

# OS HW3 Multithread Programming

石育璋 108062633

## Implementation

### Parameters Structure

```
typedef struct _merge_sort_arg_t {  
    int i;  
    int j;  
    int *a;  
} merge_sort_arg_t;
```

因為thread執行目標function需要傳遞參數，所以創建一個structure，裡面包含sorting起始位置 *i* 結束位置 *j* 以及需要sorting的array *a* .

### Thread Entry Function

```
void merge_sort_entry(void *data) {  
    // convert parameter type  
    merge_sort_arg_t *arg = (merge_sort_arg_t*) data;  
  
    merge_sort(arg->a, arg->i, arg->j);  
}
```

先轉換輸入structure type，再去執行真正想要執行的function *merge\_sort()* .

### Thread Creation

```
merge_sort_arg_t left_arg, right_arg;  
left_arg.i = 0;  
left_arg.j = mid;  
left_arg.a = input_data;  
err_l = pthread_create(&left_t, NULL, merge_sort_entry, &left_arg);
```

(上以左thread為例)分別創建merge sort左右兩個threads所需要傳遞parameters的structure，並初始化structure裡的variables，再使用pthread.h提供的 `pthread_create()` 讓thread去執行指定的function。

## Merge Result

```
pthread_join(left_t, NULL);
pthread_join(right_t, NULL);

merge(input_data, 0, mid, input_data_size-1);
```

使用 `pthread_join()` 讓main thread等待左右兩個threads完成sorting，完成兩部份sorting後，用main thread執行 `merge()` 合併左右兩個部分。

## Merge Sort

```
void merge_sort(int *arr, int l, int r) {
    // check boundary
    if(l >= r) return;

    int m = l + (r-l)/2;

    merge_sort(arr, l, m);
    merge_sort(arr, m+1, r);

    merge(arr, l, m, r);
}

void merge(int *arr, int l, int m, int r) {
    // printf("merge\n");
    int i, j, k;
    int n1 = m-l+1;
    int n2 = r-m;

    int L[n1], R[n2];

    for(i=0; i<n1; i++) {
        L[i] = *(arr+l+i);
```

```

    }
    for(j=0; j<n2; j++) {
        R[j] = *(arr+m+1+j);
    }

    i = 0;
    j = 0;
    k = 1;

    while(i<n1 && j<n2) {
        if(L[i] <= R[j]) {
            *(arr+k) = L[i];
            i++;
        } else {
            *(arr+k) = R[j];
            j++;
        }
        k++;
    }

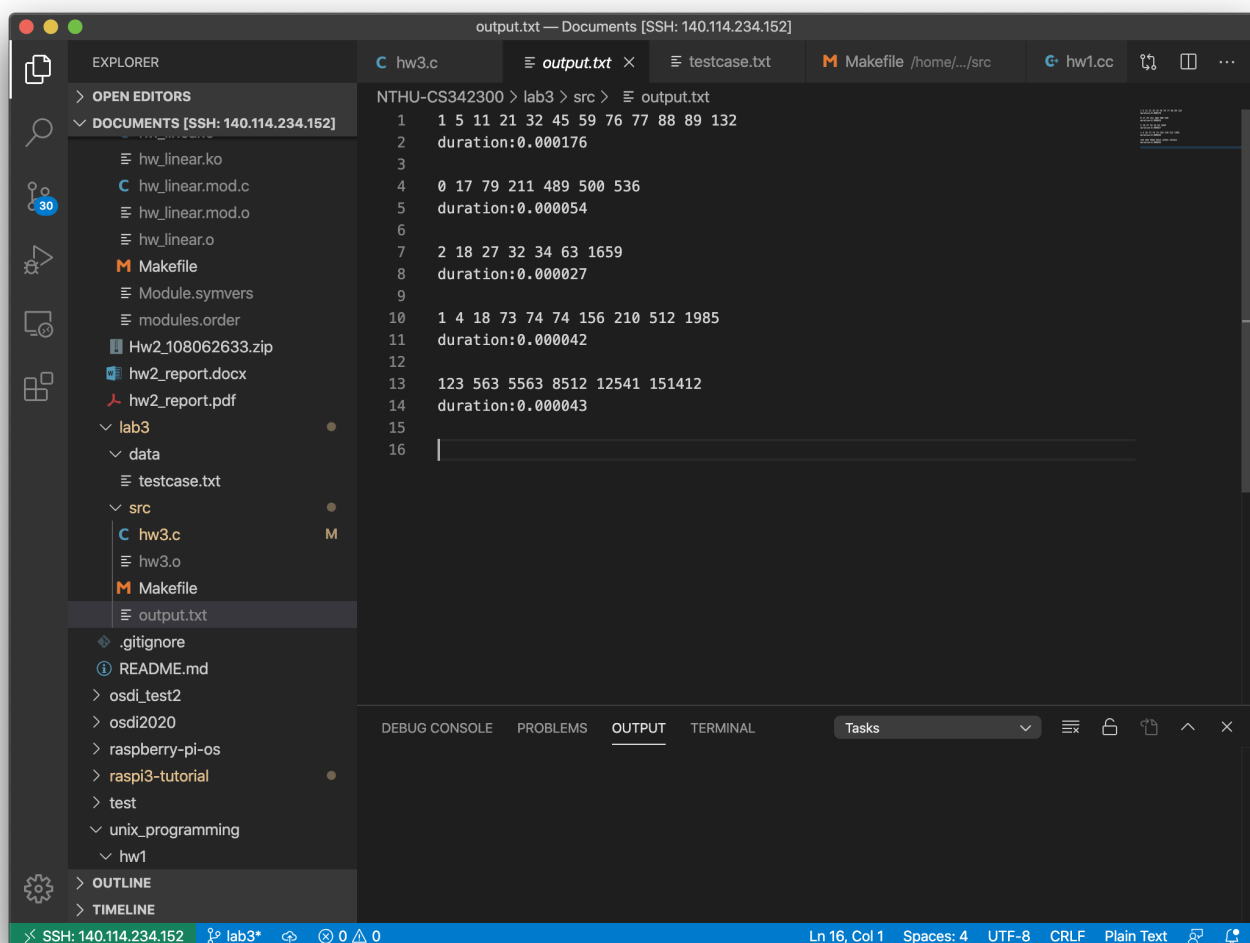
    // copy the remaining part
    while(i<n1) {
        *(arr+k) = L[i];
        i++;
        k++;
    }
    while(j<n2) {
        *(arr+k) = R[j];
        j++;
        k++;
    }
}

```

上面的code就是一般merge sort的implementation

## Result

用助教提供的 `testcase.txt` 執行結果如下：



```
output.txt — Documents [SSH: 140.114.234.152]
NTHU-CS342300 > lab3 > src > output.txt
1 1 5 11 21 32 45 59 76 77 88 89 132
2 duration:0.000176
3
4 0 17 79 211 489 500 536
5 duration:0.000054
6
7 2 18 27 32 34 63 1659
8 duration:0.000027
9
10 1 4 18 73 74 74 156 210 512 1985
11 duration:0.000042
12
13 123 563 5563 8512 12541 151412
14 duration:0.000043
15
16
```

## Reference

1. <https://www.techiedelight.com/find-execution-time-c-program/>
2. <https://www.edureka.co/blog/merge-sort-in-c/>
3. <http://blog.gitdns.org/2016/12/06/pthread/>

## Appendix - source code

### Makefile

```
CC := gcc
OPS := -pthread
TARGET := hw3.c

all: hw3.o
```

```
.PHONY: clean all run

hw3.o: $(TARGET)
    $(CC) $< $(OPS) -o [email protected]

run:
    ./hw3.o ../data/testcase.txt output.txt

clean:
    rm *.o output.txt
```

## hw3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

typedef struct _merge_sort_arg_t {
    int i;
    int j;
    int *a;
} merge_sort_arg_t;

void merge_sort_entry(void *data);
void merge_sort(int *arr, int l, int r);
void merge(int *arr, int l, int m, int r);

int main(int argc, char* argv[]) {
    FILE *in_file = fopen(argv[1], "r");
    FILE *out_file = fopen(argv[2], "w");

    // check file
    if (in_file == NULL) {
        printf("Error! Could not open input file.\n");
        exit(-1);
    }
}
```

```

}

pthread_t left_t, right_t;
int input_data[10000];
int input_data_size;
int fscanf_return = 0;

for(int m=1; !feof (in_file); m++) {
    printf("line %d\n", m);
    // read data
    input_data_size = 0;
    char c = '0';
    int err_l, err_r;
    clock_t begin = clock();

    for(int i=0; (!feof (in_file) && c != '\n' && c != '\r'); i++) {
        fscanf_return = fscanf(in_file, "%d%c", &input_data[i], &c);
        if(fscanf_return < 2) break;
        input_data_size++;
    }

    if(input_data_size == 0) continue;

    int mid = input_data_size/2;
    merge_sort_arg_t left_arg, right_arg;
    left_arg.i = 0;
    left_arg.j = mid;
    left_arg.a = input_data;
    err_l = pthread_create(&left_t, NULL, merge_sort_entry, &left_arg);

    right_arg.i = mid+1;
    right_arg.j = input_data_size-1;
    right_arg.a = input_data;
    err_r = pthread_create(&right_t, NULL, merge_sort_entry, &right_arg);

    if(err_l != 0 && err_r != 0) {
        printf("ERROR return code from pthread_create()\n");
    }
}

```

```

    }

    pthread_join(left_t, NULL);
    pthread_join(right_t, NULL);

    merge(input_data, 0, mid, input_data_size-1);

    // print data
    for(int i=0; i<input_data_size; i++) {
        fprintf(out_file, "%d ", input_data[i]);
    }
    fprintf(out_file, "\r");

    clock_t end = clock();
    double duration = end - begin;
    fprintf(out_file, "duration:%f\r\r", duration / CLOCKS_PER_SEC);
}

fclose (in_file);
fclose(out_file);
}

void merge_sort_entry(void *data) {
    // convert parameter type
    merge_sort_arg_t *arg = (merge_sort_arg_t*) data;

    merge_sort(arg->a, arg->i, arg->j);
}

void merge_sort(int *arr, int l, int r) {
    // check boundary
    if(l >= r) return;

    int m = l + (r-l)/2;

    merge_sort(arr, l, m);
    merge_sort(arr, m+1, r);
}

```

```

    merge(arr, l, m, r);
}

void merge(int *arr, int l, int m, int r) {
    // printf("merge\n");
    int i, j, k;
    int n1 = m-l+1;
    int n2 = r-m;

    int L[n1], R[n2];

    for(i=0; i<n1; i++) {
        L[i] = *(arr+l+i);
    }
    for(j=0; j<n2; j++) {
        R[j] = *(arr+m+1+j);
    }

    i = 0;
    j = 0;
    k = l;

    while(i<n1 && j<n2) {
        if(L[i] <= R[j]) {
            *(arr+k) = L[i];
            i++;
        } else {
            *(arr+k) = R[j];
            j++;
        }
        k++;
    }

    // copy the remaining part
    while(i<n1) {
        *(arr+k) = L[i];
        i++;
    }

```



```
        k++;
    }
    while(j<n2) {
        *(arr+k) = R[j];
        j++;
        k++;
    }
}
```

## testcase.txt

```
5 132 89 45 76 21 1 59 88 11 32 77
536 211 489 500 17 0 79
32 18 2 63 34 27 1659
74 73 1985 512 74 210 156 4 18 1
12541 151412 123 8512 5563 563
```

## output.txt

```
1 5 11 21 32 45 59 76 77 88 89 132
duration:0.000176

0 17 79 211 489 500 536
duration:0.000054

2 18 27 32 34 63 1659
duration:0.000027

1 4 18 73 74 74 156 210 512 1985
duration:0.000042

123 563 5563 8512 12541 151412
duration:0.000043
```