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) with 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++ 7.5.0
- ✓ GNU make: 4.1

Part 1:

Divide the matrix convolution into **four** independent parts which could execute concurrently. Then use multi-thread execution with **Global** and **Partition** scheduling to boost the performance of matrix convolution. The execution result is demonstrated in Figure 1.

```
numThread : 4
0.Matrix size : 5000
1.Matrix size : 5000
2.Matrix size : 5000
3.Matrix size : 5000
Workload Utilization : 2

===========Generate Matrix Data===========
Generate Date Spend time : 1.199

===========Start Single Thread Convolution===========
Single Thread Spend time : 5.78254

===========Start Global Multi-Thread Convolution===========
Thread ID : 0    PID : 300569    Core : 6
Thread ID : 1    PID : 300570    Core : 1
Thread ID : 2    PID : 300571    Core : 3
Thread ID : 3    PID : 300572    Core : 4
The thread 1 PID 300570 is moved from CPU 1 to 5
The thread 1 PID 300570 is moved from CPU 5 to 1
The thread 2 PID 300571 is moved from CPU 3 to 7

===============checking===============
Part1 global matrix convolution using global scheduling correct.
Part1 global matrix convolution compute result correct
Global Multi Thread Spend time : 1.33458
```

Figure 1. Global scheduling result

Please print the result of your global multi-thread matrix convolution following the format shown in Figure 1. First, print out the thread information with process ID, the core where the thread executes, and print out the migration state of each thread. Lastly, use the class check (libs/check.h) to print out the correctness of the scheduling result and matrix convolution result.

```
===========Start Partition Multi-Thread Convolution===========
Thread ID : 0    PID : 300574    Core : 0
Thread ID : 2    PID : 300576    Core : 2
Thread ID : 3    PID : 300577    Core : 3
Thread ID : 1    PID : 300575    Core : 1

===============checking===============
Part1 partition matrix convolution using parition scheduling correct.
Part1 partition matrix convolution compute result correct
Partition Multi Thread Spend time : 1.29867
```

Figure 2. Partition scheduling result

Please print the partition result of matrix convolution following Figure 2.

Part 2:

Execute 5 matrix convolutions (input/part2_input_5.txt) and 10 matrix convolutions (input/part2_input_10.txt) with different partition methods (**First-Fit**, **Best-Fit**, **Worst-Fit**).

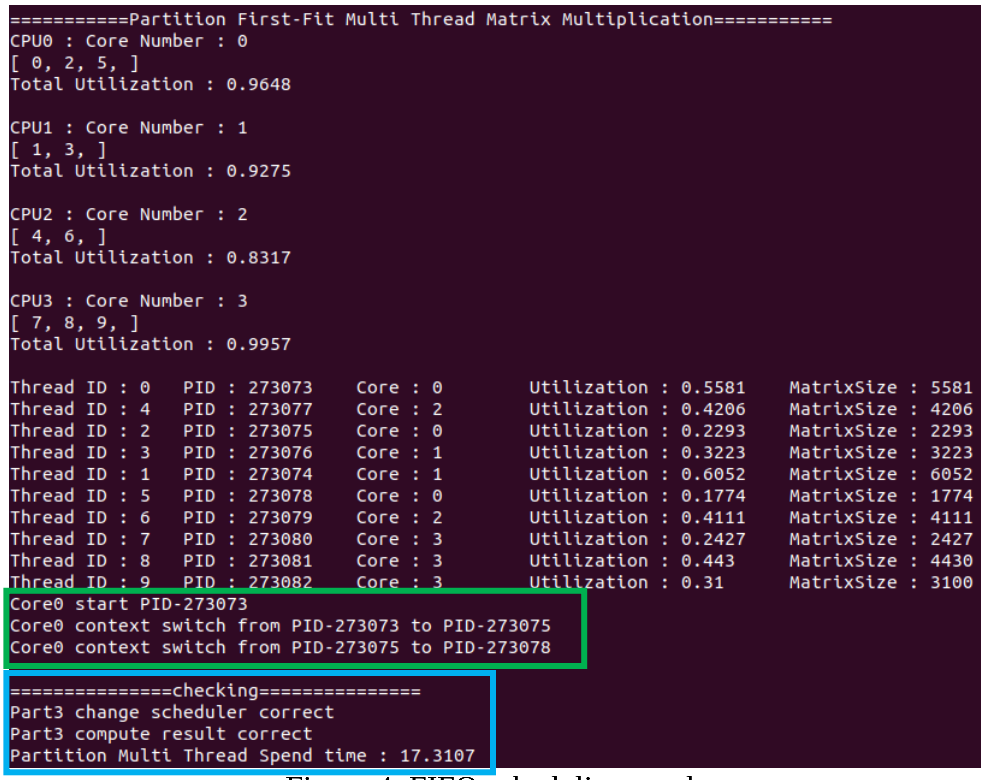
```
===========Partition First-Fit Multi Thread Matrix Multiplication===========
Thread-4 not schedulable.
CPU0 : Core Number : 0
[ 0, ]
Total Utilization : 0.5001

CPU1 : Core Number : 1
[ 1, ]
Total Utilization : 0.5001

CPU2 : Core Number : 2
[ 2, ]
Total Utilization : 0.5001

CPU3 : Core Number : 3
[ 3, ]
Total Utilization : 0.5001

Thread ID : 0    PID : 262134    Core : 0        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 3    PID : 262137    Core : 3        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 2    PID : 262136    Core : 2        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 1    PID : 262135    Core : 1        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 4    PID : 262138    Core : 6        Utilization : 0.5001    MatrixSize : 5001

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 9.79449
```

Figure 3. First-fit partition result

Please print the partition result of First-fit, Best-fit, and Worst-fit following the format shown in Figure 3. When there is a thread not schedulable then print out the non-schedulable thread in “Thread-# not schedulable” format (as shown in the green box). Otherwise, print out the partition result (as shown in the yellow box). Finally, as shown in the blue box, use the class check (libs/check.h) to print out the correctness of partition result and matrix convolution result.

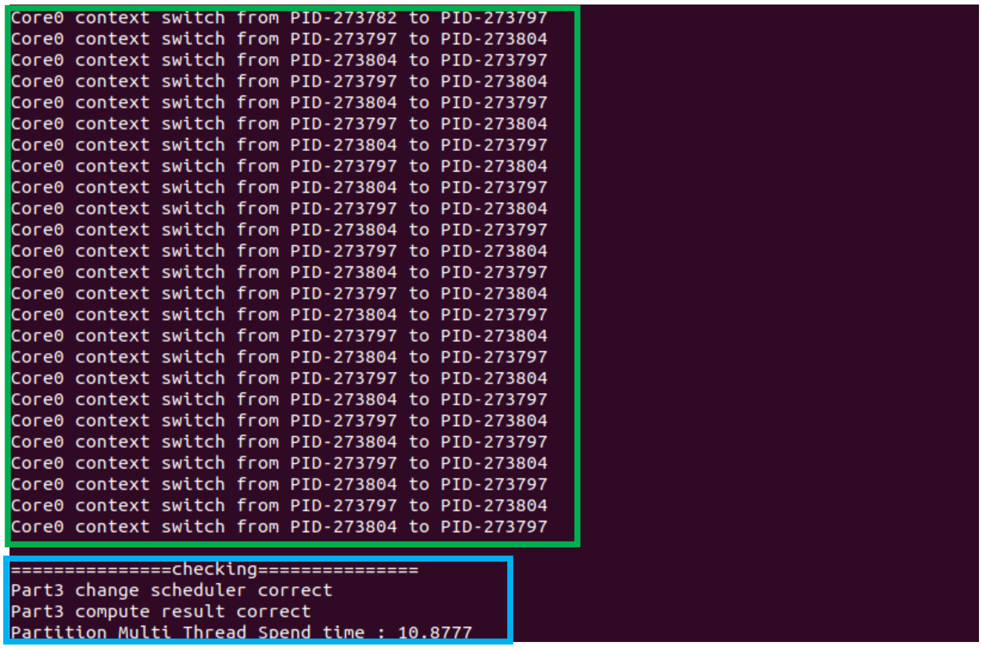
Part 3:

Using the **FIFO** and **Round-Robin** scheduling based on the partition result of **Part 2**.



```
===========Partition First-Fit Multi Thread Matrix Multiplication===========
CPU0 : Core Number : 0
[ 0, 2, 5, ]
Total Utilization : 0.9648

CPU1 : Core Number : 1
[ 1, 3, ]
Total Utilization : 0.9275

CPU2 : Core Number : 2
[ 4, 6, ]
Total Utilization : 0.8317

CPU3 : Core Number : 3
[ 7, 8, 9, ]
Total Utilization : 0.9957

Thread ID : 0    PID : 273073    Core : 0    Utilization : 0.5581    MatrixSize : 5581
Thread ID : 4    PID : 273077    Core : 2    Utilization : 0.4206    MatrixSize : 4206
Thread ID : 2    PID : 273075    Core : 0    Utilization : 0.2293    MatrixSize : 2293
Thread ID : 3    PID : 273076    Core : 1    Utilization : 0.3223    MatrixSize : 3223
Thread ID : 1    PID : 273074    Core : 1    Utilization : 0.6052    MatrixSize : 6052
Thread ID : 5    PID : 273078    Core : 0    Utilization : 0.1774    MatrixSize : 1774
Thread ID : 6    PID : 273079    Core : 2    Utilization : 0.4111    MatrixSize : 4111
Thread ID : 7    PID : 273080    Core : 3    Utilization : 0.2427    MatrixSize : 2427
Thread ID : 8    PID : 273081    Core : 3    Utilization : 0.443     MatrixSize : 4430
Thread ID : 9    PID : 273082    Core : 3    Utilization : 0.31      MatrixSize : 3100
Core0 start PID-273073
Core0 context switch from PID-273073 to PID-273075
Core0 context switch from PID-273075 to PID-273078

==============checking==============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 17.3107
```

Figure 4. FIFO scheduling result



```
Core0 context switch from PID-273782 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797
Core0 context switch from PID-273797 to PID-273804
Core0 context switch from PID-273804 to PID-273797

==============checking==============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 10.8777
```

Figure 5. Round-Robin scheduling result

Please print the context switch state on core-0 following the format in green box. Then, using the class check (libs/check.h) to print out the correctness of scheduling policy and matrix convolution result (as shown in blue box).

Command Line:

Part 1:

Compile: make part1.out

Execute: ./part1.out part1_Input.txt

Part 2:

Compile: make part2.out

Execute: ./part2.out part2_Input_5.txt

./part2.out part2_Input_10.txt

Part 3:

Compile: make part3_rr.out

make part3_fifo.out

Execute: sudo ./part3_rr.out part3_Input.txt

sudo ./part3_fifo.out part3_Input.txt

Precautions

Using the class variable "**_thread**" as the input argument of "**pthread_create**".

```
class Thread
{
  public:

    /* Constructrue */
    ~Thread();
    void init (float**, float**, float**);

    /* Part 1 *//* Part 3 */
    static void* convolution(void*);      // Perform convolution

    /* Part 1 */
    void setUpCPUAffinityMask (int);      // Pined the thread to core

    /* Part 3 */
    void setUpScheduler ();               // Set up the scheduler for current thread
```

⋮

```
  public:
    pthread_t _thread;
```

Figure 6. Class varaible _thread

Crediting:

- **Part 1**

[Global Scheduling. 10%]

  - Describe how to implement Global scheduling by using pthread. 5%
  - Describe how to observe task migration. 5%

[Partition Scheduling. 5%]

  - Describe how to implement partition scheduling by using pthread.

[Result. 10%]

  - Show the scheduling states of tasks. (You have to show the screenshot result of using the input part1_input.txt)

- **Part 2**

[Partition method Implementation. 10%]

  - Describe how to implement the three different partition methods (First-Fit, Best-Fit, Worst-Fit) in partition scheduling.

[Result. 30%]

  - Show the scheduling states of tasks. (You have to show the screenshot result of using input part2_input_5.txt and part2_input_10.txt)

- **Part 3**

[Scheduler Implementation. 10%]

  - Describe how to implement the scheduler setting in partition scheduling. (FIFO with FF, RR with FF)

[Result. 10%]

  - Show the process execution states of tasks. (You have to show the screen-shot result of using input part3_input.txt)

- **Discussion**

  - Analyze and compare the response time of the program, with single thread and multi-thread using in part 1 and part 2. (Including Single, Global, First-Fit, Best-Fit, Worst-Fit) 5%

  - Analyze and compare the characteristic of the three different partition methods (First-Fit, Best-Fit, Worst-Fit) 5%

- Analyze and compare the response time of the program, with two different schedulers. (FIFO with FF, RR with FF) 5**%**

<u>Project submits:</u>

- ✓ Submit deadline: 13:00, March. 30, 2022
- ✓ Submission: Moodle
- ✓ File name format: ESSD_Student ID_PA1.zip (e.g. ESSD_M110XXXXX_PA1.zip)
- ✓ Including source code:

```
.
├── ESSD_M110XXXXX_PA1
│   ├── input
│   │   ├── part1_input.txt
│   │   ├── part2_input_10.txt
│   │   ├── part2_input_5.txt
│   │   └── part3_input.txt
│   ├── libs
│   │   ├── check.h
│   │   └── check.o
│   ├── makefile
│   ├── pa1.cpp
│   └── src
│       ├── config.h
│       ├── cpu.cpp
│       ├── cpu.h
│       ├── system.cpp
│       ├── system.h
│       ├── thread.cpp
│       └── thread.h
└── ESSD_M110XXXXX_PA1.pdf
```

- ✓ Note: ESSD_Student ID_PA1.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

Hint:

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_Convolution (void *args);

struct parameter{
        int x;
};
   int main ()

   {

     pthread_t thread1;

     parameter my_param;

     pthread_create (  &thread1,

                       NULL,

                       Multi_Matrix_Convolution,

                       &my_param);

   }
```

POSIX Thread Join

The function pthread_join () allows the calling thread to wait for the ending of another 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 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" 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_Convolution (void *args);

   int main()

   {

     pthread_t thread1, thread2;

     pthread_create (&thread1, NULL, Multi_Matrix_Convolution, NULL);

     pthread_create (&thread2, NULL, Multi_Matrix_Convolution, NULL);

     pthread_join (&thread1, NULL);

     pthread_join (&thread2, NULL);

   }
```

POSIX Thread Mutex

The mutex object could be locked by calling pthread_mutex_lock (). If the mutex is already locked, the calling thread shall block until the mutex becomes available.

```
#include <sched.h>

pthread_mutex_t count_mutex;

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

System call

Use “syscall (SYS_gettid)” to get the PID of the 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_ID)

  {

        cpu_set_t set;

        CPU_ZERO (&set);

        CPU_SET (CPU_ID, &set);

        sched_setaffinity (0, sizeof(set), &set);

  }
```

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 and parameters by giving "policy" and "param". You can check the scheduler setting by the function return value where "-1" means setting failed.

If pid is 0, the policy and parameters are set for the calling thread. 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 to range from 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.