Programming Assignment 2 – Thread

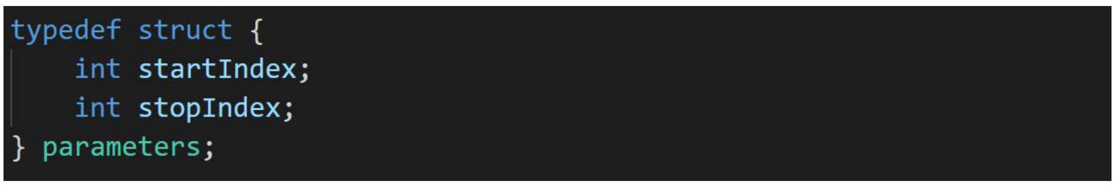
薛劭杰 1930026143

Requirement:

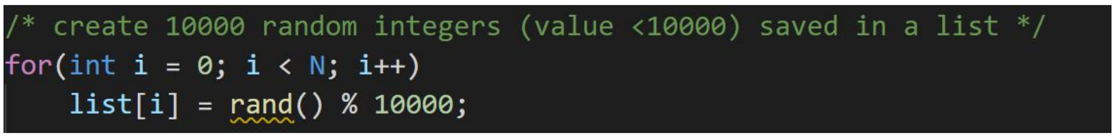
We should build a thread design to solve a sorting problem. Initially, we initialize a list which contains 10000 numbers. Then we define a structure, used to record the begin index and the ended index. In this way we can catch the part of the list. Then we can divide the process to the two threads, each thread can deal with half of list sorting, which means that it will accelerate the solution after processing by thread.

How to solve question:

Step1: Initialize a list which contains 10000 numbers, and the structure $parameters$.



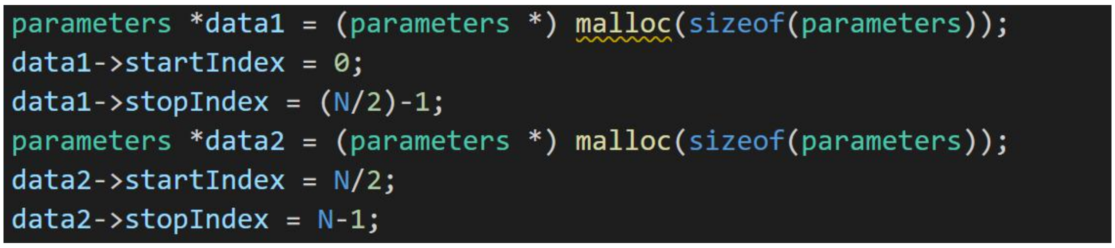
```c
typedef struct {
    int startIndex;
    int stopIndex;
} parameters;
```



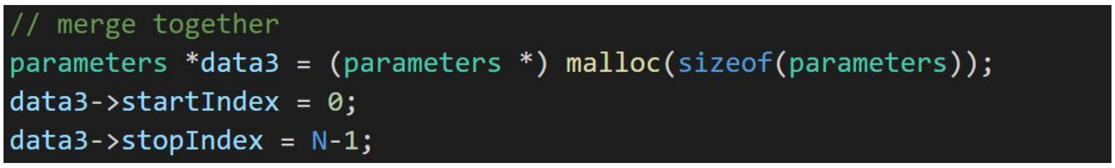
```c
/* create 10000 random integers (value <10000) saved in a list */
for(int i = 0; i < N; i++)
    list[i] = rand() % 10000;
```

Where N is macro definition with the $\#define\ N$.

Step2: Assign three structure entities, which are the index of the first half of the list, the index at the beginning and end of the list and the index at the beginning and end of the entire list respectively.



```c
parameters *data1 = (parameters *) malloc(sizeof(parameters));
data1->startIndex = 0;
data1->stopIndex = (N/2)-1;
parameters *data2 = (parameters *) malloc(sizeof(parameters));
data2->startIndex = N/2;
data2->stopIndex = N-1;
```



```c
// merge together
parameters *data3 = (parameters *) malloc(sizeof(parameters));
data3->startIndex = 0;
data3->stopIndex = N-1;
```

Step3: Create two threads and define the threads worker method which is called when the thread is created. We can create the threads by function $pthread\_create()$.

$int\ thread\_create(pthread\_t\ *tidp, pthread\_attr\_t\ *attr, (void\ *)\ worker\ method\ void\ *arg);$ The first argument is a pointer to the thread identifier. The second parameter sets the thread properties. The third argument is the starting address from which the thread runs the function and the last argument is the argument to which the function is run. If the thread was created successfully, 0 is returned. If the thread fails to be created, the error number is returned and the contents of the *thread are undefined. When success is returned, the memory unit pointed to by worker method is set to the thread ID of the newly created thread.

```
pthread_create(&pid1, NULL, worker1, data1);
pthread_create(&pid2, NULL, worker2, data2);
```

```
void *worker1(void* arg);
void *worker2(void* arg);
```

Step4: Write the worker function. Here we should pay attention to the passing of parameter. The structure entity is the parameter should be passed to the worker method, and it should not add the address operation & before it because the structure itself points to the address.
And for the sorting algorithm, we can use the bubble sort algorithm:

```
for (int i = data->startIndex; i < data->stopIndex+1; i++) {
    for(int j = data->startIndex; j < data->stopIndex; j++) {
        if(list[i] < list[j]){
            int temp = list[i];
            list[i] = list[j];
            list[j] = temp;
        }
    }
}
```

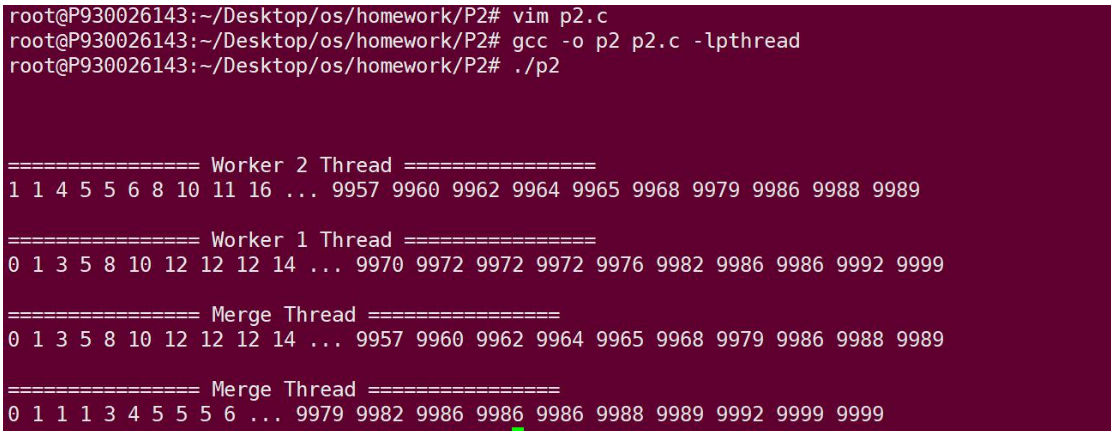
Step5: Merge two thread together and write the sorting algorithm. We can use the function $pthread\_join()$ to merge them to main one, it will wait for the thread to finish.
$int\, pthread\_join(pthread\_t\; thread, void\; **\; retval);$ The thread argument is used to specify which thread to receive the return value from; The $retval$ argument represents the received value. If the thread thread does not return a value, or if we do not need to receive a value from the thread thread, we can set the $retval$ argument to NULL.

```
pthread_join(pid1, NULL);
pthread_join(pid2, NULL);
```

For the sorting algorithm, we can adopt the insertion sort method, which can improve sorting efficiency because a part of list has been sorted in the two threads.

```
for(i = 1; i < (data3->stopIndex - data3->startIndex); i++){
    temp = list[i];  // Recond the right(insertion) values
    j = i-1;
    while(j >= 0 && list[j] > temp){
        list[j+1] = list[j];  // If left one is bigger than insertion one
        j--;  // Move forward
    }
    list[j+1] = temp;  // Insertion values
}
```

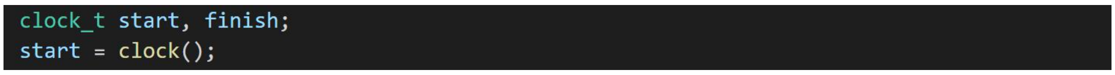
The additional head file we need are $< stdlib.h >$ for dynamically allocated memory (malloc) and $< pthread.h >$ for create threads and join thread function.
Step6: Compile and run:



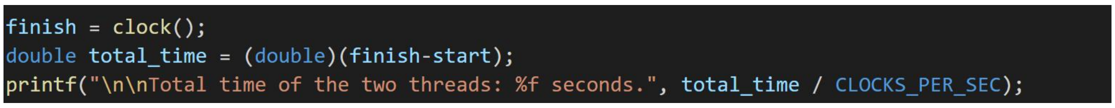
```
root@P930026143:~/Desktop/os/homework/P2# vim p2.c
root@P930026143:~/Desktop/os/homework/P2# gcc -o p2 p2.c -lpthread
root@P930026143:~/Desktop/os/homework/P2# ./p2



================ Worker 2 Thread ================
1 1 4 5 5 6 8 10 11 16 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Worker 1 Thread ================
0 1 3 5 8 10 12 12 12 14 ... 9970 9972 9972 9972 9976 9982 9986 9986 9992 9999

================ Merge Thread ================
0 1 3 5 8 10 12 12 12 14 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Merge Thread ================
0 1 1 1 3 4 5 5 5 6 ... 9979 9982 9986 9986 9986 9988 9989 9992 9999 9999
```

Time Test:

We can use the function in the $< time.h >$ head file.

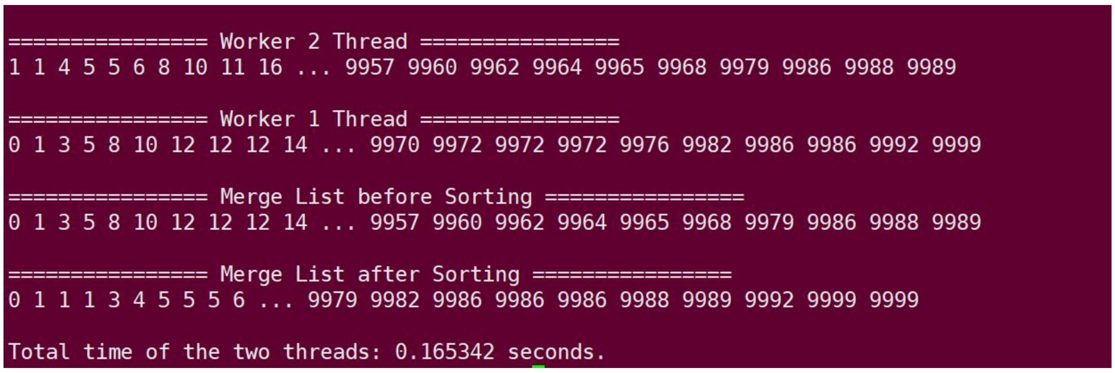


```c
clock_t start, finish;
start = clock();
```



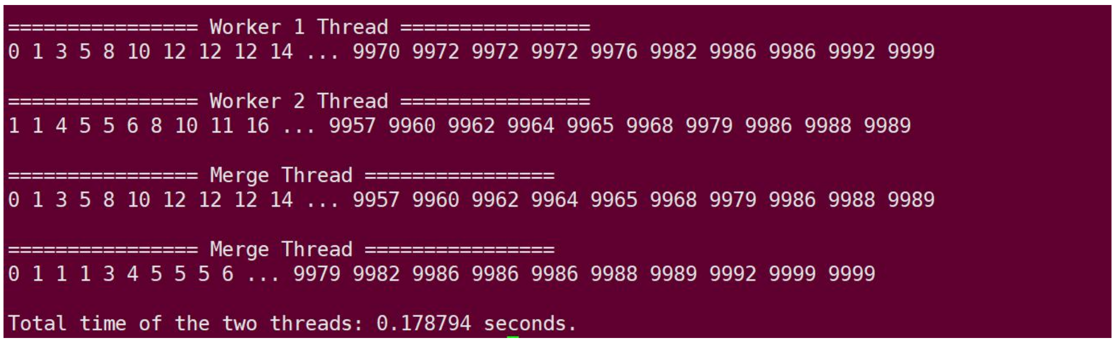
```c
finish = clock();
double total_time = (double)(finish-start);
printf("\n\nTotal time of the two threads: %f seconds.", total_time / CLOCKS_PER_SEC);
```

1. With the two threads case, it takes about 0.165342 seconds.



```
================ Worker 2 Thread ================
1 1 4 5 5 6 8 10 11 16 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Worker 1 Thread ================
0 1 3 5 8 10 12 12 12 14 ... 9970 9972 9972 9972 9976 9982 9986 9986 9992 9999

================ Merge List before Sorting ================
0 1 3 5 8 10 12 12 12 14 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Merge List after Sorting ================
0 1 1 1 3 4 5 5 5 6 ... 9979 9982 9986 9986 9986 9988 9989 9992 9999 9999

Total time of the two threads: 0.165342 seconds.
```

2. Without the two threads case, it takes about 0.178794 seconds.



```
================ Worker 1 Thread ================
0 1 3 5 8 10 12 12 12 14 ... 9970 9972 9972 9972 9976 9982 9986 9986 9992 9999

================ Worker 2 Thread ================
1 1 4 5 5 6 8 10 11 16 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Merge Thread ================
0 1 3 5 8 10 12 12 12 14 ... 9957 9960 9962 9964 9965 9968 9979 9986 9988 9989

================ Merge Thread ================
0 1 1 1 3 4 5 5 5 6 ... 9979 9982 9986 9986 9986 9988 9989 9992 9999 9999

Total time of the two threads: 0.178794 seconds.
```

And absolutely the first one case should be **faster** than the later.