嵌入式系統軟體設計

Embedded System Software Design

PA1

指導教授: 陳雅淑 教授

課程學生: M10907314 張祐銓

● Part 1

[Global Scheduling. 10%]

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

設定每個 Thread 個別要計算哪幾列矩陣，以平均分派的方式，因為是四個 Thread 所以各被分派 500 列。

```
System::System(char* input_file)
{
    loadInput(input_file); // Set up threadSet, singleResult, multiResult, and matrix

    for (int i = 0; i<numThread; i++) {
#if (PART == 1)
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[0], multiResult[0], matrix[0]);
        /*~~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~~~*/
        // Set up the calculate range of matrix.
        threadSet[i].setStartCalculatePoint(threadSet[i].matrixSize()/numThread*(i));
        threadSet[i].setEndCalculatePoint(threadSet[i].matrixSize()/numThread*(i+1));
        /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~*/
#else
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[i], multiResult[i], matrix[i]);
#endif
```

使用 pthread_create 創建每個 Thread 執行，並對每個 Thread 做 join 等待每個 Thread 都執行完畢。

```
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);
    }
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

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

在運算函式中如果有設定綁定 Core 的話就執行 setUpCPUAffinityMask，剛進來時就先抓取目前在哪個 core 上及 PID 號碼並顯示資訊。

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

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

    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    // Set up the affinity mask
    if(obj->setCore != -1)
        obj->setUpCPUAffinityMask(obj->setCore);
    obj->core = sched_getcpu();
    obj->PID = syscall(SYS_gettid);

    if(PART != 3){
        pthread_mutex_lock( &count_Mutex );
        obj->printInformation();
        pthread_mutex_unlock( &count_Mutex );
    }
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    /* 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()){
                if(PART == 1)
                    printf("The thread %d PID %d is moved from CPU %d to %d\n",
                        obj->ID(), obj->PID, obj->core, sched_getcpu());
                obj->core = sched_getcpu();
            }
            /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

- Describe how to observe task migration. 5%

在上圖的 matrixMultiplication 在運算矩陣時還會隨時去查看目前被哪個 core 執行，假如與上個紀錄的 core 不同就顯示目前工作情況。

[Partition Scheduling. 5%]

- Describe how to implement partition scheduling by using pthread.

setUpCPUAffinityMask 會透過 sched_setaffinity 這個指令將目前的 thread 綁定在指定的 cpu_num 的 Core 上。

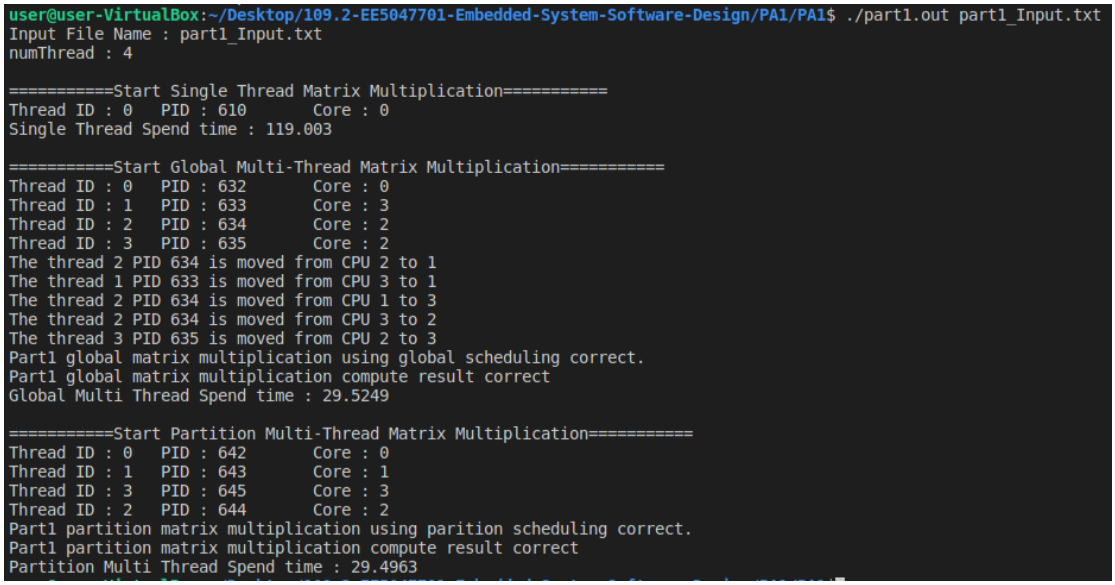
```
void
Thread::setUpCPUAffinityMask(int cpu_num)
{
    /*~~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~~*/
    // Pined the thread to core.
    if(cpu_num == -1)
        return;
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(cpu_num,&cpuset);
    sched_setaffinity(0, sizeof(cpu_set_t), &cpuset);
    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~*/
}
```

使用 pthread_create 創建每個 Thread 執行，並對每個 Thread 做 join 等待每個 Thread 都執行完畢，與 global 不同在於要預先設定想要綁在哪個 Core 上。

```
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){
        if(PART == 1) threadSet[i].setThreadCore(i % CORE_NUM);
        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();
}
```

[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.

在 First-Fit 中將每個 Thread task 拿去跟每個 core 去做判斷，假如 utilization 相加後小於等於 1 的話就安排到這個 core 內，從 index 比較小的 core 開始判斷，假如每個 core 都排不進的話顯示無法排入的資訊。

```
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 parititon first-fit and print result.
    for(int i = 0 ; i < numThread ; ++i){
        bool isArrange = false;
        for(int j = 0 ; j < CORE_NUM ; ++j){
            if(cpuSet[j].utilization() + threadSet[i].utilization() <= 1){
                cpuSet[j].pushThreadToCPU(&threadSet[i]);
                threadSet[i].setThreadCore(j);
                isArrange = true;
                break;
            }
        }
        if(!isArrange){
            printf("Thread-%d not schedulable\n", threadSet[i].ID());
        }
    }
    for(int i = 0; i < CORE_NUM; ++i){
        cpuSet[i].printCPUInformation();
    }
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

在 Best-Fit 中，要選擇 utilization 最高的 core 把工作加進去，先將 Thread task 拿去跟每個 core 去做判斷，假如這個 core 無法放入這個 task 就查看下一個，如果這個 core 能容下這個 task 且比之前所記錄的最大的那個 core 的 utilization 還高的話就記錄下最大的為這個 core，每個 core 都查看過一次後再進行一次確認是否能放下這個 task，假如不行就顯示無法排入的資訊。

```
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 best-fit and print result.
    for(int i = 0 ; i < numThread ; ++i){
        bool isArrange = false;
        int maxiCore = 0;
        for(int j = 0 ; j < CORE_NUM ; ++j){
            if(cpuSet[maxiCore].utilization() + threadSet[i].utilization() > 1){
                maxiCore = j;
            }
            else if(cpuSet[j].utilization() > cpuSet[maxiCore].utilization()
                && cpuSet[j].utilization() + threadSet[i].utilization() <= 1){
                maxiCore = j;
            }
        }
        if(cpuSet[maxiCore].utilization() + threadSet[i].utilization() <= 1){
            cpuSet[maxiCore].pushThreadToCPU(&threadSet[i]);
            threadSet[i].setThreadCore(maxiCore);
            isArrange = true;
        }
        if(!isArrange){
            printf("Thread-%d not schedulable\n", threadSet[i].ID());
        }
    }
    for(int i = 0; i < CORE_NUM; ++i){
        cpuSet[i].printCPUInformation();
    }
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
```

在 Worst-Fit 中，要選擇 utilization 最低的 core 把工作加進去，比較簡單的是只要查看每個 core 的 utilization 並從中選擇出最低的那個，並判斷是否能夠將 task 排進去。

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

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

    /*~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
    // Implement partition worst-fit and print result.
    for(int i = 0 ; i < numThread ; ++i){
        bool isArrange = false;
        int miniCore = 0;
        for(int j = 0 ; j < CORE_NUM ; ++j){
            if(cpuSet[j].utilization() < cpuSet[miniCore].utilization()){
                miniCore = j;
            }
        }
        if(cpuSet[miniCore].utilization() + threadSet[i].utilization() <= 1){
            cpuSet[miniCore].pushThreadToCPU(&threadSet[i]);
            threadSet[i].setThreadCore(miniCore);
            isArrange = true;
        }
        if(!isArrange){
            printf("Thread-%d not schedulable\n", threadSet[i].ID());
        }
    }
    for(int i = 0; i < CORE_NUM; ++i){
        cpuSet[i].printCPUInformation();
    }
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

[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)

./part2.out part2_Input_10.txt 的結果

```
user@user-VirtualBox:~/Desktop/109.2-EE5047701-Embedded-System-Software-Design/PA1/PA1$ ./part2.out ./part2_Input_10.txt
Input File Name : ./part2_Input_10.txt
numThread : 10

==========Start Single Thread Matrix Multiplication==========
Thread ID : 0   PID : 1135      Core : 3        Utilization : 0.38      MatrixSize : 760
Thread ID : 1   PID : 1135      Core : 3        Utilization : 0.346     MatrixSize : 692
Thread ID : 2   PID : 1135      Core : 3        Utilization : 0.389     MatrixSize : 778
Thread ID : 3   PID : 1135      Core : 3        Utilization : 0.34      MatrixSize : 680
Thread ID : 4   PID : 1135      Core : 3        Utilization : 0.2665    MatrixSize : 533
Thread ID : 5   PID : 1135      Core : 3        Utilization : 0.344     MatrixSize : 688
Thread ID : 6   PID : 1135      Core : 3        Utilization : 0.3065    MatrixSize : 613
Thread ID : 7   PID : 1135      Core : 3        Utilization : 0.367     MatrixSize : 734
Thread ID : 8   PID : 1135      Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 9   PID : 1135      Core : 3        Utilization : 0.373     MatrixSize : 746
Single Thread Spend time : 22.1305

==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-9 not 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 : 0   PID : 1145      Core : 0        Utilization : 0.38      MatrixSize : 760
Thread ID : 2   PID : 1147      Core : 1        Utilization : 0.389     MatrixSize : 778
Thread ID : 7   PID : 1152      Core : 3        Utilization : 0.367     MatrixSize : 734
Thread ID : 5   PID : 1150      Core : 2        Utilization : 0.344     MatrixSize : 688
Thread ID : 1   PID : 1146      Core : 0        Utilization : 0.346     MatrixSize : 692
Thread ID : 6   PID : 1151      Core : 2        Utilization : 0.3065    MatrixSize : 613
Thread ID : 9   PID : 1154      Core : 0        Utilization : 0.373     MatrixSize : 746
Thread ID : 3   PID : 1148      Core : 1        Utilization : 0.34      MatrixSize : 680
Thread ID : 8   PID : 1153      Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 4   PID : 1149      Core : 0        Utilization : 0.2665    MatrixSize : 533
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.07153
```

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-9 not 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 : 5   PID : 1162      Core : 2        Utilization : 0.344     MatrixSize : 688
Thread ID : 0   PID : 1157      Core : 0        Utilization : 0.38      MatrixSize : 760
Thread ID : 3   PID : 1160      Core : 1        Utilization : 0.34      MatrixSize : 680
Thread ID : 7   PID : 1164      Core : 3        Utilization : 0.367     MatrixSize : 734
Thread ID : 4   PID : 1161      Core : 1        Utilization : 0.2665    MatrixSize : 533
Thread ID : 6   PID : 1163      Core : 2        Utilization : 0.3065    MatrixSize : 613
Thread ID : 1   PID : 1158      Core : 0        Utilization : 0.346     MatrixSize : 692
Thread ID : 9   PID : 1166      Core : 2        Utilization : 0.373     MatrixSize : 746
Thread ID : 8   PID : 1165      Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 2   PID : 1159      Core : 1        Utilization : 0.389     MatrixSize : 778
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 6.622
```

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-8 not 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 : 2   PID : 1170      Core : 2        Utilization : 0.389     MatrixSize : 778
Thread ID : 0   PID : 1168      Core : 0        Utilization : 0.38      MatrixSize : 760
Thread ID : 3   PID : 1171      Core : 3        Utilization : 0.34      MatrixSize : 680
Thread ID : 7   PID : 1175      Core : 2        Utilization : 0.367     MatrixSize : 734
Thread ID : 6   PID : 1174      Core : 0        Utilization : 0.3065    MatrixSize : 613
Thread ID : 5   PID : 1173      Core : 1        Utilization : 0.344     MatrixSize : 688
Thread ID : 1   PID : 1169      Core : 1        Utilization : 0.346     MatrixSize : 692
Thread ID : 9   PID : 1177      Core : 3        Utilization : 0.373     MatrixSize : 746
Thread ID : 8   PID : 1176      Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 4   PID : 1172      Core : 3        Utilization : 0.2665    MatrixSize : 533
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.75248
```

./part2.out part2_Input_20.txt 的結果

```
user@user-VirtualBox:~/Desktop/109.2-EE5047701-Embedded-System-Software-Design/PA1/PA1$ ./part2.out ./part2_Input_20.txt
Input File Name : ./part2_Input_20.txt
numThread : 20

==========Start Single Thread Matrix Multiplication==========
Thread ID : 0   PID : 2174      Core : 3        Utilization : 0.02      MatrixSize : 40
Thread ID : 1   PID : 2174      Core : 3        Utilization : 0.16      MatrixSize : 320
Thread ID : 2   PID : 2174      Core : 3        Utilization : 0.08      MatrixSize : 160
Thread ID : 3   PID : 2174      Core : 3        Utilization : 0.08      MatrixSize : 160
Thread ID : 4   PID : 2174      Core : 3        Utilization : 0.12      MatrixSize : 240
Thread ID : 5   PID : 2174      Core : 3        Utilization : 0.296     MatrixSize : 592
Thread ID : 6   PID : 2174      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 7   PID : 2174      Core : 3        Utilization : 0.05      MatrixSize : 100
Thread ID : 8   PID : 2174      Core : 3        Utilization : 0.233     MatrixSize : 466
Thread ID : 9   PID : 2174      Core : 3        Utilization : 0.131     MatrixSize : 262
Thread ID : 10  PID : 2174      Core : 3        Utilization : 0.333     MatrixSize : 666
Thread ID : 11  PID : 2174      Core : 3        Utilization : 0.272     MatrixSize : 544
Thread ID : 12  PID : 2174      Core : 3        Utilization : 0.241     MatrixSize : 482
Thread ID : 13  PID : 2174      Core : 3        Utilization : 0.128     MatrixSize : 256
Thread ID : 14  PID : 2174      Core : 3        Utilization : 0.116     MatrixSize : 232
Thread ID : 15  PID : 2174      Core : 3        Utilization : 0.29      MatrixSize : 580
Thread ID : 16  PID : 2174      Core : 3        Utilization : 0.1       MatrixSize : 200
Thread ID : 17  PID : 2174      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 18  PID : 2174      Core : 3        Utilization : 0.333     MatrixSize : 666
Thread ID : 19  PID : 2174      Core : 3        Utilization : 0.1825    MatrixSize : 365
Single Thread Spend time : 11.8865
```

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-19 not 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 : 0   PID : 2178      Core : 0        Utilization : 0.02      MatrixSize : 40
Thread ID : 8   PID : 2186      Core : 1        Utilization : 0.233     MatrixSize : 466
Thread ID : 12  PID : 2190      Core : 2        Utilization : 0.241     MatrixSize : 482
Thread ID : 3   PID : 2181      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 17  PID : 2195      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 6   PID : 2184      Core : 1        Utilization : 0.3465    MatrixSize : 693
Thread ID : 10  PID : 2188      Core : 1        Utilization : 0.333     MatrixSize : 666
Thread ID : 14  PID : 2192      Core : 2        Utilization : 0.116     MatrixSize : 232
Thread ID : 5   PID : 2183      Core : 0        Utilization : 0.296     MatrixSize : 592
Thread ID : 15  PID : 2193      Core : 3        Utilization : 0.29      MatrixSize : 580
Thread ID : 7   PID : 2185      Core : 0        Utilization : 0.05      MatrixSize : 100
Thread ID : 4   PID : 2182      Core : 0        Utilization : 0.12      MatrixSize : 240
Thread ID : 11  PID : 2189      Core : 2        Utilization : 0.272     MatrixSize : 544
Thread ID : 1   PID : 2179      Core : 0        Utilization : 0.16      MatrixSize : 320
Thread ID : 2   PID : 2180      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 13  PID : 2191      Core : 2        Utilization : 0.128     MatrixSize : 256
Thread ID : 9   PID : 2187      Core : 0        Utilization : 0.131     MatrixSize : 262
Thread ID : 19  PID : 2197      Core : 1        Utilization : 0.1825    MatrixSize : 365
Thread ID : 18  PID : 2196      Core : 3        Utilization : 0.333     MatrixSize : 666
Thread ID : 16  PID : 2194      Core : 2        Utilization : 0.1       MatrixSize : 200
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 5.41601
```

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-19 not 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 : 0   PID : 2201      Core : 0        Utilization : 0.02      MatrixSize : 40
Thread ID : 10  PID : 2211      Core : 1        Utilization : 0.333     MatrixSize : 666
Thread ID : 5   PID : 2206      Core : 0        Utilization : 0.296     MatrixSize : 592
Thread ID : 19  PID : 2220      Core : 3        Utilization : 0.1825    MatrixSize : 365
Thread ID : 3   PID : 2204      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 7   PID : 2208      Core : 0        Utilization : 0.05      MatrixSize : 100
Thread ID : 6   PID : 2207      Core : 1        Utilization : 0.3465    MatrixSize : 693
Thread ID : 8   PID : 2209      Core : 1        Utilization : 0.233     MatrixSize : 466
Thread ID : 9   PID : 2210      Core : 0        Utilization : 0.131     MatrixSize : 262
Thread ID : 1   PID : 2202      Core : 0        Utilization : 0.16      MatrixSize : 320
Thread ID : 11  PID : 2212      Core : 2        Utilization : 0.272     MatrixSize : 544
Thread ID : 18  PID : 2219      Core : 3        Utilization : 0.333     MatrixSize : 666
Thread ID : 12  PID : 2213      Core : 2        Utilization : 0.241     MatrixSize : 482
Thread ID : 13  PID : 2214      Core : 2        Utilization : 0.128     MatrixSize : 256
Thread ID : 17  PID : 2218      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 4   PID : 2205      Core : 0        Utilization : 0.12      MatrixSize : 240
Thread ID : 16  PID : 2217      Core : 2        Utilization : 0.1       MatrixSize : 200
Thread ID : 2   PID : 2203      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 14  PID : 2215      Core : 2        Utilization : 0.116     MatrixSize : 232
Thread ID : 15  PID : 2216      Core : 3        Utilization : 0.29      MatrixSize : 580
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 5.42999
```

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-17 not 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 : 2   PID : 2224      Core : 2        Utilization : 0.08      MatrixSize : 160
Thread ID : 1   PID : 2223      Core : 1        Utilization : 0.16      MatrixSize : 320
Thread ID : 6   PID : 2228      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 0   PID : 2222      Core : 0        Utilization : 0.02      MatrixSize : 40
Thread ID : 4   PID : 2226      Core : 0        Utilization : 0.12      MatrixSize : 240
Thread ID : 5   PID : 2227      Core : 2        Utilization : 0.296     MatrixSize : 592
Thread ID : 8   PID : 2230      Core : 1        Utilization : 0.233     MatrixSize : 466
Thread ID : 12  PID : 2234      Core : 1        Utilization : 0.241     MatrixSize : 482
Thread ID : 15  PID : 2237      Core : 1        Utilization : 0.29      MatrixSize : 580
Thread ID : 9   PID : 2231      Core : 0        Utilization : 0.131     MatrixSize : 262
Thread ID : 13  PID : 2235      Core : 3        Utilization : 0.128     MatrixSize : 256
Thread ID : 17  PID : 2239      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 19  PID : 2241      Core : 3        Utilization : 0.1825    MatrixSize : 365
Thread ID : 14  PID : 2236      Core : 3        Utilization : 0.116     MatrixSize : 232
Thread ID : 16  PID : 2238      Core : 2        Utilization : 0.1       MatrixSize : 200
Thread ID : 11  PID : 2233      Core : 2        Utilization : 0.272     MatrixSize : 544
Thread ID : 3   PID : 2225      Core : 3        Utilization : 0.08      MatrixSize : 160
Thread ID : 10  PID : 2232      Core : 0        Utilization : 0.333     MatrixSize : 666
Thread ID : 18  PID : 2240      Core : 0        Utilization : 0.333     MatrixSize : 666
Thread ID : 7   PID : 2229      Core : 0        Utilization : 0.05      MatrixSize : 100
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 4.91505
```

● Part 3

[Scheduler Implementation. 10%]

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

System 初始化時就會先安排 thread 的 schedule 方式看是 FIFO 還是 RR。

```
System::System(char* input_file)
{
    loadInput(input_file); // Set up threadSet, singleResult, multiResult, and matrix

    for (int i = 0; i<numThread; i++) {
#if (PART == 1)
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[0], multiResult[0], matrix[0]);
        /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~*/
        // Set up the calculate range of matrix.
        threadSet[i].setStartCalculatePoint(threadSet[i].matrixSize()/numThread*(i));
        threadSet[i].setEndCalculatePoint(threadSet[i].matrixSize()/numThread*(i+1));
        /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[i], multiResult[i], matrix[i]);
#endif

#if (PART == 3)
        /*~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
        // Set the scheduling policy for thread.
        if(SCHEDULING == SCHED_FIFO)
            threadSet[i].setSchedulingPolicy(SCHED_FIFO);
        else if(SCHEDULING == SCHED_RR)
            threadSet[i].setSchedulingPolicy(SCHED_RR);
        /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#endif
```

顯示 Core 0 的資訊，假如 current_PID 與讀取的目前 PID 不同就顯示訊息，如果 current_PID 是-1 表示為剛開始。

```
#if (PART == 3)
    /*~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
    // Obaserve the execute thread on core-0
    if(sched_getcpu() == 0){
        if(current_PID != syscall(SYS_gettid)){
            if(current_PID == -1)
                printf("Core0 start PID-%d\n", (int)syscall(SYS_gettid));
            else
                printf("Core0 context switch from PID-%d to PID-%d\n", current_PID, (int)syscall(SYS_gettid));

            current_PID = syscall(SYS_gettid);
        }
    }
    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#endif
    }
```

根據先前已經設定的 schedule 方式，透過 sched_setscheduler 將目前的 thread 修改 schedule 方式。

```
void
Thread::setUpScheduler()
{
    /*~~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~~*/
    // Set up the scheduler for current thread
    if(schedulingPolicy() == SCHED_RR){
        struct sched_param sp;
        sp.sched_priority = sched_get_priority_max(SCHED_RR);
        int ret = sched_setscheduler(0, SCHED_RR, &sp);
    }
    else if(schedulingPolicy() == SCHED_FIFO){
        struct sched_param sp;
        sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
        int ret = sched_setscheduler(0, SCHED_FIFO, &sp);
    }
    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~*/
}
```

[Result. 10%]

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

執行 sudo ./part3_fifo.out part3_Input.txt 的結果

```
user@user-VirtualBox:~/Desktop/109.2-EE5047701-Embedded-System-Software-Design/PA1/PA1$ sudo ./part3_fifo.out part3_Input.txt
Input File Name : part3_Input.txt
numThread : 10

==========Start Single Thread Matrix Multiplication==========
Thread ID : 0   PID : 21600     Core : 2        Utilization : 0.38      MatrixSize : 760
Thread ID : 1   PID : 21600     Core : 3        Utilization : 0.346     MatrixSize : 692
Thread ID : 2   PID : 21600     Core : 3        Utilization : 0.389     MatrixSize : 778
Thread ID : 3   PID : 21600     Core : 3        Utilization : 0.34      MatrixSize : 680
Thread ID : 4   PID : 21600     Core : 3        Utilization : 0.2665    MatrixSize : 533
Thread ID : 5   PID : 21600     Core : 3        Utilization : 0.344     MatrixSize : 688
Thread ID : 6   PID : 21600     Core : 3        Utilization : 0.3065    MatrixSize : 613
Thread ID : 7   PID : 21600     Core : 3        Utilization : 0.367     MatrixSize : 734
Thread ID : 8   PID : 21600     Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 9   PID : 21600     Core : 3        Utilization : 0.373     MatrixSize : 746
Single Thread Spend time : 20.6482

==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-9 not 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-21605
Core0 context switch from PID-21605 to PID-21606
Core0 context switch from PID-21606 to PID-21609
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 5.98818
```

執行 sudo ./part3_rr.out part3_Input.txt 的結果

```
user@user-VirtualBox:~/Desktop/109.2-EE5047701-Embedded-System-Software-Design/PA1/PA1$ sudo ./part3_rr.out part3_Input.txt
Input File Name : part3_Input.txt
numThread : 10

===========Start Single Thread Matrix Multiplication===========
Thread ID : 0   PID : 21559     Core : 2        Utilization : 0.38      MatrixSize : 760
Thread ID : 1   PID : 21559     Core : 2        Utilization : 0.346     MatrixSize : 692
Thread ID : 2   PID : 21559     Core : 2        Utilization : 0.389     MatrixSize : 778
Thread ID : 3   PID : 21559     Core : 2        Utilization : 0.34      MatrixSize : 680
Thread ID : 4   PID : 21559     Core : 2        Utilization : 0.2665    MatrixSize : 533
Thread ID : 5   PID : 21559     Core : 2        Utilization : 0.344     MatrixSize : 688
Thread ID : 6   PID : 21559     Core : 2        Utilization : 0.3065    MatrixSize : 613
Thread ID : 7   PID : 21559     Core : 2        Utilization : 0.367     MatrixSize : 734
Thread ID : 8   PID : 21559     Core : 2        Utilization : 0.397     MatrixSize : 794
Thread ID : 9   PID : 21559     Core : 2        Utilization : 0.373     MatrixSize : 746
Single Thread Spend time : 19.9631

===========Partition First-Fit Multi Thread Matrix Multiplication===========
Thread-9 not 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-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
```

```
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21572
Core0 context switch from PID-21572 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Core0 context switch from PID-21568 to PID-21569
Core0 context switch from PID-21569 to PID-21568
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 6.08656
```

● 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%

Part1:

在 part1 當中會發現每次結果都會差不多 Single 的時間會是 Global 和 Partition 的 4 倍，因為也剛好多工都是被分派一樣的負擔處理資料大小。而 Global 比 Partition 慢了 0.03~3 秒都有，切換次數越多的話差距越大。

Part2:

可以觀察下兩張表得知，觀察出在 Input_10 的時候 Worst-Fit 比 First-Fit 和 Best-Fit 還要慢得多，而在 Input_20 的時候 Worst-Fit 比 First-Fit 和 Best-Fit 還要快一點。

下表示跑 part2_Input.txt 的結果

	Single	First-Fit	Best-Fit	Worst-Fit
1	20.0506	6.24375	6.54249	9.37197
2	21.2568	8.0721	7.16341	9.73308
3	23.1702	7.01627	7.57659	9.84595
4	21.6108	6.78828	7.20736	9.5696
5	21.4864	6.71977	6.8267	9.45477
average	21.51496	6.968034	7.06331	9.595074

下表示跑 part2_Input_txt 的結果

	Single	First-Fit	Best-Fit	Worst-Fit
1	11.8865	5.41601	5.42999	4.91505
2	12.2008	5.93865	5.33299	5.05575
3	12.7978	5.74678	5.56681	5.74835
4	12.8495	5.72912	6.05211	4.79197
5	12.607	5.65019	5.48213	5.2881
average	12.46832	5.69615	5.572806	5.159844

在下兩張圖中是 Worst-Fit 跑 Input_10 的結果，然後把每個 Thread 的執行花費的時間給顯示出來，然後第一張圖是原本的順序從 Core 0 到 Core 3，第二張圖則是從 Core 3 到 Core 0，因為 Thread-8 沒有被排進去所以不觀察，可以發現到說安排在 Core 3 的 Thread 都跑得比較慢，在第二張圖中 Core 1 的 utilization 比較大還是一樣是 Core 3 跑最後，然後 utilization 最大的 Core 1 所有 Thread 的完成時間其跟時同一批次的 First-Fit 和 Best-Fit 完成時間差不多 6.87 秒。

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-8 not 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 : 20145     Core : 0        Utilization : 0.38      MatrixSize : 760
Thread ID : 9   PID : 20154     Core : 3        Utilization : 0.373     MatrixSize : 746
Thread ID : 1   PID : 20146     Core : 1        Utilization : 0.346     MatrixSize : 692
Thread ID : 2   PID : 20147     Core : 2        Utilization : 0.389     MatrixSize : 778
Thread ID : 4   PID : 20149     Core : 3        Utilization : 0.2665    MatrixSize : 533
Thread ID : 3   PID : 20148     Core : 3        Utilization : 0.34      MatrixSize : 680
Thread ID : 6   PID : 20151     Core : 0        Utilization : 0.3065    MatrixSize : 613
Thread ID : 7   PID : 20152     Core : 2        Utilization : 0.367     MatrixSize : 734
Thread ID : 5   PID : 20150     Core : 1        Utilization : 0.344     MatrixSize : 688
Thread ID : 8   PID : 20153     Core : 3        Utilization : 0.397     MatrixSize : 794
Thread: 6, spent: 2.99751
Thread: 4, spent: 4.10593
Thread: 0, spent: 5.01789
Thread: 5, spent: 5.14941
Thread: 1, spent: 5.17946
Thread: 7, spent: 5.8633
Thread: 2, spent: 6.53097
Thread: 3, spent: 7.69443
Thread: 9, spent: 9.09439
Thread: 8, spent: 9.5813
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 9.61231
```

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-8 not schedulable
Core Number : 0
[ 3, 4, 9, ]
Total Utilization : 0.9795

Core Number : 1
[ 2, 7, ]
Total Utilization : 0.756

Core Number : 2
[ 1, 5, ]
Total Utilization : 0.69

Core Number : 3
[ 0, 6, ]
Total Utilization : 0.6865

Thread ID : 0   PID : 20016     Core : 3        Utilization : 0.38      MatrixSize : 760
Thread ID : 1   PID : 20017     Core : 2        Utilization : 0.346     MatrixSize : 692
Thread ID : 6   PID : 20022     Core : 3        Utilization : 0.3065    MatrixSize : 613
Thread ID : 8   PID : 20024     Core : 3        Utilization : 0.397     MatrixSize : 794
Thread ID : 7   PID : 20023     Core : 1        Utilization : 0.367     MatrixSize : 734
Thread ID : 4   PID : 20020     Core : 0        Utilization : 0.2665    MatrixSize : 533
Thread ID : 2   PID : 20018     Core : 1        Utilization : 0.389     MatrixSize : 778
Thread ID : 9   PID : 20025     Core : 0        Utilization : 0.373     MatrixSize : 746
Thread ID : 3   PID : 20019     Core : 0        Utilization : 0.34      MatrixSize : 680
Thread ID : 5   PID : 20021     Core : 2        Utilization : 0.344     MatrixSize : 688
Thread: 4, spent: 2.7489
Thread: 6, spent: 4.87644
Thread: 5, spent: 5.33847
Thread: 1, spent: 5.37905
Thread: 3, spent: 5.67005
Thread: 7, spent: 6.04142
Thread: 9, spent: 6.41146
Thread: 2, spent: 6.87413
Thread: 0, spent: 8.16697
Thread: 8, spent: 8.57662
Part2 partiton result !!WRONG!!
Part2 compute result correct
Partition Multi Thread Spend time : 8.60208
```

在下兩張圖中是跑 Input_20 的結果，因為 Best-Fit 跟 First-Fit 排程結果一樣所以取 Best-Fit 跟 Worst-Fit 做比較，也是能夠發現說 Core 3 完成所有 Thread 的時間還是最慢。

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-19 not 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: 0, spent: 0.000290232
Thread: 7, spent: 0.0486356
Thread: 2, spent: 0.135293
Thread: 3, spent: 0.190984
Thread: 16, spent: 0.231041
Thread: 14, spent: 0.345009
Thread: 13, spent: 0.450793
Thread: 4, spent: 0.476323
Thread: 9, spent: 0.56721
Thread: 1, spent: 0.781114
Thread: 19, spent: 1.32704
Thread: 8, spent: 2.24948
Thread: 12, spent: 2.4223
Thread: 5, spent: 2.83025
Thread: 11, spent: 2.93885
Thread: 15, spent: 4.5596
Thread: 10, spent: 5.86589
Thread: 6, spent: 6.12941
Thread: 18, spent: 6.26425
Thread: 17, spent: 6.57216
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 6.61125
```

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-17 not 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: 0, spent: 0.000330376
Thread: 7, spent: 0.00435572
Thread: 3, spent: 0.11431
Thread: 2, spent: 0.127204
Thread: 16, spent: 0.241628
Thread: 14, spent: 0.287368
Thread: 13, spent: 0.378658
Thread: 4, spent: 0.535787
Thread: 9, spent: 0.567728
Thread: 19, spent: 0.865574
Thread: 1, spent: 0.879066
Thread: 8, spent: 2.18616
Thread: 12, spent: 2.3365
Thread: 15, spent: 3.08579
Thread: 11, spent: 3.13387
Thread: 5, spent: 3.52276
Thread: 18, spent: 4.95437
Thread: 10, spent: 5.00127
Thread: 6, spent: 5.19421
Thread: 17, spent: 5.19038
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 5.21587
```

在上面的結果中觀察到 Core 3 跑得都最慢不確定原因為何，如果不去觀看 Core 3 最後完成時間的話 Worst-Fit 和 First-Fit 和 Best-Fit 看起來的執行結果平均時間都是差不多的。

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

FIFO 會將先排入的 Thread 先做完再去做下個 Thread。

RR 則是可以觀察到 PID 切換是三個再輪迴執行 (1981>>1980>>1984>>1980>>1981>>1984>>…….)。

FIFO 比 RR 還要快一點，因為 RR 需要一直切換 Thread，能發現說 FIFO 必須要等到比自己還要先排的做完才能執行所以如果有一個很長的 Thread 先佔住的話後面比較短的 Thread 就會被 delay 很久，而 RR 就是會循環就執行所以不用擔心較短的 Thread 反應時間會很長，但相對的要多花費 switch 的時間。