Embedded System Software Design

Project 1

Problem Definition:

By parallelizing some regions in a program, we can reduce the execution duration. In this project, you are asked to observe the performance of programs with single and multi-threaded execution by POSIX thread in Linux, and to observe the response time of such program managed by global and partition (First-Fit, Best-Fit, Worst-Fit) schedulers.

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 Creation

The pthread_create () function starts a new thread.

.

```
# include <pthread.h>
int pthread_create (
                pthread_t *thread,
                const pthread_attr_t *attr,
                void(*start_routine) (void *),
                void *arg
        );
```

Implement thread creating

```
#include <pthread.h>
void * Multi_Matrix_Multiplication (void *args);

int main ()
{
    pthread_t thread1;
    pthread_create (&thread1, NULL, Multi_Matrix_Multiplication, NULL);
}
```

Implement for loop

```
#include <pthread.h>
#define CORE_NUM 4

struct Thread_Data
{
            int Start;
            int End;
            int Total_Size;
            int Thread_ID;
            int Core;
            float** Input_Matrix;
            float** Output_Matrix;
};

void* Global_Multi_Matrix_Multiplication (void *args);

int main ()
{
    Thread_Data Multi_Thread_Data [CORE_NUM];
    for (int i = 0; i < CORE_NUM; i++) {
        pthread_create(&pthread_Thread[i], NULL,
Global_Multi_Matrix_Multiplication, &Multi_Thread_Data[i]);
    }
}

void* Global_Multi_Matrix_Multiplication (void *args)
{
            Thread_Data *Thread = (struct Thread_Data*) args;
                                        .
                                        .
}
```

Implement for non static member function

```
#include <pthread.h>

class Thread {

void* Multi_Matrix_Multiplication (void *args)
          {
             cout << “Multi_Matrix_Multiplication” << endl;
          }
};

typedef void * (*THREADFUNCPTR) (void *); // function pointer

int main ()
{
    Thread *t1 = new Thread;
    pthread_t thread1;
    pthread_create (&thread1, NULL, (THREADFUNCPTR)
&Thread::Multi_Matrix_Multiplication , t1 ) ;
}
```

POSIX Thread Join

The function pthread_join () allows the calling thread to wait for the ending of the target thread. If the thread has already terminated, then pthread_join () returns immediately. The thread specified by thread must be joinable which means that the thread shall to be ended.

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **retval)
```

Implement thread join

We need to use thread_join () to synchronize our threads when the threads are terminated. If the parameter “retval” is not Null, then pthread_join () copies the exit status of the target thread into the location pointed to by “retval”

```
#include <pthread.h>
void * Multi_Matrix_Multiplication(void *args);

int main()
{
	pthread_t thread1, thread2;
	pthread_create(&thread1, NULL, Multi_Matrix_Multiplication, NULL);
	pthread_create(&thread2, NULL, Multi_Matrix_Multiplication, NULL);
	pthread_join(&thread1,NULL);
	pthread_join(&thread2,NULL);
}
```

POSIX Thread Mutex

The mutex object be locked by calling pthread_mutex_lock (). If the mutex is already locked, the calling thread shall block 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_lock( &count_mutex );
pthread_mutex_unlock( &count_mutex );
```

.

POSIX Thread System call

In pthread, we use “syscall (SYS_gettid)” to get the PID of current thread and use “sched_setaffinity (pid_t pid, size_t cpusetsize, const cpu_set_t *mask)” to set of CPUs on which it is eligible to run.

```
int Get_PID(void)
{
        int PID = syscall(SYS_gettid);
        return PID
}
void Set_CPU( int CPU_NUM )
{
        cpu_set_t set;
        CPU_ZERO(&set);
        CPU_SET(CPU_NUM, &set);
        sched_setaffinity(0, sizeof(set), &set);
}
```

Global

```
===========Start Global Multi Thread Matrix Multiplication===========
The thread 0 PID : 6846 is on CPU2
The thread 2 PID : 6848 is on CPU1
The thread 1 PID : 6847 is on CPU0
The thread 3 PID : 6849 is on CPU3
The thread 2 PID 6848 is moved from CPU 1 to CPU0
The thread 1 PID 6847 is moved from CPU 0 to CPU1
The thread 1 PID 6847 is moved from CPU 1 to CPU0
The thread 2 PID 6848 is moved from CPU 0 to CPU1
The thread 0 PID 6846 is moved from CPU 2 to CPU1
Global Multi Thread Spend time : 31.7453
====Result PASS====
```

Partition

```
===========Start Partition Multi Thread Matrix Multiplication===========
The thread 0 PID : 6850 is on CPU0
The thread 2 PID : 6852 is on CPU2
The thread 1 PID : 6851 is on CPU1
The thread 3 PID : 6853 is on CPU3
Partition Multi Thread Spend time : 30.647
====Result PASS====
```

Partition First-Fit

```
===========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread 19 is not push.
CPU0 : Core Number : 0
[ 0, 1, 2, 3, 6, ]
Total Utilization : 0.981

CPU1 : Core Number : 1
[ 4, 5, 7, 8, 9, 10, ]
Total Utilization : 0.957

CPU2 : Core Number : 2
[ 11, 12, 13, 14, 15, 16, ]
Total Utilization : 0.9525

CPU3 : Core Number : 3
[ 17, 18, ]
Total Utilization : 0.766

Thread ID : 0   PID : 2575      Core : 0        Utilization : 0.133     Matrix_Size : 266
Thread ID : 2   PID : 2577      Core : 0        Utilization : 0.08      Matrix_Size : 160
Thread ID : 3   PID : 2578      Core : 0        Utilization : 0.344     Matrix_Size : 688
Thread ID : 6   PID : 2579      Core : 0        Utilization : 0.096     Matrix_Size : 192
Thread ID : 4   PID : 2580      Core : 1        Utilization : 0.152     Matrix_Size : 304
Thread ID : 5   PID : 2581      Core : 1        Utilization : 0.157     Matrix_Size : 314
Thread ID : 7   PID : 2582      Core : 1        Utilization : 0.32      Matrix_Size : 640
Thread ID : 8   PID : 2583      Core : 1        Utilization : 0.16      Matrix_Size : 320
Thread ID : 9   PID : 2584      Core : 1        Utilization : 0.068     Matrix_Size : 136
Thread ID : 10  PID : 2585      Core : 1        Utilization : 0.1       Matrix_Size : 200
Thread ID : 11  PID : 2586      Core : 2        Utilization : 0.1465    Matrix_Size : 293
Thread ID : 12  PID : 2587      Core : 2        Utilization : 0.1585    Matrix_Size : 317
Thread ID : 13  PID : 2588      Core : 2        Utilization : 0.165     Matrix_Size : 330
Thread ID : 14  PID : 2589      Core : 2        Utilization : 0.3565    Matrix_Size : 713
Thread ID : 15  PID : 2590      Core : 2        Utilization : 0.05      Matrix_Size : 100
Thread ID : 16  PID : 2591      Core : 2        Utilization : 0.076     Matrix_Size : 152
Thread ID : 17  PID : 2592      Core : 3        Utilization : 0.373     Matrix_Size : 746
Thread ID : 18  PID : 2593      Core : 3        Utilization : 0.393     Matrix_Size : 786
Thread ID : 1   PID : 2576      Core : 0        Utilization : 0.328     Matrix_Size : 656
Multi Thread Spend time : 7.19498
```

Command Line

PART1:

Compiler : g++ -pthread pthread_part1.cpp -o pthread_part1.out

Execute : ./pthread_part1.out

PART2:

Compiler : g++ -pthread pthread_part2.cpp -o pthread_part2.out

Execute : ./pthread_part2.out <Input file>

Crediting:

※CPU must be limited in four cores.

- PART I

[Global Scheduling. 25%]

- Describe how to implement multithread by using pthread. **10%**
- Describe how to estimate task migration. **5%**
- Show the scheduling states of tasks. **10%**

[Partition Scheduling. 20%]

- Describe how to implement partition scheduling by using pthread. **10%**
- Show the scheduling states of tasks. **10%**

- PART II

[Scheduler Implementation. 45%]

- Describe how to implement the scheduler setting in partition scheduling. (First-Fit, Best-Fit, Worst-Fit) **15%**
- Show the scheduling states of tasks. (You have to show the result of Input_10.txt and Input_20.txt) **30%**

[Result. 10%]

- Analyze and compare the response time of the program, with three execution types. (Single, Global, Partition) **10%**

Project submits

Submit deadline: 12:30, Apr. 22, 2020

Submission: Moodle

File name format: ESSD_Student ID_HW1.zip

※ Strictly prohibited copying!

ESSD_Student ID_HW1.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