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

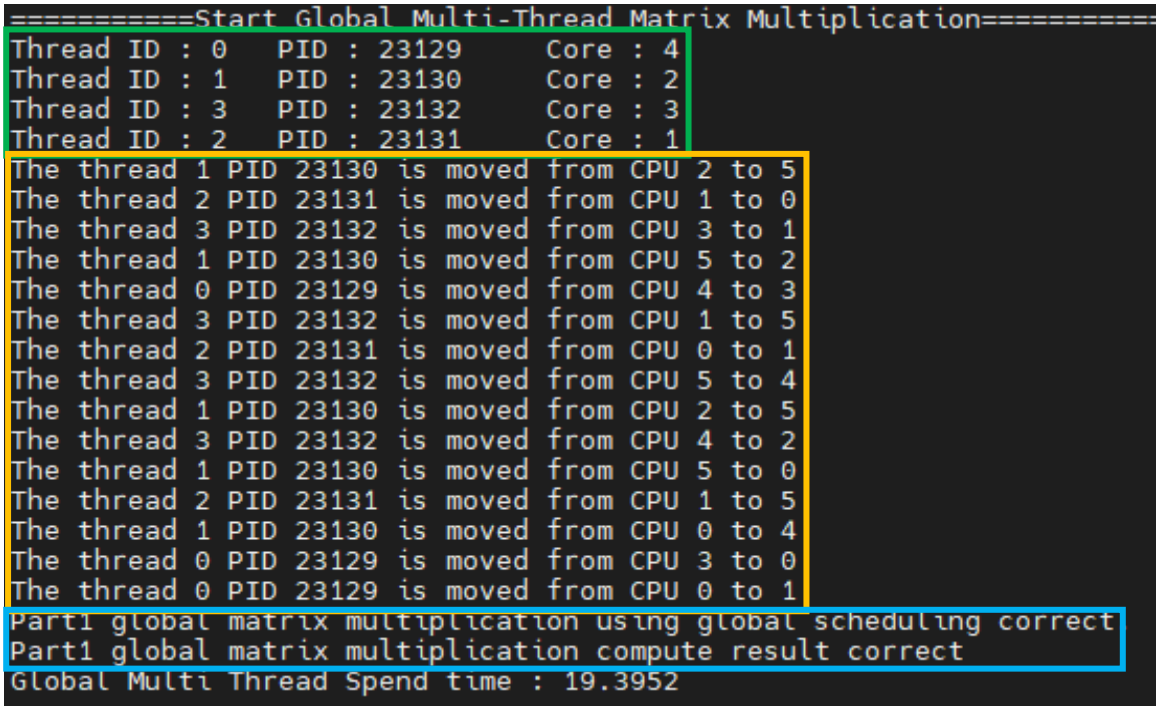


Figure 1. Global scheduling result

Please print the result of your global multi-thread matrix multiplication following the format shown in figure 1. First, print out the thread information with process id and the core thread execute on, and print out the migration state of each thread. Lastly, using the class check (libs/check.h) to print out the correctness of the scheduling result and matrix multiplication result.

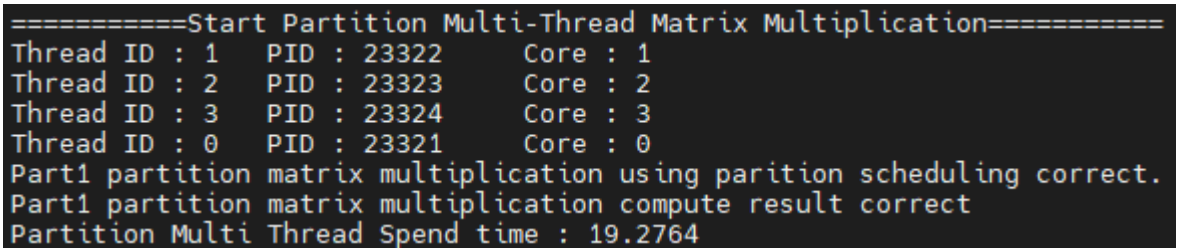


Figure 2. Partition scheduling result

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

Part 2:

Execute 10 matrix multiplications (input/part2_Input_10.txt) and 20 (input/part2_Input_20.txt) matrix multiplications with different partition methods (**First-Fit**, **Best-Fit**, **Worst-Fit**).

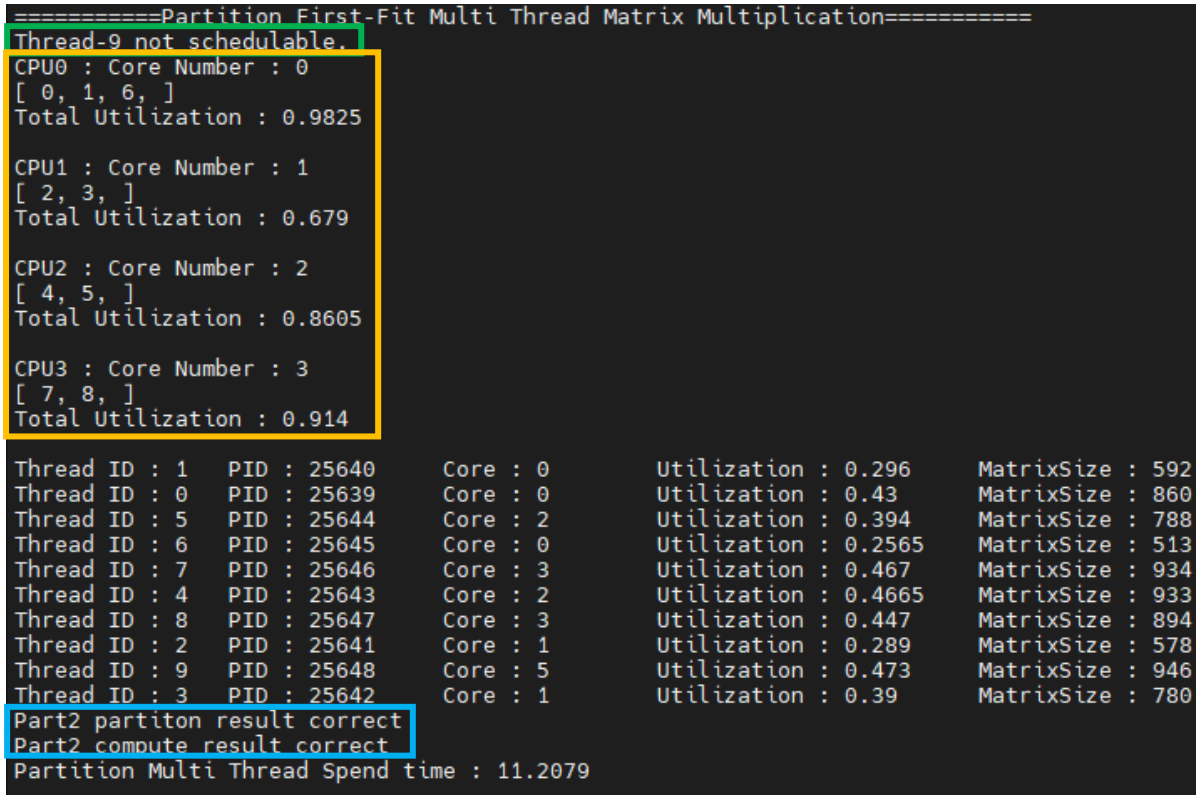


Figure 3. First-fit partition result

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 green box). Otherwise, print out the partition result (as shown in yellow box). Finally, as shown in blue box, using the class check (libs/check.h) to print out the correctness of partition result and matrix multiplication result.

Part 3:

Using the **FIFO** and **Round-Robin** scheduling based on the partition result of **Part2**.

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
CPU0 : Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

CPU1 : Core Number : 1
[ 2, 3, ]
Total Utilization : 0.729

CPU2 : Core Number : 2
[ 5, 6, ]
Total Utilization : 0.6505

CPU3 : Core Number : 3
[ 7, 8, ]
Total Utilization : 0.764

Core0 start PID-32636
Core0 context switch from PID-32636 to PID-32635
Core0 context switch from PID-32635 to PID-32639
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 11.8632
```

Figure 4. FIFO 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 multiplication 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_10.txt

./part2.out part2_Input_20.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 "**pthreadThread**" as the input argument of "**pthread_create**".

```
class Thread
{

  public:
    void setThreadID (int); // Set the _ID of thread
    void setThreadCore (int); // Set the core thread pinned on
    void setThreadMatrixSize (int); // Set the hight / widht of matrix which thread calculate on
    void setStartCalculatePoint (int);  // Set the start calculate point for matrix multiplication. (hint: part1)
    void setEndCalculatePoint (int); // Set the end calculate point for matrix multiplication. (hint: part1)
    void setSchedulingPolicy (int); // Set the scheduling policy for thread. (hint: part3)
    void setCheck (Check*);

    void initialThread (float**, float**, float**); // Set the pointer point to singleResult, multResult, and matrix define in System.
    void singleMatrixMultiplication (); // Single thread matrix multiplication.
    void printInformation (); // Print out the thread informaiton.

    /* Part1 and Part3 */
    static void* matrixMultiplication (void*); // Multi-thread matrix multiplication

    /* Part1 */
    void setUpCPUAffinityMask (int); // Pined the thread on specify core

    /* Part 3 */
    void setUpScheduler (); // Set the scheduling policy for specify thread.

    float utilization () {return _utilization;};
    int matrixSize () {return _matrixSize;};
    int ID () {return _ID;};
    int schedulingPolicy () {return _schedulingPolicy;};

  public:
    pthread_t pthreadThread;
```

Figure 5. Class varaible pthreadThread

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_10.txt and part2_Input_20.txt)

- **Part 3**

[Scheduler Implementation. 10%]

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

[Result. 10%]

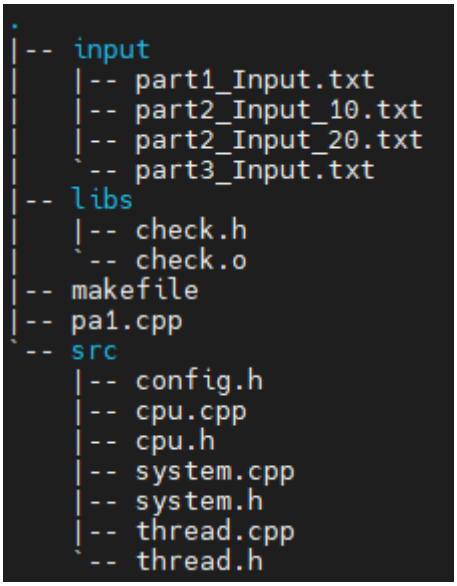
- Show the process execution states of tasks. (You have to show the screenshot result of using input part3_Input.txt)

- **Discussion**

- Analyze and compare the response time of the program, with single thread and multi-thread using in part1 and part2. (Including Single, Global, First-Fit, Best-Fit, Worst-Fit) 10**%**
- Analyze and compare the response time of the program, with two different schedulers. (FIFO with FF, RR with FF) 5**%**

Project submits:

- ✓ Submit deadline: 12:30, April. 21, 2021
- ✓ Submission: Moodle
- ✓ File name format: ESSD_Student ID_PA1.zip (e.g. ESSD_M10704328_PA1.zip)
- ✓ Including source code:



- ✓ 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_Multiplication (void *args);

    int main ()

    {

       pthread_t thread1;

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

    }
```

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 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 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_NUM )

  {

        cpu_set_t set;

        CPU_ZERO(&set);

        CPU_SET(CPU_NUM, &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 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 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 ranges 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.