Concurrent Merge sort

What I done

1. sorted the given array by using normal merge sort
2. sorted the given array by using the different processes merge sort
3. sorted the given array by using the threaded merge sort
4. Performance analysis on based on the above results

Merge sort using different processors

> Recursively make two child processes, one for the left half, one for the right half. If the number of elements in the array for a process is less than 5, perform a selection sort.The parent of the two children then merges the result and returns back to the parent.

```
void proc_mergesort(int *arr,int start,int end)
{
    int len=end-start+1;
    if(len<=5)
    {
        selection_sort(start,end);
        return NULL;
    }
    else
    {
        pid_t left,right;
        left=fork();
        if(left==0)
        {
            proc_mergesort(arr,start,start+len/2-1);
            _exit(0);
        }
        else
        {
            right=fork();
            if(right==0)
            {
                proc_mergesort(arr,start+len/2,end);
                _exit(0);
            }
        }
        int status;
        waitpid(left,&status,0);
        waitpid(right,&status,0);
        merge(arr,start,start+len/2-1,end);
    }

}
```

> This part of the code checks the array length if the length is less than 5 implements selection sort.If its greater than 5 then, splits array into two parts.It forks and creates a process to sort left part.In the parent, it forks again to create a process to sort right part.In the parent,it waits for the child processes sorting left and right part to join.

```
void selection_sort(int l,int r)
{
    for(int i=l;i<=r;i++)
        {
            int min=i;
            for(int j=i+1;j<=r;j++)
            {
                if(arr[j]<arr[min])
                    min=j;
            }
            int temp=arr[min];
            arr[min]=arr[i];
            arr[i]=temp;
        }

}
```

> This part of the code merges the two halves in all the methods.

```
void merge(int *arr,int start,int mid,int end)
{
    int left[10004],right[10004],ll=mid-start+1,rl=end-mid;
    for(int i=0;i<ll;i++)
        left[i]=arr[start+i];
    for(int i=0;i<rl;i++)
        right[i]=arr[mid+1+i];
    int a=0,b=0,c=0;
    while(a<ll && b<rl)
    {
        if(left[a]<right[b])
        {
            arr[c+start]=left[a++];
            c++;
        }
        else
        {
            arr[c+start]=right[b++];
            c++;
        }
    }
    while(a<ll)
    {
        arr[c+start]=left[a++];
        c++;
    }
    while(b<rl)
    {
        arr[c+start]=right[b++];
        c++;
    }
}
```

Bonus part

> using the threads

- checking the length if the length is less than 5, goes to selction sort else threads are created to sort left and right half of an array.structs are passed to it by passing all the parameters of the left and right arrays.these threads performs the sorting and we waits for them to join.

```
void * thread_mergesort(void * temp)
{
    struct timespec ts;
    struct node * elem = (node *) temp;
    int l=elem->l;
    int r=elem->r;
    if(l>r)
    {
        return NULL;
    }
    if((r-l)<=5)
    {
        selection_sort(l,r);
        return NULL;
    }
    struct node a1,a2;
    int m=(l+r)/2;
    pthread_t tid1,tid2;
    a1.l=l;
    a1.r=m;
    pthread_create(&tid1,NULL,thread_mergesort,&a1);
    a2.l=m+1;
    a2.r=r;
    pthread_create(&tid2,NULL,thread_mergesort,&a2);
    // joining threads after being sorted
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    pthread_mutex_lock(&mutex);
    merge(arr,l,m,r);
    pthread_mutex_unlock(&mutex);
}
```

Analysis:

- The normal merge sort runs fastest and the threads sort runs the faster than the process as creating process as it takes more time and has more overhead.

For n=5

> this is the output I get in my terminal

- Enter number of elements:

> 5

- Enter the elements in the array :

> 4 5 2 1 6

- sorted array

> 1 2 4 5 6

- time taken by threaded_mergesort: 0.000372
- sorted array

> 1 2 4 5 6

- time taken by concurrent process_mergesort: 0.000001
- sorted array

> 1 2 4 5 6

- time taken by normal_mergesort: 0.000023

For n=10

- Enter number of elements:

> 10

- Enter the elements in the array :

> 5 6 7 9 1 2 7 4 5 6

- sorted array

> 1 2 4 5 5 6 6 7 7 9

- time taken by threaded_mergesort: 0.000730
- sorted array

> 1 2 4 5 5 6 6 7 7 9

- time taken by concurrent process_mergesort: 0.000854
- sorted array

> 1 2 4 5 5 6 6 7 7 9

- time taken by normal_mergesort: 0.000004

From the above results we can say that normal merge sort faster than the thread

merge sort,thread merge sort runs faster than the process merge sort.