Operating System hw3

105061212 王家駿

1. Data structure

```c
// array to store the information of subarray
typedef struct sArray{
    int low, mid, high;
} Array;
```

At first, we define an Array structure, which contains the information of subarray in which it begins and ends, and it would be used at the merge sort later.

2. Main function

```c
int main(int argc, char* argv[])
{
    FILE* input, * output;
    char buff[200000];
    int buff_size = 0;
    char* line;
    int i;
    double time0, time1;
    pthread_t thread1, thread2, thread3;
    Array arr0, arr1, arr2;


    if (argc != 3) {  // format not correct
        printf("ERROR!\n");
        return -1;
    }
    else {  // open input and output files
        input = fopen(argv[1], "r");
        output = fopen(argv[2], "w");
    }

    while ((buff[buff_size] = fgetc(input)) != EOF) {  // read input
        if (buff[buff_size] == '\n') {  // end of line
            line = buff;
            arr_size = 0;
            buff_size = 0;
            while (sscanf(line, " %d%n", &arr[arr_size], &buff_size) == 1) {  // convert strings to integers
                arr_size++;
                line += buff_size;
            }
```

```
            // set subarray range
            arr0.low = 0;
            arr0.high = arr_size - 1;
            arr0.mid = (arr0.low + arr0.high) / 2;
            arr1.low = 0;
            arr1.high = arr0.mid;
            arr1.mid = (arr1.low + arr1.high) / 2;
            arr2.low = arr0.mid + 1;
            arr2.high = arr_size - 1;
            arr2.mid = (arr2.low + arr2.high) / 2;

            time0 = GetTime();  // get current time

            // two threads for merge sort, one thread for merge them together
            pthread_create(&thread1, NULL, merge_sort1, &arr1);
            pthread_create(&thread2, NULL, merge_sort2, &arr2);

            pthread_join(thread1, NULL);
            pthread_join(thread2, NULL);

            pthread_create(&thread3, NULL, merge_thread, &arr0);
            pthread_join(thread3, NULL);

            time1 = GetTime();  // get current time
            printf("time: %e seconds\n", time1 - time0);

            for (i = 0; i < arr_size; i++) {  // output results
                fprintf(output, "%d ", arr[i]);
            }
            fprintf(output, "\n");

            buff_size = 0;
        }
        else buff_size++;
    }
```

In the main function, we first read the input data by using sscanf function, and then we create three threads, two for merging two parts of the array and the rest one for linking the two subarrays together.

We initially create the first two threads, and then the third thread later. With the use of pthread_join function, we could ensure that the third thread would execute only after the first two thread are done.

3. The first two threads

```
void* merge_sort1(void* arr_data)  // first thread for the former half
{
    Array* a = arr_data;

    // merge sort
    if (a->low < a->high) {
        mergesort(a->low, a->mid);
        mergesort(a->mid + 1, a->high);
        merge(a->low, a->high, a->mid);
    }

    pthread_exit(NULL);
}
```

In the first thread, we call the merge sort algorithm to sort the subarray respectively by the subarray information we defined before. The second thread works the same way as the first thread.

4. The last thread

```
void* merge_thread(void* arr_data)  // merge two threads
{
    Array* a = arr_data;

    merge(a->low, a->high, a->mid);  // merge them

    pthread_exit(NULL);
}
```

In the last thread, the two subarrays would be merged together by using the merge function. And this thread would run only after the previous two threads are done.

5. Merge sort algorithm

```
void mergesort(int low, int high)  // merge sort algorithm
{
    int mid;

    if (low < high) {
        mid = (low + high) / 2;
        mergesort(low, mid);
        mergesort(mid + 1, high);
        merge(low, high, mid);
    }
}
```

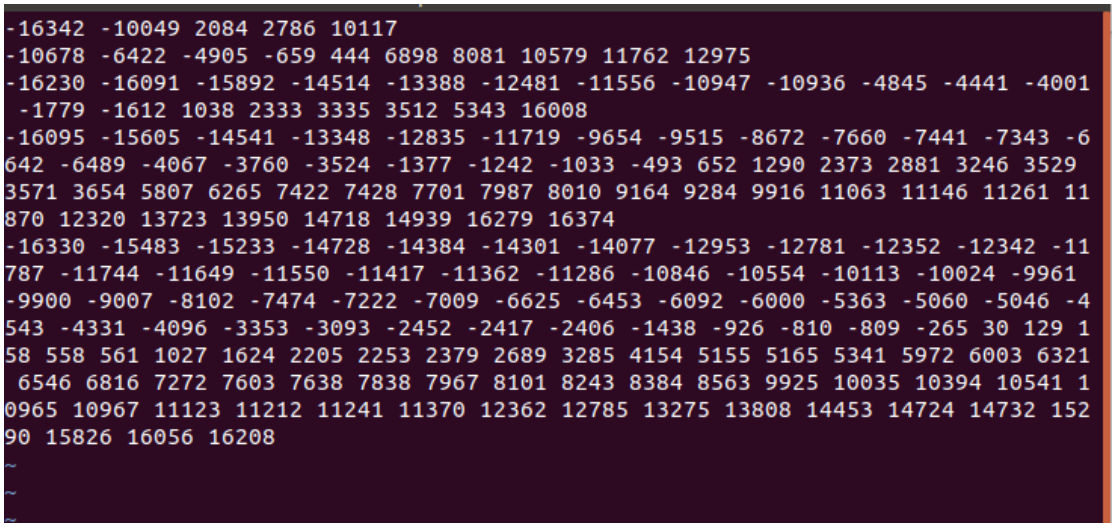
```
void merge(int low, int high, int mid)  // merge to subarrays
{
    int h = low;
    int i = low;
    int j = mid + 1;
    int k;
    int* b = (int*)malloc(sizeof(int) * 10050);

    while (h <= mid && j <= high) {
        if (arr[h] < arr[j]) {
            b[i] = arr[h];
            h++;
        }
        else {
            b[i] = arr[j];
            j++;
        }
        i++;
    }

    if (h > mid) {
        for (k = j; k <= high; k++) {
            b[i] = arr[k];
            i++;
        }
    }
    else {
        for (k = h; k <= mid; k++) {
            b[i] = arr[k];
            i++;
        }
    }
    for (k = low; k <= high; k++) arr[k] = b[k];

    free(b);
}
```

In the merge sort algorithm, we use the divide-and-conquer method to separate the array into small pieces, and then merge them with the merge function. After the recursion calls go to the end, the whole array would be well sorted.

6. Results

```
time: 1.204014e-03 seconds
time: 1.781607e-02 seconds
time: 1.514101e-02 seconds
time: 3.027916e-04 seconds
time: 7.746935e-03 seconds
```

Run at ubuntu


-16342 -10049 2084 2786 10117
-10678 -6422 -4905 -659 444 6898 8081 10579 11762 12975
-16230 -16091 -15892 -14514 -13388 -12481 -11556 -10947 -10936 -4845 -4441 -4001
-1779 -1612 1038 2333 3335 3512 5343 16008
-16095 -15605 -14541 -13348 -12835 -11719 -9654 -9515 -8672 -7660 -7441 -7343 -6
642 -6489 -4067 -3760 -3524 -1377 -1242 -1033 -493 652 1290 2373 2881 3246 3529
3571 3654 5807 6265 7422 7428 7701 7987 8010 9164 9284 9916 11063 11146 11261 11
870 12320 13723 13950 14718 14939 16279 16374
-16330 -15483 -15233 -14728 -14384 -14301 -14077 -12953 -12781 -12352 -12342 -11
787 -11744 -11649 -11550 -11417 -11362 -11286 -10846 -10554 -10113 -10024 -9961
-9900 -9007 -8102 -7474 -7222 -7009 -6625 -6453 -6092 -6000 -5363 -5060 -5046 -4
543 -4331 -4096 -3353 -3093 -2452 -2417 -2406 -1438 -926 -810 -809 -265 30 129 1
58 558 561 1027 1624 2205 2253 2379 2689 3285 4154 5155 5165 5341 5972 6003 6321
6546 6816 7272 7603 7638 7838 7967 8101 8243 8384 8563 9925 10035 10394 10541 1
0965 10967 11123 11212 11241 11370 12362 12785 13275 13808 14453 14724 14732 152
90 15826 16056 16208

output.txt