OS Assignment 6

Written by Akilesh B, CS13B1042

September 13, 2015

Multithreaded sorting

Objective:

To sort the elements of an array by dividing into two equal subparts, sort each sublist and merge to get the sorted list.

How I accomplished?

I used a structure element which has indexes of the beginning and the end of the array.

```
struct element //struct containing the starting and ending indexes
{
int beg;
int end;
};
```

I use merge sort as the sorting algorithm. In merge_sort function, I declare two objects of the element structure. First object has beg = start of the array, end = mid element of the array. Second object has start = mid + 1 and end = last element of the array. These objects take care of sorting the first half and second half of the array respectively.

```
struct element var1, var2;
pthread_t thread1, thread2;
int mid = (beg+end)/2; //mid element
var1.beg = beg; //var1 takes care of first half of the array
var1.end = mid;
var2.beg = mid+1; //var2 takes care of second half of the array
var2.end = end;
```

Now I create two threads, first thread sorts the first half of the array and the second sorts the second half.

```
ret_val1 = pthread_create(&thread1, NULL, merge_sort, (void *)&var1);
ret_val2 = pthread_create(&thread2, NULL, merge_sort, (void *)&var2);
```

pthread_create returns 0 on success. This is used to check if the thread is successfully created or not.

Wait for both these threads to finish. This is achieved by using pthread_join(thread1, NULL) and pthread_join(thread2, NULL).

Finally merge the two sorted halves. This is done by the merge thread. Two separate sorting threads, sort each sublist recursively using merge sort and merge thread merges the two sublists into a single sorted list.

I get the number of elements as input from the user. The input array and output array is dynamically allocated memory using this value of n. The elements of the input array are randomly generated using the seed srand((unsigned)time(&t)+10).

In order to measure the time taken for sorting, I measure the time just before calling the merge_sort function in the main and just after sorting is done.

```
gettimeofday(&timevar, NULL);
long long t1 = timevar.tv_usec; //time before sorting
pthread_create(&init_thread, NULL, merge_sort, &init);
pthread_join(init_thread, NULL);
gettimeofday(&timevar, NULL);
long long t2 = timevar.tv_usec; //time after sorting.
```

The difference t2-t1 is the time taken for multi threaded sorting.

Multiprocess sorting

Objective:

The objective is to perform task done in part 1 using multi process instead of threads.

How I accomplished?

I follow a approach similar to what I did in part 1. Using a structure element, which has indexes of the beginning and the end of the array.

```
struct element //struct containing the starting and ending indexes
{
int beg;
int end;
};
```

I use merge sort as the sorting algorithm. In merge_sort function, I declare two objects of the element structure. First object has beg = start of the array, end = mid element of the array. Second object has start = mid + 1 and end = last element of the array. These objects take care of sorting the first half and second half of the array respectively.

struct element var1, var2;

int mid = (beg+end)/2; //mid element

var1.beg = beg; //var1 takes care of first half of the array

var1.end = mid;

var2.beg = mid+1; //var2 takes care of second half of the array

var2.end = end;

The parent process now creates two child processes. First child process performs sorting on the first half of the array and the second child process performs sorting on the second half of the array.

The parent waits for both its children to complete their respective tasks and finally merge both the sublists together to form a single sorted list.

I share the memory using mmap so that all the children process have access to input array.

input = mmap(NULL, sizeof *input, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

pid_t child_a, child_b; //create two children

child_a = fork(); // child 1 created by parent

if(child_a == 0)

{

merge_sort((void *) &var1); //child 1 performs sorting of first half of input array

_exit(0);

} else

{

child_b = fork(); //child 2 created by parent.

if(child_b == 0)

{ merge_sort((void *) &var2); //child 2 performs sorting of second half of child array

_exit(0); }

else

{ wait(NULL); // Parent waiting for both the children to execute first.

wait(NULL); //parent waits for both its children to finish their tasks and performs merge functionality

int i = beg, j = mid+1, k = beg;

while(i <= mid && j <= end) //as long as i is less than mid and j is less than end

```
{ if(input[i] < input[j])
output[k++] = input[i++];
else output[k++] = input[j++]; }
for(; i <= mid; i++)
output[k++] = input[i];
for(; j <= end; j++)
output[k++] = input[j];
for(i = beg; i <= end; i++)
input[i] = output[i];
}
}
```

Finally merge the two sorted halves. This is done by the parent process which waits for both its children to finish.

I get the number of elements as input from the user. The input array and output array is dynamically allocated memory using this value of n. The elements of the input array are randomly generated using the seed srand((unsigned)time(&t)+10).

In order to measure the time taken for sorting, I measure the time just before calling the merge_sort function in the main and just after sorting is done.

```
gettimeofday(&timevar, NULL);
long long t1 = timevar.tv_usec; //time before sorting
merge_sort((void *) &init);
gettimeofday(&timevar, NULL);
long long t2 = timevar.tv_usec; //time after sorting.
```

The difference t2-t1 is the time taken for multi process sorting.

Comparison

I measured the performance of multi threaded sort and multi process sort by varying the size of the input unsorted array from 100 to 1000 in intervals of 50.

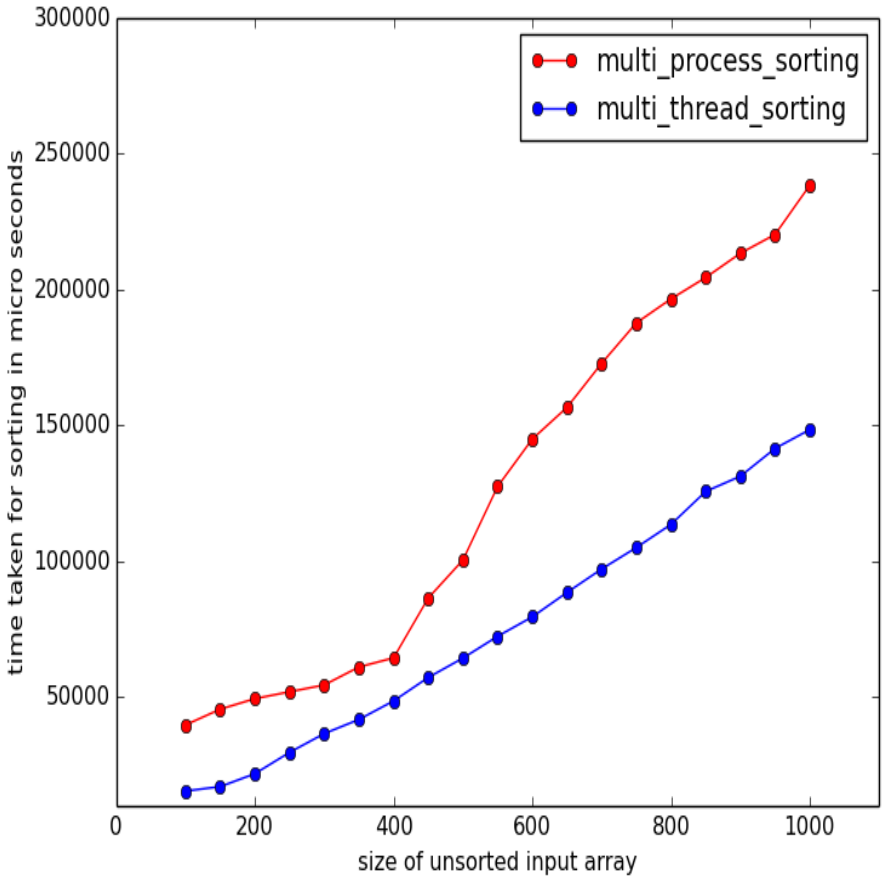
Vales obtained for multi process sorting (average obtained after 3 runs)

Size of input unsorted array	Time for sorting in micro seconds
100	39643
150	45437
200	49323
250	51855
300	54363
350	60967
400	64340
450	86353
500	100345
550	127423
600	144892
650	156542
700	172773
750	187645
800	196487
850	204356
900	213286
950	220192
1000	238306

Values obtained for multi thread sorting (average obtained after 3 runs)

Size of input unsorted array	Time for sorting in micro seconds
100	15340
150	16936
200	21703
250	29542
300	36458
350	41656
400	48482
450	57123
500	64259
550	72288
600	79432
650	88417
700	97056
750	104958
800	113476
850	125606
900	131211
950	141445
1000	148353

Graph



Conclusion:

Thus, multi process sorting and multi threaded sorting were done. Time taken for sorting was compared as input array size changed from 100 to 1000. It is seen from the graph and values obtained that multi thread sorting is faster than multi process sorting on most occasions.