

Embedded System Software Design Project 1

M10907324 吳俊逸

● Part 1 [Global Scheduling. 10%]

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

```
void
System::globalMultiCoreMatrixMulti()
{
    std::cout << "\n=====Start Global Multi-Thread Matrix Multiplication===== " << std::endl;
    check->setCheckState(GLOBAL);
    setStartTime();

    /*~~~~~Your code(PART1)~~~~~*/
    // Create thread and join
    for (int i = 0; i < numThread; i++) (pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]));
    for (int i = 0; i < numThread; i++) (pthread_join(threadSet[i].pthreadThread, NULL));
    //將create與join分開，這樣才會讓系統一次性地將create做完之後再join
    /*~~~~~END~~~~~*/

    setEndTime();
    std::cout << "Global Multi Thread Spend time : " << _timeUse << std::endl;
    cleanMultiResult();
}

void*
Thread::matrixMultiplication(void* args)
{
    Thread *obj = (Thread*)args;

    #if (PART == 3)
        obj->setUpScheduler();
    #endif

    /*~~~~~Your code(PART1)~~~~~*/
    // Set up the affinity mask
    obj->setUpCPUAffinityMask(sched_getcpu());
    /*~~~~~END~~~~~*/
    /* Print Thread information */
    obj->core = sched_getcpu();
    obj->PID = syscall(SYS_gettid);
    obj->printInformation();
    /* matrix multiplication */
    for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {
        for (int j = 0; j < obj->_matrixSize; j++) {
            obj->multiResult[i][j] = 0;
            for (int k = 0; k < obj->_matrixSize; k++) {
                obj->multiResult[i][j] += obj->matrix[i][k] * obj->matrix[k][j];
            }
            /*~~~~~Your code(PART1)~~~~~*/
            // Observe the thread migration
            if (obj->core != sched_getcpu())
            {
                std::cout << "The thread " << obj->_ID << " PID : " << obj->PID << " is move from CPU " << obj->core << " to " << sched_getcpu() << std::endl;
                obj->core = sched_getcpu();
            }
            /*~~~~~END~~~~~*/
        }
    }
}
```

- Describe how to observe task migration. 5%

```
void
System::partitionMultiCoreMatrixMulti()
{
    #if (PART == 1)
        std::cout << "\n=====Start Partition Multi-Thread Matrix Multiplication===== " << std::endl;
        check->setCheckState(PARTITION);
    #endif
    setStartTime();

    /*~~~~~Your code(PART1)~~~~~*/
    // Set thread execute core.
    // Create thread and join.

    for (int i = 0; i < numThread; i++) ( threadSet[i].setUpCPUAffinityMask(i); pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]); )
    for (int i = 0; i < numThread; i++) ( pthread_join(threadSet[i].pthreadThread, NULL); )
    //將create與join分開，這樣才會讓系統一次性地將create做完之後再join
    /*~~~~~END~~~~~*/

    setEndTime();
    std::cout << "Partition Multi Thread Spend time : " << _timeUse << std::endl;
    cleanMultiResult();
}
```

```

void
Thread::setUpCPUAffinityMask(int cpu_num)
{
    /*~~~~~Your code(PART1)~~~~~*/
    // Pined the thread to core.將線程固定到核心。

    int s;
    cpu_set_t cpuset;
    pthread_t thread;
    thread = pthread_self();//自己的PID
    CPU_ZERO(&cpuset);
    CPU_SET(cpu_num, &cpuset);
    s = pthread_setaffinity_np(thread, sizeof(cpuset), &cpuset);
    //if (s != 0) {handle_error_en(s, "pthread_setaffinity_np");}
    /* Check the actual affinity mask assigned to the thread. */
    s = pthread_getaffinity_np(thread, sizeof(cpuset), &cpuset);
    //if (s != 0) {handle_error_en(s, "pthread_getaffinity_np");}
    //std::cout << "Set returned by pthread_getaffinity_np() contained:";
    //for (int j = 0; j < CPU_SETSIZE; j++) {
    //    if (CPU_ISSET(j, &cpuset)) {
    //        std::cout << "CPU " << j << "\n" << std::endl;
    //    }
    //}
    //}
    //exit(EXIT_SUCCESS);
    /*~~~~~END~~~~~*/
}

```

[Partition Scheduling. 5%]

- Describe how to implement partition scheduling by using pthread.

```

System::System(char* input_file)
{
    loadInput(input_file); // Set up threadSet, singleResult, multiResult, and matrix
    int init_num = 0;
    int range_num00[4] = { 0,0,0,0 };
    for (int i = 0; i < numThread; i++) {
        #if (PART == 1)
            // Set the singleResult, multiResult, and matrix to thread.
            threadSet[i].initialThread(singleResult[0], multiResult[0], matrix[0]);
            /*~~~~~Your code(PART1)~~~~~*/
            // Set up the calculate range of matrix.
            for (int i = 0; i < (threadSet[i].matrixSize() % numThread); i++) { range_num00[i] = 1; } //如果不是4的倍數時，如何平均分配
            int range_num = (threadSet[i].matrixSize() / numThread) + range_num00[i]; //平均分配所有數量的矩陣
            threadSet[i].setStartCalculatePoint(init_num); //設定矩陣開始大小
            init_num += range_num; //遞增矩陣需要的編號
            threadSet[i].setEndCalculatePoint(init_num); //設定矩陣結束大小
            //std::cout << "Start : " << init_num - range_num << "\tEnd : " << init_num << std::endl; //檢視想法是否正確
            /*~~~~~END~~~~~*/
        #else
            // Set the singleResult, multiResult, and matrix to thread.
            threadSet[i].initialThread(singleResult[i], multiResult[i], matrix[i]);
        #endif
    }
}

```

[Result. 10%]

- Show the scheduling states of tasks. (You have to show the screenshot result of using the input part1_Input.txt)

```

Input File Name : ./input/part1_Input.txt
numThread : 4

=====Start Single Thread Matrix Multiplication=====
Thread ID : 0    PID : 3193    Core : 0
Single Thread Spend time : 127.419

=====Start Global Multi-Thread Matrix Multiplication=====
Thread ID : 1    PID : 3196    Core : 1
Thread ID : 0    PID : 3195    Core : 2
Thread ID : 3    PID : 3198    Core : 0
Thread ID : 2    PID : 3197    Core : 3
The thread 0 PID : 3195 is move from CPU 2 to 1
The thread 3 PID : 3198 is move from CPU 0 to 2
The thread 3 PID : 3198 is move from CPU 2 to 1
The thread 3 PID : 3198 is move from CPU 1 to 2
The thread 2 PID : 3197 is move from CPU 3 to 1
The thread 1 PID : 3196 is move from CPU 1 to 3
The thread 2 PID : 3197 is move from CPU 1 to 3
The thread 3 PID : 3198 is move from CPU 2 to 1
The thread 3 PID : 3198 is move from CPU 1 to 2
The thread 2 PID : 3197 is move from CPU 3 to 1
The thread 1 PID : 3196 is move from CPU 3 to 0
The thread 0 PID : 3195 is move from CPU 0 to 3
The thread 3 PID : 3198 is move from CPU 2 to 1
The thread 2 PID : 3197 is move from CPU 1 to 2
The thread 3 PID : 3198 is move from CPU 1 to 0
The thread 1 PID : 3196 is move from CPU 0 to 1
The thread 1 PID : 3196 is move from CPU 1 to 0
The thread 0 PID : 3195 is move from CPU 3 to 1
The thread 3 PID : 3198 is move from CPU 0 to 2
The thread 3 PID : 3198 is move from CPU 2 to 1
The thread 3 PID : 3198 is move from CPU 1 to 3
The thread 0 PID : 3195 is move from CPU 1 to 2
The thread 2 PID : 3197 is move from CPU 2 to 1
Part1 global matrix multiplication using global scheduling correct.
Part1 global matrix multiplication compute result correct
Global Multi Thread Spend time : 38.9793

```

```

=====Start Partition Multi-Thread Matrix Multiplication=====
Thread ID : 0    PID : 18088    Core : 0
Thread ID : 1    PID : 18089    Core : 1
Thread ID : 2    PID : 18090    Core : 2
Thread ID : 3    PID : 18091    Core : 3
Part1 partition matrix multiplication using partition scheduling correct.
Part1 partition matrix multiplication compute result correct
Partition Multi Thread Spend time : 24.128

```

● 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%]

```
void
System::partitionFirstFit()
{
    std::cout << "\n=====Partition First-Fit Multi Thread Matrix Multiplication===== " << std::endl;
    #if (PART == 2)
        check->setCheckState(PARTITION_FF);
    #endif

    for (int i = 0; i < CORE_NUM; i++)
        cpuSet[i].emptyCPU(); // Reset the CPU set

    /*~~~~~Your code(PART2)~~~~~*/
    // Implement partition first-fit and print result.

    setStartTime();
    System::printCPUInformation_self(1); //顯示CPU分配狀態
    for (int i = 0; i < numThread; i++) {
        if (cpuSet[0].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPUAffinityMask(0); cpuSet[0].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[1].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPUAffinityMask(1); cpuSet[1].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[2].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPUAffinityMask(2); cpuSet[2].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[3].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPUAffinityMask(3); cpuSet[3].pushThreadToCPU(&threadSet[i]); }
        pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]);
    }
    for (int i = 0; i < numThread; i++) { pthread_join(threadSet[i].pthreadThread, NULL); }
    /*~~~~~END~~~~~*/
    setEndTime();
    std::cout << "Partition Multi Thread Spend time : " << _timeUse << std::endl;
    cleanMultiResult();
    //partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

FF：如上圖所示，我先逐一計算單次的 thread 是否可以放入 core0~3 中，如果符合 Utilization<1 則放置於 core 最小編號的 core 中。

```
void
System::partitionBestFit()
{
    std::cout << "\n=====Partition Best-Fit Multi Thread Matrix Multiplication===== " << std::endl;
    #if (PART == 2)
        check->setCheckState(PARTITION_BF);
    #endif

    for (int i = 0; i < CORE_NUM; i++)
        cpuSet[i].emptyCPU(); // Reset the CPU set

    /*~~~~~Your code(PART2)~~~~~*/
    // Implement partition first-fit and print result.
    setStartTime();

    System::printCPUInformation_self(2); //顯示CPU分配狀態
    for (int i = 0; i < numThread; i++) {
        float cpu_U_0 = cpuSet[0].utilization();
        float cpu_U_1 = cpuSet[1].utilization();
        float cpu_U_2 = cpuSet[2].utilization();
        float cpu_U_3 = cpuSet[3].utilization();
        if (cpuSet[0].utilization() + threadSet[i].utilization() < 1 && cpu_U_0 >= cpu_U_1 && cpu_U_0 >= cpu_U_2 && cpu_U_0 >= cpu_U_3) { threadSet[i].setUpCPUAffinityMask(0); cpuSet[0].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[1].utilization() + threadSet[i].utilization() < 1 && cpu_U_1 >= cpu_U_2 && cpu_U_1 >= cpu_U_3) { threadSet[i].setUpCPUAffinityMask(1); cpuSet[1].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[2].utilization() + threadSet[i].utilization() < 1 && cpu_U_2 >= cpu_U_3) { threadSet[i].setUpCPUAffinityMask(2); cpuSet[2].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[3].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPUAffinityMask(3); cpuSet[3].pushThreadToCPU(&threadSet[i]); }
        pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]);
    }
    for (int i = 0; i < numThread; i++) { pthread_join(threadSet[i].pthreadThread, NULL); }
    /*~~~~~END~~~~~*/
    setEndTime();
    std::cout << "Partition Multi Thread Spend time : " << _timeUse << std::endl;
    cleanMultiResult();
    //partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

BF：如上圖所示，我先記錄現在所有 core 的 Utilization，逐一計算單次的 thread 是否可以放入 core0~3 中，如果符合 Utilization<1 則放置於 core 的 Utilization 最大的 core 中。

```

void
SysExec::partitionWorstFit()
{
    std::cout << "\n=====Partition Worst-Fit Multi Thread Matrix Multiplication===== " << std::endl;
    if (PART == 2)
        check->setState(PARTITION_WF);
    #endif

    for (int i = 0; i < CORE_NUM; i++)
        cpuSet[i].emptyCPU();

    /*~~~~~Your code(PART2)~~~~~*/
    // Implement partition first-fit and print result.
    setStartTime();

    SysExec::printCPUInformation_self(3); //顯示CPU分配狀態
    for (int i = 0; i < numThread; i++) {
        float cpu_u_0 = cpuSet[0].utilization();
        float cpu_u_1 = cpuSet[1].utilization();
        float cpu_u_2 = cpuSet[2].utilization();
        float cpu_u_3 = cpuSet[3].utilization();
        if (cpuSet[0].utilization() + threadSet[i].utilization() < 1 && cpu_u_0 <= cpu_u_1 && cpu_u_0 <= cpu_u_2 && cpu_u_0 <= cpu_u_3) { threadSet[i].setCPUAffinityMask(0); cpuSet[0].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[1].utilization() + threadSet[i].utilization() < 1 && cpu_u_1 <= cpu_u_0 && cpu_u_1 <= cpu_u_2 && cpu_u_1 <= cpu_u_3) { threadSet[i].setCPUAffinityMask(1); cpuSet[1].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[2].utilization() + threadSet[i].utilization() < 1 && cpu_u_2 <= cpu_u_0 && cpu_u_2 <= cpu_u_1 && cpu_u_2 <= cpu_u_3) { threadSet[i].setCPUAffinityMask(2); cpuSet[2].pushThreadToCPU(&threadSet[i]); }
        else if (cpuSet[3].utilization() + threadSet[i].utilization() < 1 && cpu_u_3 <= cpu_u_0 && cpu_u_3 <= cpu_u_1 && cpu_u_3 <= cpu_u_2) { threadSet[i].setCPUAffinityMask(3); cpuSet[3].pushThreadToCPU(&threadSet[i]); }
        pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]);
    }
    for (int i = 0; i < numThread; i++) { pthread_join(threadSet[i].pthreadThread, NULL); }
    /*~~~~~END~~~~~*/
    setEndTime();
    std::cout << "Partition Multi Thread Spend time : " << _timeSec << std::endl;
    cleanMultiResult();
    //partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}

```

WF：如上圖所示，我先記錄現在所有 core 的 Utilization，逐一計算單次的 thread 是否可以放入 core0~3 中，如果符合 Utilization<1 則放置於 core 的 Utilization 最小的 core 中。

- 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)

1. using input part2_Input_10.txt

```

=====Partition First-Fit Multi Thread Matrix Multiplication=====
Thread-9 is no schedulable
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

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

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

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

Thread ID : 1   PID : 3120   Core : 0   Utilization : 0.346   MatrixSize : 692
Thread ID : 3   PID : 3122   Core : 1   Utilization : 0.34    MatrixSize : 680
Thread ID : 2   PID : 3121   Core : 1   Utilization : 0.389   MatrixSize : 778
Thread ID : 0   PID : 3119   Core : 0   Utilization : 0.38    MatrixSize : 760
Thread ID : 5   PID : 3124   Core : 2   Utilization : 0.344   MatrixSize : 688
Thread ID : 9   PID : 3128   Core : 3   Utilization : 0.373   MatrixSize : 746
Thread ID : 6   PID : 3125   Core : 2   Utilization : 0.3065  MatrixSize : 613
Thread ID : 8   PID : 3127   Core : 3   Utilization : 0.397   MatrixSize : 794
Thread ID : 4   PID : 3123   Core : 0   Utilization : 0.2665  MatrixSize : 533
Thread ID : 7   PID : 3126   Core : 3   Utilization : 0.367   MatrixSize : 734
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 9.03346

```

```

=====Partition Best-Fit Multi Thread Matrix Multiplication=====
Thread-9 is no schedulable
Core Number : 0
[ 0, 1, ]
Total Utilization : 0.726

Core Number : 1
[ 2, 3, 4, ]
Total Utilization : 0.9955

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

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

Thread ID : 1   PID : 3131   Core : 0   Utilization : 0.346   MatrixSize : 692
Thread ID : 4   PID : 3134   Core : 1   Utilization : 0.2665   MatrixSize : 533
Thread ID : 5   PID : 3135   Core : 2   Utilization : 0.344   MatrixSize : 688
Thread ID : 9   PID : 3139   Core : 3   Utilization : 0.373   MatrixSize : 746
Thread ID : 3   PID : 3133   Core : 1   Utilization : 0.34   MatrixSize : 680
Thread ID : 6   PID : 3136   Core : 2   Utilization : 0.3065   MatrixSize : 613
Thread ID : 8   PID : 3138   Core : 3   Utilization : 0.397   MatrixSize : 794
Thread ID : 0   PID : 3130   Core : 0   Utilization : 0.38   MatrixSize : 760
Thread ID : 7   PID : 3137   Core : 3   Utilization : 0.367   MatrixSize : 734
Thread ID : 2   PID : 3132   Core : 1   Utilization : 0.389   MatrixSize : 778
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.88796

```

```

=====Partition Worst-Fit Multi Thread Matrix Multiplication=====
Thread-8 is no schedulable
Core Number : 0
[ 0, 6, ]
Total Utilization : 0.6865

Core Number : 1
[ 1, 5, ]
Total Utilization : 0.69

Core Number : 2
[ 2, 7, ]
Total Utilization : 0.756

Core Number : 3
[ 3, 4, 9, ]
Total Utilization : 0.9795

Thread ID : 0   PID : 3140   Core : 0   Utilization : 0.38   MatrixSize : 760
Thread ID : 1   PID : 3141   Core : 1   Utilization : 0.346   MatrixSize : 692
Thread ID : 2   PID : 3142   Core : 2   Utilization : 0.389   MatrixSize : 778
Thread ID : 4   PID : 3144   Core : 3   Utilization : 0.2665   MatrixSize : 533
Thread ID : 3   PID : 3143   Core : 3   Utilization : 0.34   MatrixSize : 680
Thread ID : 8   PID : 3148   Core : 2   Utilization : 0.397   MatrixSize : 794
Thread ID : 5   PID : 3145   Core : 1   Utilization : 0.344   MatrixSize : 688
Thread ID : 6   PID : 3146   Core : 0   Utilization : 0.3065   MatrixSize : 613
Thread ID : 9   PID : 3149   Core : 3   Utilization : 0.373   MatrixSize : 746
Thread ID : 7   PID : 3147   Core : 2   Utilization : 0.367   MatrixSize : 734
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 9.17527

```

2. using input part2_Input_20.txt

```
=====Partition First-Fit Multi Thread Matrix Multiplication=====
Thread-19 is no schedulable
Core Number : 0
[ 0, 1, 2, 3, 4, 5, 7, 9, ]
Total Utilization : 0.937

Core Number : 1
[ 6, 8, 10, ]
Total Utilization : 0.9125

Core Number : 2
[ 11, 12, 13, 14, 16, ]
Total Utilization : 0.857

Core Number : 3
[ 15, 17, 18, ]
Total Utilization : 0.9695

Thread ID : 5   PID : 3321   Core : 0   Utilization : 0.296   MatrixSize : 592
Thread ID : 6   PID : 3322   Core : 1   Utilization : 0.3465   MatrixSize : 693
Thread ID : 4   PID : 3320   Core : 0   Utilization : 0.12     MatrixSize : 240
Thread ID : 3   PID : 3319   Core : 0   Utilization : 0.08     MatrixSize : 160
Thread ID : 2   PID : 3318   Core : 0   Utilization : 0.08     MatrixSize : 160
Thread ID : 8   PID : 3324   Core : 1   Utilization : 0.233    MatrixSize : 466
Thread ID : 7   PID : 3323   Core : 0   Utilization : 0.05     MatrixSize : 100
Thread ID : 1   PID : 3317   Core : 0   Utilization : 0.16     MatrixSize : 320
Thread ID : 9   PID : 3325   Core : 0   Utilization : 0.131    MatrixSize : 262
Thread ID : 10  PID : 3326   Core : 1   Utilization : 0.333    MatrixSize : 666
Thread ID : 11  PID : 3327   Core : 2   Utilization : 0.272    MatrixSize : 544
Thread ID : 15  PID : 3331   Core : 3   Utilization : 0.29     MatrixSize : 580
Thread ID : 14  PID : 3330   Core : 2   Utilization : 0.116    MatrixSize : 232
Thread ID : 12  PID : 3328   Core : 2   Utilization : 0.241    MatrixSize : 482
Thread ID : 13  PID : 3329   Core : 2   Utilization : 0.128    MatrixSize : 256
Thread ID : 19  PID : 3335   Core : 3   Utilization : 0.1825   MatrixSize : 365
Thread ID : 0   PID : 3316   Core : 0   Utilization : 0.02     MatrixSize : 40
Thread ID : 18  PID : 3334   Core : 3   Utilization : 0.333    MatrixSize : 666
Thread ID : 16  PID : 3332   Core : 2   Utilization : 0.1      MatrixSize : 200
Thread ID : 17  PID : 3333   Core : 3   Utilization : 0.3465   MatrixSize : 693
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 6.3026
```



```

=====Partition Best-Fit Multi Thread Matrix Multiplication=====
Thread-19 is no schedulable
Core Number : 0
[ 0, 1, 2, 3, 4, 5, 7, 9, ]
Total Utilization : 0.937

Core Number : 1
[ 6, 8, 10, ]
Total Utilization : 0.9125

Core Number : 2
[ 11, 12, 13, 14, 16, ]
Total Utilization : 0.857

Core Number : 3
[ 15, 17, 18, ]
Total Utilization : 0.9695

Thread ID : 5   PID : 3279   Core : 0   Utilization : 0.296   MatrixSize : 592
Thread ID : 6   PID : 3280   Core : 1   Utilization : 0.3465   MatrixSize : 693
Thread ID : 4   PID : 3278   Core : 0   Utilization : 0.12     MatrixSize : 240
Thread ID : 3   PID : 3277   Core : 0   Utilization : 0.08     MatrixSize : 160
Thread ID : 14  PID : 3288   Core : 2   Utilization : 0.116    MatrixSize : 232
Thread ID : 15  PID : 3289   Core : 3   Utilization : 0.29     MatrixSize : 580
Thread ID : 2   PID : 3276   Core : 0   Utilization : 0.08     MatrixSize : 160
Thread ID : 8   PID : 3282   Core : 1   Utilization : 0.233    MatrixSize : 466
Thread ID : 12  PID : 3286   Core : 2   Utilization : 0.241    MatrixSize : 482
Thread ID : 16  PID : 3290   Core : 2   Utilization : 0.1      MatrixSize : 200
Thread ID : 19  PID : 3293   Core : 3   Utilization : 0.1825   MatrixSize : 365
Thread ID : 9   PID : 3283   Core : 0   Utilization : 0.131    MatrixSize : 262
Thread ID : 18  PID : 3292   Core : 3   Utilization : 0.333    MatrixSize : 666
Thread ID : 10  PID : 3284   Core : 1   Utilization : 0.333    MatrixSize : 666
Thread ID : 7   PID : 3281   Core : 0   Utilization : 0.05     MatrixSize : 100
Thread ID : 17  PID : 3291   Core : 3   Utilization : 0.3465   MatrixSize : 693
Thread ID : 13  PID : 3287   Core : 2   Utilization : 0.128    MatrixSize : 256
Thread ID : 1   PID : 3275   Core : 0   Utilization : 0.16     MatrixSize : 320
Thread ID : 11  PID : 3285   Core : 2   Utilization : 0.272    MatrixSize : 544
Thread ID : 0   PID : 3274   Core : 0   Utilization : 0.02     MatrixSize : 40

Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 5.57709

```



```

=====Partition Worst-Fit Multi Thread Matrix Multiplication=====
Thread-17 is no schedulable
Core Number : 0
[ 0, 4, 7, 9, 10, 18, ]
Total Utilization : 0.987

Core Number : 1
[ 1, 8, 12, 15, ]
Total Utilization : 0.924

Core Number : 2
[ 2, 5, 11, 16, ]
Total Utilization : 0.748

Core Number : 3
[ 3, 6, 13, 14, 19, ]
Total Utilization : 0.853

Thread ID : 0   PID : 3294   Core : 0   Utilization : 0.02   MatrixSize : 40
Thread ID : 1   PID : 3295   Core : 1   Utilization : 0.16   MatrixSize : 320
Thread ID : 2   PID : 3296   Core : 2   Utilization : 0.08   MatrixSize : 160
Thread ID : 3   PID : 3297   Core : 3   Utilization : 0.08   MatrixSize : 160
Thread ID : 4   PID : 3298   Core : 0   Utilization : 0.12   MatrixSize : 240
Thread ID : 8   PID : 3302   Core : 1   Utilization : 0.233   MatrixSize : 466
Thread ID : 6   PID : 3300   Core : 3   Utilization : 0.3465   MatrixSize : 693
Thread ID : 5   PID : 3299   Core : 2   Utilization : 0.296   MatrixSize : 592
Thread ID : 7   PID : 3301   Core : 0   Utilization : 0.05   MatrixSize : 100
Thread ID : 11  PID : 3305   Core : 2   Utilization : 0.272   MatrixSize : 544
Thread ID : 10  PID : 3304   Core : 0   Utilization : 0.333   MatrixSize : 666
Thread ID : 14  PID : 3308   Core : 3   Utilization : 0.116   MatrixSize : 232
Thread ID : 9   PID : 3303   Core : 0   Utilization : 0.131   MatrixSize : 262
Thread ID : 17  PID : 3311   Core : 2   Utilization : 0.3465   MatrixSize : 693
Thread ID : 13  PID : 3307   Core : 3   Utilization : 0.128   MatrixSize : 256
Thread ID : 18  PID : 3312   Core : 0   Utilization : 0.333   MatrixSize : 666
Thread ID : 16  PID : 3310   Core : 2   Utilization : 0.1   MatrixSize : 200
Thread ID : 15  PID : 3309   Core : 1   Utilization : 0.29   MatrixSize : 580
Thread ID : 19  PID : 3313   Core : 3   Utilization : 0.1825   MatrixSize : 365
Thread ID : 12  PID : 3306   Core : 1   Utilization : 0.241   MatrixSize : 482
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 4.7383

```

● Part 3 [Scheduler Implementation. 10%]

- Describe how to implement the scheduler setting in partition scheduling. (FIFO with FF, RR with FF) [Result. 10%]

1. FIFO with FF

```
=====Partition First-Fit Multi Thread Matrix Multiplication=====
Thread-9 is no schedulable
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

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

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

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

Core0 start PID - 5612
Core0 context switch from PID - 5612 to PID - 5613
Core0 context switch from PID - 5613 to PID - 5616
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 13.6761
```

2. RR with FF

```
=====Partition First-Fit Multi Thread Matrix Multiplication=====
Thread-9 is no schedulable
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

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

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

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

Core0 start PID - 5670
Core0 context switch from PID - 5670 to PID - 5671
Core0 context switch from PID - 5671 to PID - 5674
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 12.2593
```

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

方法一：紀錄所有放到 core0 的所有 thread 再 print 出來

```
int aaa = 0; //紀錄上一個PID
int account = 0; //紀錄進入core0的計數器
int aTID[10] = { 0,0,0,0,0,0,0,0,0,0 }; //紀錄有進入core0 的所有thread number
for (int i = 0; i < numThread; i++) {
    if (cpuSet[0].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPAffinityMask(0); cpuSet[0].pushThreadToCPU(&threadSet[i]); aTID[account] = i; account++; }
    else if (cpuSet[1].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPAffinityMask(1); cpuSet[1].pushThreadToCPU(&threadSet[i]); }
    else if (cpuSet[2].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPAffinityMask(2); cpuSet[2].pushThreadToCPU(&threadSet[i]); }
    else if (cpuSet[3].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPAffinityMask(3); cpuSet[3].pushThreadToCPU(&threadSet[i]); }
    pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]);
}
}
#if (PART == 3)
for (int i = 0; i < numThread; i++) {
    if (aTID[i] != 0) {
        //逐一列印core0 的 PID 交換過程
        std::cout << "Core0 context switch from PID - " << aaa << " to PID - " << threadSet[aTID[i]].PID_self() << std::endl; aaa = threadSet[aTID[i]].PID_self();
    }
    else if (i == 0) { std::cout << "Core0 start PID - " << threadSet[aTID[0]].PID_self() << std::endl; aaa = threadSet[aTID[0]].PID_self(); } //直接列印第一筆資料
}
#endif
#endif
```

方法二：逐一執行 print 放入 core0 內的 thread

```
int bbb = 0; //紀錄上一個PID
int bbbb = 0; //紀錄逐一thread是否進入core0
for (int i = 0; i < numThread; i++) {
    float cpu_U_0 = cpuSet[0].utilization();
    float cpu_U_1 = cpuSet[1].utilization();
    float cpu_U_2 = cpuSet[2].utilization();
    float cpu_U_3 = cpuSet[3].utilization();
    if (cpuSet[0].utilization() + threadSet[i].utilization() < 1 && cpu_U_0 >= cpu_U_1 && cpu_U_0 >= cpu_U_2 && cpu_U_0 >= cpu_U_3) { threadSet[i].setUpCPAffinityMask(0); cpuSet[0].pushThreadToCPU(&threadSet[i]); bbbb = 1; }
    else if (cpuSet[1].utilization() + threadSet[i].utilization() < 1 && cpu_U_1 >= cpu_U_2 && cpu_U_1 >= cpu_U_3) { threadSet[i].setUpCPAffinityMask(1); cpuSet[1].pushThreadToCPU(&threadSet[i]); bbbb = 0; }
    else if (cpuSet[2].utilization() + threadSet[i].utilization() < 1 && cpu_U_2 >= cpu_U_3) { threadSet[i].setUpCPAffinityMask(2); cpuSet[2].pushThreadToCPU(&threadSet[i]); bbbb = 0; }
    else if (cpuSet[3].utilization() + threadSet[i].utilization() < 1) { threadSet[i].setUpCPAffinityMask(3); cpuSet[3].pushThreadToCPU(&threadSet[i]); bbbb = 0; }
    pthread_create(&threadSet[i].pthreadThread, NULL, threadSet[i].matrixMultiplication, &threadSet[i]);
    if (bbbb == 1) {
        #if (PART == 3)
        if (bbb == 0) { std::cout << "Core0 start PID - " << threadSet[i].PID_self() << std::endl; } //直接列印第一筆資料
        else { std::cout << "Core0 context switch from PID - " << bbb << " to PID - " << threadSet[i].PID_self() << std::endl; } //逐一列印core0 的 PID 交換過程
        bbb = threadSet[i].PID_self(); //更新PID
    }
}
#endif
```

FIFO 與 RR 方法：

```
void
Thread::setUpScheduler()
{
    /*~~~~~Your code(PART3)~~~~~*/
    // Set up the scheduler for current thread 為當前線程設置調度程序
    #include <sched.h>
    struct sched_param sp;
    #if (SCHEDULING == SCHED_RR)
        sp.sched_priority = sched_get_priority_max(SCHED_RR);
        int ret = sched_setscheduler(0, SCHED_RR, &sp);
    #endif
    #if (SCHEDULING == SCHED_FIFO)
        sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
        int ret = sched_setscheduler(0, SCHED_FIFO, &sp);
    #endif

    /*~~~~~END~~~~~*/
}
```

● Discussion

- Analyze and compare the response time of the program, with single thread and multi-thread using in part1 and part2. (Including Single, Global, FirstFit, Best-Fit, Worst-Fit) 10%

	Single	Global	FirstFit	Best-Fit	Worst-Fit
--	--------	--------	----------	----------	-----------

responsetime	127.419	38.9793	9.03346	8.88796	9.17527
--------------	---------	---------	---------	---------	---------

我們可以從實驗數據得知，單執行序一定會比多執行序還要花時間。本次的例子所使用的核芯數量最大為 4 核，故多執行序所花費的時間為單執行序的四分之一。再來討論後續的 FirstFit, Best-Fit, Worst-Fit 這三種，這三種多執行序皆是先將 Thread 分配好再執行，從而得知這三種多執行序會比一般的多執行序 (Global) 還要快完成。我們還可以利用上表所示的數據，間接的證明的這三種多執行序的英文名稱由來。

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

	FIFO with FF	RR with FF
response time	12.8467	12.5775

FIFO (First Input First Output)簡單說就是指先進先出；RR(Round-robin) 通常指將多個某物輪流用於某事。在用於此系統中 RR 是將 thread 分成相同大小，這樣有利於傳輸至 BUS 的等候時間，因為在 FIFO(沒有切割)的時候，需要先等候該 thread 執行完之後才會接上下一個 thread；然而在 RR 中我們可以大幅減少在兩個(或多個)不同 core 中的 thread 傳輸至 BUS 的時間。從實驗結果也可以證明將 thread 切割之後會比沒有切割還要來的快，但是由於本次實驗中的 thread 的 Utilization 較小，如果有較大 Utilization 的 thread 放入此系統，則會發現明顯的差異。