the number of threads that are synchronizing on the barrier, and set up threads to perform tasks and wait at the barrier until all the threads reach the barrier. When the last thread arrives at the barrier, all the threads resume execution.

```
#include <sched.h>

pthread_barrier_t barr;
pthread_barrier_init(&barr, NULL, NUMBER_OF_CORE);
pthread_barrier_wait(&barr);
```

Scheduler setting

Linux supports several schedulers, such as FIFO and Round-Robin. We can use the function "sched_setscheduler(pid_t pid, int policy, const struct sched_param* param)" to set the scheduling policy of the process specified by pid to policy and the scheduling parameters to "param".

If pid is 0, the policy and parameters are set for the calling process. The following policies are available:

- SCHED_FIFO

First in first out. Processes are executed on the CPU in the order in which they were added to the queue of processes to be run, for each priority.

- SCHED_RR

Round-Robin. Identical to SCHED_FIFO except that a process runs only for the defined time slice (see sched_rr_get_interval()). Once the process has completed its time slice it is placed on the tail of the queue of processes to be run, for its priority.

```
#include <sched.h>

struct sched_param sp;

sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
ret = sched_setscheduler(0, SCHED_FIFO, &sp);
```

Linux allows the **static priority** value range 1 to 99 for SCHED_FIFO and SCHED_RR. Please use "sched_get_priority_max" to set the priority of processes such that the process is executed in "RT" mode.

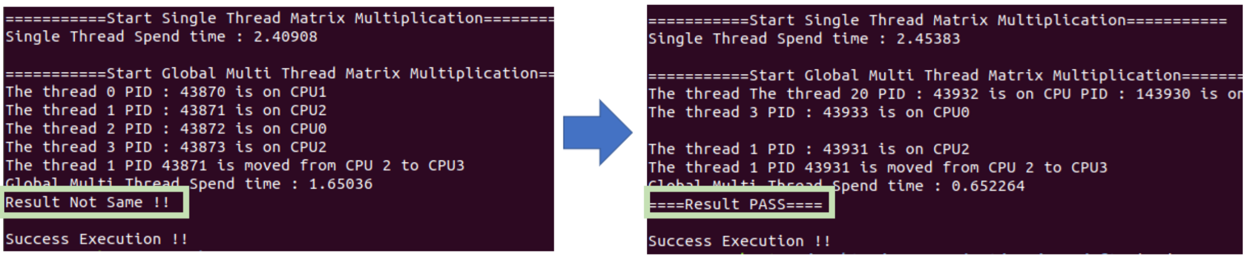
```
top - 18:44:09 up 10 min,  1 user,  load average: 1.28, 1.06, 0.70
Threads: 537 total,   5 running, 465 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.8 us,  0.2 sy,  0.0 ni, 80.8 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8089168 total,  5746196 free,  1327760 used,  1015212 buff/cache
KiB Swap:   998396 total,   998396 free,        0 used.  6370104 avail Mem

  PID P COMMAND     PR USER      NI    VIRT    RES    SHR S %CPU %MEM     TIME+
 2844 4 FIFO.out    rt root       0   48224   1624   1476 S 75.4  0.0   1:34.41
 2882 0 FIFO.out    rt root       0   48224   1624   1476 R 24.3  0.0   0:00.73
 2883 1 FIFO.out    rt root       0   48224   1624   1476 R 24.3  0.0   0:00.73
 2884 2 FIFO.out    rt root       0   48224   1624   1476 R 24.3  0.0   0:00.73
  227 5 kworker/u+  20 root       0       0      0      0 I  1.0  0.0   0:24.35
 2427 7 chromium-+  20 luiz2295   0 1628572 390512 140608 S  1.0  4.8   0:17.95
  229 6 kworker/u+  20 root       0       0      0      0 I  0.7  0.0   0:32.03
 1123 7 Xorg        20 root       0  482732  89520  46192 S  0.3  1.1   0:12.49
 1681 3 compiz      20 luiz2295   0 1324032 109160  76416 S  0.3  1.3   0:09.56
 2069 6 chromium-+  20 luiz2295   0 4739328 426608 142596 S  0.3  5.3   0:03.79
 2431 4 Chrome_Ch+  20 luiz2295   0 1628572 390512 140608 S  0.3  4.8   0:01.69
 2434 7 Compositor  20 luiz2295   0 1628572 390512 140608 S  0.3  4.8   0:05.73
 2802 3 TaskSched+  20 luiz2295   0 1628572 390512 140608 S  0.3  4.8   0:00.14
 2777 7 gnome-ter+  20 luiz2295   0  599196  36908  28620 S  0.3  0.5   0:00.62
 2858 3 top         20 luiz2295   0   49548   4324   3288 R  0.3  0.1   0:00.13
    1 2 systemd     20 root       0  185324   6028   4088 S  0.0  0.1   0:01.16
    2 2 kthreadd    20 root       0       0      0      0 S  0.0  0.0   0:00.00
```

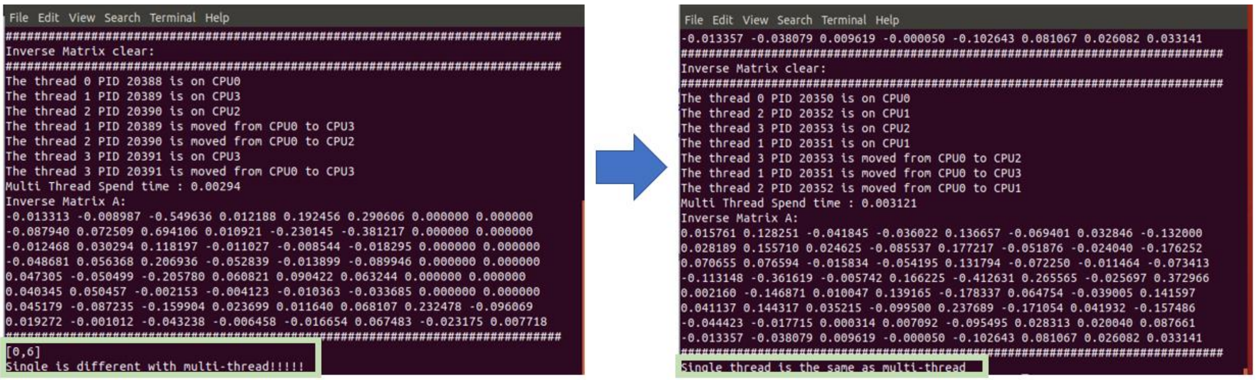
※You have to use superuser for execution

Output

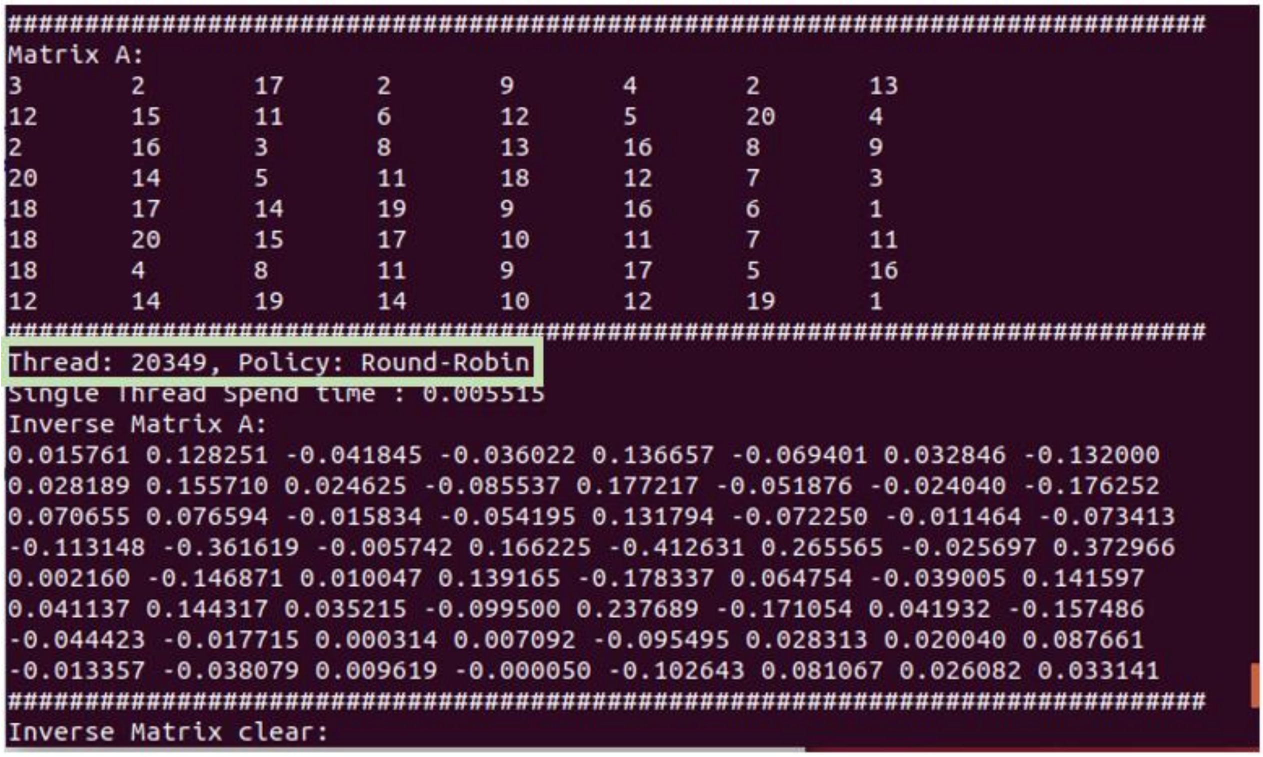
- Resource Sharing

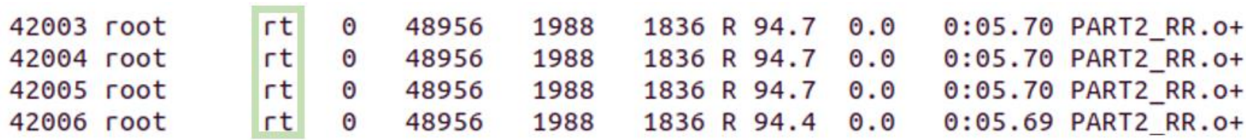


- Synchronization



- Scheduler Implementation





- **<u>The file must contain the following files</u>**

  - PART1_Mutex.cpp
  - PART1_Reentrant.cpp
  - PART2_Barrier.cpp
  - PART2_FIFO.cpp
  - PART2_RR.cpp
  - Student ID_HW2.pdf

嚴禁抄襲，發生該類似情況者，一律以零分計算

Command Line

Display top CPU process: top –H

Crediting :

- PART I

  [Resource Sharing. 50%]

  - Indicate incorrect parts and explain the reason. **10%**
  - Describe how to protect the resource by using Mutex and show the result. **15%**
  - Describe how to modify Non-Reentrant Functions to Reentrant Functions by using local variables and show the result. **15%**
  - Compare the elapsed time between the two methods and describe the reasons. **10%**

- PART II

  [Synchronization. 30%]

  - Indicate incorrect parts and explain the reason. **15%**
  - Describe how to synchronize thread by using Barrier and show the result. **15%**

  [Scheduler Implementation. 20%]

  - Describe how to implement the scheduler setting in global scheduling. (FIFO, RR) **10%**
  - Show the scheduling states of tasks. **10%**

Project submit

Submit deadline: 12 :20, May. 20, 2020

Submission: Moodle

File name format: ESSD_Student ID_HW2.zip

※ Strictly prohibited copying !

ESSD_Student ID_HW2.zip must include the report and source code.

嚴禁抄襲，發生該類似情況者，一律以零分計算