CS356 Project 3

Name : HaotianXue

Student ID : 518021910506

Email : xavihart@sjtu.edu.cn

CS356 Project 3
- Introduction
- Multithreaded Sorting Application
  - Analysis
  - Code Structure
  - Testing and Results
- Fork-Join Sorting Application
  - Merge Sort
    - Code Structure
    - Testing and Results
  - Quick Sort
    - Code Structure
    - Testing and Results
- Conclusion

Introduction

This is the report for project3 for CS356, also for COS book chapter4. We have two sections to do, firstly we are required to use multi-thread to sort an array, then we simulate the fork-join algorithm to do array sorting by Merge Sort Algorithm and Quick Sort Algorithm respectively.

Multithreaded Sorting Application

Analysis

In this part we are going to sort an array using multi-thread program, to be specific, first we break an array into two sperate parts, then we use two threads to sort the two arrays respectively using Bubble Sort Algorithm, then we use another thread to merge the two sorted arrays.

Code Structure

We use `pthread` in linux to do multithread work. Since the parameters for the sorting and merging functions are more than one, we use a `struct` to package them, then we use `void*` in `C` to pass the format parameters.

Then code can be divided into three parts: get data from `data.txt`, which includes the array information; create two thread to do sorting work; merge the two array to get the sorted array:

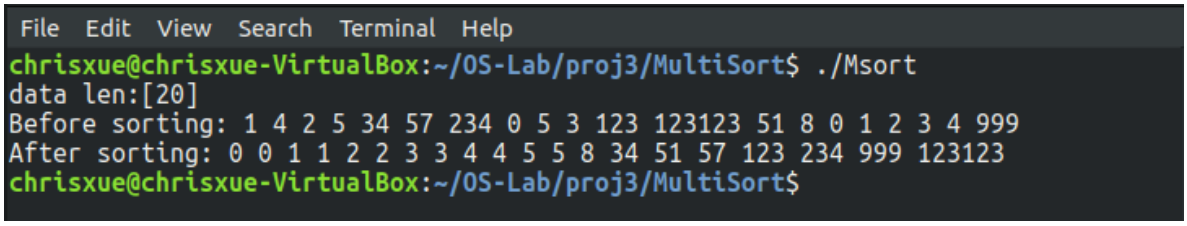
```
#include<stdio.h>
#include<pthread.h>
```

```c
#include<stdlib.h>

int *array;
int *result;
int l, mid;


typedef struct{
   int* arg1;
   int arg2;
   int arg3;
}args_sort;

typedef struct{
   int * arg1;
   int * arg2;
   int arg3;
   int arg4;
   int arg5;
   int arg6;
}args_merge;


void sort(void*);
void merge(void*);

int main(void){
   FILE *fin;
   fin = fopen("data.txt", "rb");
   fscanf(fin,"%d", &l);
   printf("data len:[%d]\n", l);
   array = (int*) malloc(l * sizeof(int));
   result = (int*) malloc(l * sizeof(int));
   for(int i = 0;i < l;++i){
       fscanf(fin, "%d", &array[i]);
   }
   printf("Before sorting: ");
   for(int i = 0;i < l;++i){
       printf("%d ", array[i]);
   }printf("\n");

   mid = l >> 2;


   pthread_t t1, t2, t3;
   pthread_attr_t attr;
   pthread_attr_init(&attr);

   args_sort argA={array, 0, mid};
   args_sort argB={array, mid+1, l-1};
   args_merge argC={array, result, 0, mid+1, mid, l-1};
  // argA = (struct args_sort*)malloc(sizeof(args_sort));
  // argB = (struct args_sort*)malloc(sizeof(args_sort));
  // argC = (struct args_merge*)malloc(sizeof(args_merge));


   pthread_create(&t1, &attr, sort,&(argA));
   pthread_create(&t2, &attr, sort, &argB);
```

```
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   pthread_create(&t3, &attr, merge, &argC);
   pthread_join(t3, NULL);




  // sort(array, 0, mid);
  // sort(array, mid + 1, l - 1);
  // merge(array, result, 0, mid + 1, mid, l - 1);

   printf("After sorting: ");
   for(int i = 0;i < l;++i){
       printf("%d ", result[i]);
   }printf("\n");


   free(array);
   free(result);


  return 0;
}


void sort(void* arg){
    int*a;
    int l, r;
    args_sort *args = (args_sort*) arg;
    a = args -> arg1;
    l = args -> arg2;
    r = args -> arg3;

    for(int i = r - 1;i >= l; --i){
        for(int j = l;j <= i;++j){
        if(a[j] > a[j + 1]){
            int tmp;
        tmp = a[j];
        a[j] = a[j + 1];
        a[j + 1]  = tmp;
        }
    }
    }

    return;
}

void merge(void*arg){
    args_merge* args = (args_merge*) arg;
    int*a, *b, l1, l2, r1, r2;
    a = args -> arg1;
    b = args -> arg2;
    l1 = args -> arg3;
    l2 = args -> arg4;
    r1 = args -> arg5;
    r2 = args -> arg6;
```

```
    int p1 = l1, p2 = l2, pb = 0;
    while(p1 <= r1 && p2 <= r2){
        if(a[p1] > a[p2]){
       b[pb ++] = a[p2];
       p2 ++;
    }else{
           b[pb ++] = a[p1];
       p1 ++;
    }
    }
   while(p1 <= r1) b[pb ++] = a[p1 ++];
   while(p2 <= r2) b[pb ++] = a[p2 ++];
   return ;
}
```

Testing and Results



Fork-Join Sorting Application

Merge Sort

Code Structure

Just like the traditional merge sort algorithm, we break and array into two half, recursively call the sort function to make the two arrays sorted, then we merge them to get the answer.

In this implementation, we create two threads to sort the left and right half, one thread to merge then in each sort function until the subarray to sort in shorter than two.

Also we use struct to save the parameters, and data is saved in `data.txt`.

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

int *array;
int l;

typedef struct{
   int* arg1; // array
   int arg2;  // start
   int arg3;  // end
```

```c
}args_sort;

typedef struct{
   int* arg1; // source array
   int arg2;  // l1
   int arg3;  // l2
   int arg4;  // r1
   int arg5;  // r2
}args_merge;

void sort(void *);
void merge(void*);


int main(){
   FILE *fin;
   fin = fopen("data.txt", "rb");
   fscanf(fin,"%d", &l);
   printf("data len:[%d]\n", l);
   array = (int*) malloc(l * sizeof(int));
   for(int i = 0;i < l;++i){
       fscanf(fin, "%d", &array[i]);
   }
   printf("Before sorting: ");
   for(int i = 0;i < l;++i){
       printf("%d ", array[i]);
   }printf("\n");


   pthread_t t;
   pthread_attr_t attr;
   pthread_attr_init(&attr);
   args_sort arg0={array, 0, l-1};
   pthread_create(&t, &attr, sort, &(arg0));
   pthread_join(t, NULL);

   printf("After sorting: ");
   for(int i = 0;i < l;++i){
       printf("%d ", array[i]);
   }printf("\n");

   free(array);
   return 0;

}

void sort(void* arg){
        args_sort* args = (args_sort*) arg;
        int *a, l, r;

        a = args -> arg1;
        l = args -> arg2;
        r = args -> arg3;
        int mid = (l + r) >> 1;


    if(r - l < 2){
        int r_ = a[r], l_ = a[l];
```

```
        a[l] = r_ > l_ ? l_ : r_;
        a[r] = r_ > l_ ? r_ : l_;
        pthread_exit(0);
        return;
    }

    args_sort argL = {a, l, mid};
    args_sort argR = {a, mid + 1, r};
    args_merge argM = {a, l, mid+1, mid, r};

    pthread_t t_1, t_2, t_3;
    pthread_attr_t attr;

    pthread_attr_init(&attr);
    pthread_create(&t_1, &attr, sort, &argL);
    pthread_create(&t_2, &attr, sort, &argR);
    pthread_join(t_1, NULL);
    pthread_join(t_2, NULL);

    pthread_create(&t_3, &attr, merge, &argM);
    pthread_join(t_3, NULL);

    pthread_exit(0);
    return;
}


void merge(void* arg){
    args_merge* args = (args_merge*) arg;
    int *a, l1, l2, r1, r2;

    a = args -> arg1;
    l1 = args -> arg2;
    l2 = args -> arg3;
    r1 = args -> arg4;
    r2 = args -> arg5;
    int* b;
    b = (int*) malloc((r2 - l1 + 1) * sizeof(int));
    int pb=0, pl=l1, pr=l2;
    while(pl <= r1 && pr <= r2){
        if(a[pl] > a[pr]){
            b[pb++] = a[pr++];
        }else{
            b[pb++] = a[pl++];
        }
    }
    while(pl <= r1) b[pb++] = a[pl ++];
    while(pr <= r2) b[pb++] = a[pr ++];
    for(int i = l1; i <= r2;++i){
        a[i] = b[i - l1];
    }
    free(b);
    pthread_exit(0);
    return;
}
```

Testing and Results

```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj3/MergeSort$ ./MergeSort
data len:[20]
Before sorting: 1 4 2 5 34 57 234 0 5 3 123 123123 51 8 0 1 2 3 4 999
After sorting: 0 0 1 1 2 2 3 3 4 4 5 5 8 34 51 57 123 234 999 123123
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj3/MergeSort$
```

Quick Sort

Code Structure

In Quick Sort Algorithm, we first pick the first element `x` as a pivot. Then we put all the elements greater than `x` in the right of `x`, all the elements less than `x` in the left of `x`. After that we recursively call the function to sort the left and right part of the array respectively.

Also we use two threads to sort the left and right part after reordering the array by pivot `x`:

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

int *array;
int l;

typedef struct{
   int* arg1; // array
   int arg2;  // start
   int arg3;  // end
}args_sort;


void sort(void *);

int main(){
   FILE *fin;
   fin = fopen("data.txt", "rb");
   fscanf(fin,"%d", &l);
   printf("data len:[%d]\n", l);
   array = (int*) malloc(l * sizeof(int));
   for(int i = 0;i < l;++i){
       fscanf(fin, "%d", &array[i]);
   }
   printf("Before sorting: ");
   for(int i = 0;i < l;++i){
       printf("%d ", array[i]);
   }printf("\n");


   pthread_t t;
   pthread_attr_t attr;
   pthread_attr_init(&attr);
   args_sort arg0={array, 0, l-1};
   pthread_create(&t, &attr, sort, &(arg0));
   pthread_join(t, NULL);
```

```
    printf("After sorting: ");
    for(int i = 0;i < l;++i){
        printf("%d ", array[i]);
    }printf("\n");

    free(array);
    return 0;

}


void sort(void* arg){
    args_sort* args = (args_sort*) arg;
    int *a, l, r;
    int mid = (l + r) >> 1;
    a = args -> arg1;
    l = args -> arg2;
    r = args -> arg3;

    if(r - l < 2){
        int r_ = a[r], l_ = a[l];
        a[l] = r_ > l_ ? l_ : r_;
        a[r] = r_ > l_ ? r_ : l_;
        pthread_exit(0);
        return;
    }

    int x=a[l],i=l,j=r;
    while(i<j)
    {
        while(i<j&&a[j]>=x)
            j--;
        a[i]=a[j];
        while(i<j&&a[i]<=x)
            i++;
        a[j]=a[i];
    }
    a[i]=x;
    args_sort arg1 = {a, l, i - 1};
    args_sort arg2 = {a, i+1, r};
    pthread_t t_1, t_2;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    pthread_create(&t_1, &attr, sort, &arg1);
    pthread_create(&t_2, &attr, sort, &arg2);
    pthread_join(t_1, NULL);
    pthread_join(t_2, NULL);

    pthread_exit(0);
    return;
}
```

Testing and Results

```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj3/QuickSort$ ./Qsort
data len:[30]
Before sorting: 1 4 2 5 34 57 234 0 5 3 123 123123 51 8 0 1 2 3 4 999 1 3 2 4 5 3 6 34 5 10
After sorting: 0 0 1 1 1 2 2 2 3 3 3 3 4 4 4 5 5 5 5 6 10 8 34 34 51 57 123 234 999 123123
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj3/QuickSort$
```

Conclusion

In this project, we learn about multi-thread programming and fork-join algorithm. We implement three kinds of sorting algorithm utilizing multi-thread implementation. This makes me have a better understanding of threads and fork-join algorithm, also it helps me learn more about c program in linux.

Thanks for the instructions and useful help offered by Prof. Wu and all the TAs!