Q1

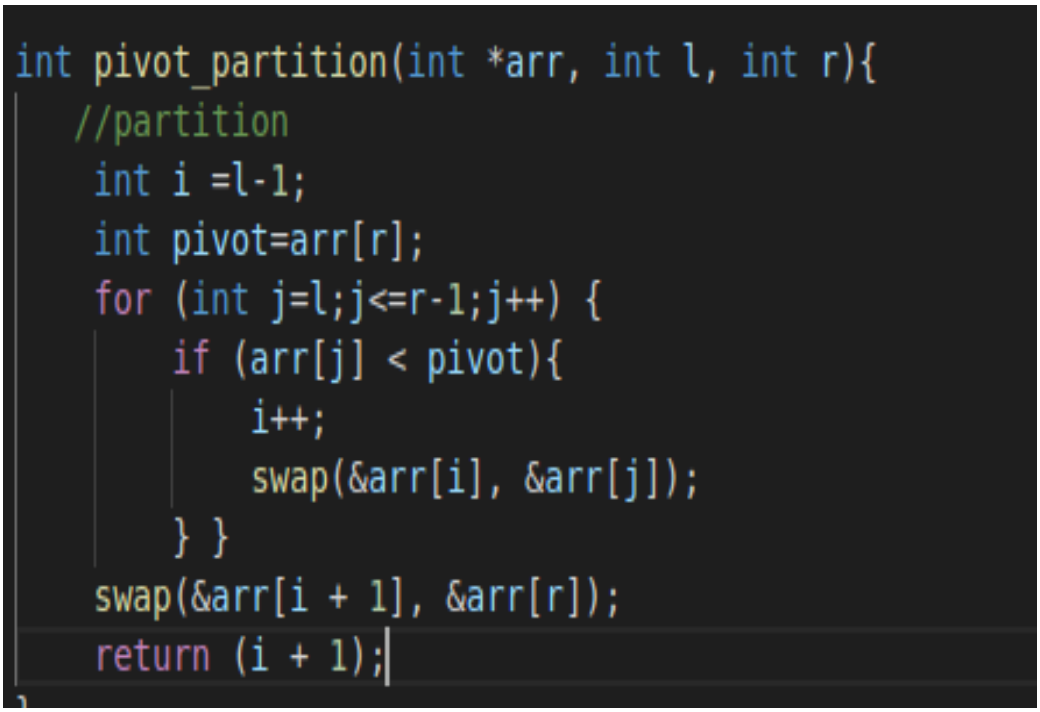
Concurrent Quicksort

In this question i do the normal quicksort,the concurrent quicksort(with shared memory) and the threaded quicksort.

Functions:

-->>partition function:

It fixes the pivot as the rightmost element and then places smaller elements than the pivot to the left of the pivot and greater elements to the right;



```
int pivot_partition(int *arr, int l, int r){
   //partition
    int i =l-1;
    int pivot=arr[r];
    for (int j=l;j<=r-1;j++) {
        if (arr[j] < pivot){
            i++;
            swap(&arr[i], &arr[j]);
        } }
    swap(&arr[i + 1], &arr[r]);
    return (i + 1);
```

-->>concurrent_qsort function:

It first sets the pivot by calling the pivot function and then create two processes and then recursively sorts the left part and the left part in the two different processes using the same shared memory;

```
void concurrent_qsort(int *arr, int l, int r){
    if(l>r) _exit(1);
    if(r-l+1<5){
        for(int i=l;i<r;i++){
            int j=i+1;
            for(;j<=r;j++)
                if(arr[j]<arr[i] && j<=r) {
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
        }
        return;
    }
    int piv=pivot_partition(arr, l, r);
    int pid1 = fork();
    int pid2;
    if(pid1==0){
        concurrent_qsort(arr, l, piv-1);
        _exit(1);
    }
    else{
        pid2 = fork();
        if(pid2==0){
            concurrent_qsort(arr,piv+1,r);
            _exit(1);
        }
        else{
            //wait for the right and the left half to get sorted
            int status;
            waitpid(pid1, &status, 0);
```

-->>threaded_quicksort In this function first I create a struct a1 and a2 which contains the left and the right index of the two subarrays to consider and the array and then sort both parts of the array in separate threads;

```
void *threaded_qsort(void* a){
    struct arg *args = (struct arg*) a;
    int l = args->l;
    int r = args->r;
    int *arr = args->arr;
    if(l>r) return NULL;
    if(r-l+1<5){
        int a[5], mi=INT_MAX, mid=-1;
        for(int i=l;i<r;i++){
            int j=i+1;
            for(;j<=r;j++)
                if(arr[j]<arr[i] && j<=r)
                {
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
        }
        return NULL;
    }
    int piv= pivot_partition(arr,l,r);
    struct arg a1;
    a1.l = l;
    a1.r = piv-1;
    a1.arr = arr;
    pthread_t tid1;
    //sort right half array
    pthread_create(&tid1, NULL, threaded_qsort, &a1);
    struct arg a2;
    a2.l = piv+1;
    a2.r = r;
    a2.arr = arr;
    pthread_t tid2;
    pthread_create(&tid2, NULL, threaded_qsort, &a2);
    //wait for the two halves to get sorted
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

//  int piv= pivot_partition(arr,l,r);
```

-->>launch function:

Here i create the shared memory for concurrent quick sort.Then I run all the three quick sort types(viz. normal quick sort,concurrent quick sort and threaded quicksort) and compare their running times.

```
void launch(long long int n){
    struct timespec ts;
    //getting shared memory
    int *arr = shareMem(sizeof(int)*(n+1));
    for(int i=0;i<n;i++) scanf("%d", arr+i);

    int brr[n+1];
    for(int i=0;i<n;i++) brr[i] = arr[i];

    printf("Running concurrent_qsort for n = %lld\n", n);
    printf("\nbefore sorting\n");
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    printf("\n");

    clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
    long double st = ts.tv_nsec/(1e9)+ts.tv_sec;
    //multiprocess concurrent_qsort
    concurrent_qsort(arr, 0, n-1);

    clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
    long double en = ts.tv_nsec/(1e9)+ts.tv_sec;
    printf("time = %Lf\n", en - st);
    long double t1 = en-st;

    printf("\nafter sorting\n");
        for(int i=0;i<n;i++) printf("%d ",arr[i]);
        printf("\n");
     for(int i=0;i<n;i++) arr[i] = brr[i];
```

```
pthread_t tid;
struct arg a;
a.l = 0;
a.r = n-1;
a.arr = brr;

printf("Running threaded_concurrent_qsort for n = %lld\n", n);
printf("\nbefore sorting\n");
for(int i=0;i<n;i++) printf("%d ",brr[i]);
printf("\n");

clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
st = ts.tv_nsec/(1e9)+ts.tv_sec;

//multithreaded concurrent_qsort
pthread_create(&tid, NULL, threaded_qsort, &a);
pthread_join(tid, NULL);

clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
en = ts.tv_nsec/(1e9)+ts.tv_sec;
printf("time = %Lf\n", en - st);
long double t2 = en-st;
    printf("\nafter sorting\n");
    for(int i=0;i<n;i++) printf("%d ",brr[i]);
    printf("\n");
 for(int i=0;i<n;i++) brr[i] = arr[i];


printf("Running normal_qsort for n = %lld\n", n);
 printf("\nbefore sorting\n");
```

Main function:>> In main function we scan length of array and just call the launch function.

```
int main(){
    long long int n;
    scanf("%lld", &n);
    launch(n);
    return 0;
}
```

Time taken by threaded_quicksort is less than
concurrent quicksort because in concurrent quicksort we create two process which
require th os to make virtual space which is overhead
but in thread we create thread for each process and the share data between each
other but this not a case in concurrent process in that we have to create shared
memory area so that the process can share data and accessing data will take time so
overall concurrent quicksort requires morre time.
The reason why both the above sort are slower as compare to normal quicksort is
because here the value of n(vo. of element to be sorted) is small so for small value it is
wastage of time and resource for CPU to create thread or forking a process .But as n
get larger the become faster as compered to normal
quicksort .Threaded quicksort will
take less time compared to concurrent quicksort.