OS 과제 보고서

[Multi-threaded sorting application]

12141579 윤찬미
010-9287-1679
cmmcme0604@gmail.com

I. 개요

• 구현의 목적

– 다양한 Thread 함수를 사용하며 Thread에 대한 이해를 높이고 Multi-threaded에 대한 이해도를 높인다.

• 요구 사항

– 파일 입출력을 사용하여 배열을 받아온다.
– <pthread.h> 를 include 하여 POSIX 함수를 이용한다.
– 배열을 전역 변수로 선언 한 후 세부분으로 나눈다.
– 나눈 부분을 각각의 Thread를 이용하며 정렬해준다
– 나눈 thread가 모두 끝날 때 까지 SpinLock을 해준다.
– 정렬된 배열을 새로운 thread를 이용하여 Merge 해준다.

II.Parameter 및 함수의기능

1. parameters

```
typedef struct parameters
{
    int from;
    int to;
} parameters;
```

– 구조체를 이용하여 parameters를 생성해 주었다.
– sort 함수를 위해 선언된 구조체이다.
– parameters 구조체 안에 정렬 할 배열의 처음과 끝을 받았다.

2. Thirdparameters

```
typedef struct Thirdparameters
{
    int from1;
    int from2;
    int from3;
} Thirdparameters;
```

– 구조체를 이용하여 Thirdparameters를 생성해 주었다.
– Merge 함수에서 사용 하기 위해 만들어진 구조체이다.
– 나눈 배열의 시작 점을 받는다.

• 함수의 기능

1.void *Sort(parameters *)

-> 파라미터 구조체가 가지고 있는 양 끝 점 사이를 bubble sort 하는 함수이다.

2. int Small(int *,int *,int *)

-> Merge를 할 때 세곳의 배열 위치중 가장 작은 값을 반환한다.

3. int TwoSmall(int *,int *)

-> Merge를 할 때 두곳의 배열 위치중 가장 작은 값을 반환한다.

4. void *Merge(Thirdparameters *)

-> 3개로 나눈 thread를 Merge하여 정렬된 배열을 만드는 함수이다.

III.함수 명세

```
freopen("input.txt","r",stdin);

while(scanf("%d",&arr[i])!=EOF)
    i++;

int a=i-1;
a=a/3;
int t1=i%3;

parameters *data1 = (parameters *) malloc (sizeof(parameters));
parameters *data2 = (parameters *) malloc (sizeof(parameters));
parameters *data3 = (parameters *) malloc (sizeof(parameters));

data1->from = 0;
data1->to = a-1;

data2->from = data1->to +1;
data2->to = data2->from + a-1;

data3->from = data2->to + 1;
data3->to = i-1;

if(t1==2)
    data2->to++;
```

main

- 파일 입출력을 통해 배열을 받아 배열의 크기를 저장한다.
- 3개로 나눈 부분의 처음과 끝을 각각 지정 해준다.

```
Thirdparameters *alldata=(Thirdparameters *) malloc (sizeof(Thirdparameters));

alldata->from1=data1->from;
alldata->from2=data2->from;
alldata->from3=data3->from;

pthread_t tid1; pthread_t tid2;
pthread_t tid3; pthread_t tid4;

while(sol!=3) {
pthread_create(&tid1, NULL, Sort, data1);
pthread_create(&tid2, NULL, Sort, data2);
pthread_create(&tid3, NULL, Sort, data3);

pthread_join(tid1, NULL);
pthread_join(tid2, NULL);
pthread_join(tid3, NULL);

}

pthread_create(&tid4,NULL,Merge, alldata);
 pthread_join(tid4,NULL);
```

- 세 파라미터의 처음 부분을 Merge에서 사용하는 파라미터에 저장한다.
- 4개의 pthread_id를 만들어 준다.
- thread는 총 3개 발생하므로 SpinLock를 걸어 3개 부분 모두 Sort가 끝날 때까지 기다린다.
- pthread_create 함수를 통해 배열의 각 부분을 Sort 해준다.
- 나눈 세 부분을 Merge 하여 배열을 재정렬 한다.

```
void *Sort(parameters *P)
{
     int start=P->from;
     int end=P->to;
     int k,j,temp=0;
     for(k=start;k<=end;k++)
     {
          for(j=start;j<end;j++)
          {
               if(arr[j+1]<arr[j])
               {
                    temp=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
               }
          }
     }

     sol++;

     pthread_exit(0);

}
```

Sort

- bubble sort를 이용하며 각 파라미터가 가지고 있는 처음과 끝 부분을 정렬하였다.

- SpinLock을 위해 sol을 ++ 해주었다..

- ptread_exit를 사용하여 thread를 종료 시킨다.

```
int Small(int *n1,int *n2,int *n3)
{
     int min=arr[*n1];

     if(min>arr[*n2])
          min=arr[*n2];

     if(min>arr[*n3])
          min=arr[*n3];


     if(min==arr[*n1])
          (*n1)++;
     else if(min==arr[*n2])
          (*n2)++;
     else if(min==arr[*n3])
          (*n3)++;


     return min;
}
```

Small

-n1과 n2와 n3에 위치한 배열의 값 중에서
가장 작은 것을 반환한다.

-이때 가장 작은 값을 가지는 배열의 위치는 한칸 이동한다.

```c
int TwoSmall(int *n1,int *n2)
{
    int min=arr[*n1];

    if(min>arr[*n2])
        min=arr[*n2];

    if(min==arr[*n1])
        (*n1)++;
    else if(min==arr[*n2])
        (*n2)++;

    return min;
}
```

TwoSmall

- n1과 n2에위치한 배열의 값 중에서
가장 작은 것을 반환한다.

-이때 가장 작은 값을 가지는 배열의 위치는 한칸 이동한다.

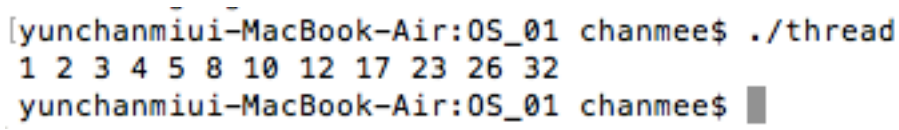
```c
void *Merge(Thirdparameters *T)
{
    int merge[1010];
    int start[3];
    int cnt=0;
    start[0]=T->from1;
    start[1]=T->from2;
    start[2]=T->from3;


    while (start[0]!=T->from2 || start[1]!=T->from3 || start[2]!=i)
    {
        if(start[0]==T->from2 && start[1]!=T->from3 && start[2]!=i)
            merge[cnt++]=TwoSmall(&start[1],&start[2]);

        else if(start[0]!=T->from2 && start[1]==T->from3  && start[2]!=i )
            merge[cnt++]=TwoSmall(&start[0],&start[2]);


        else if(start[0]!=T->from2 && start[1]!=T->from3 && start[2]==i )
            merge[cnt++]=TwoSmall(&start[0],&start[1]);


        else if(start[0]!=T->from2 && start[1]==T->from3 &&start[2]==i)
            merge[cnt++]=arr[start[0]++];


        else if(start[0]==T->from2 &&start[1]!=T->from3 &&start[2]==i)
            merge[cnt++]=arr[start[1]++];


        else if(start[0]==T->from2 && start[1]==T->from3 && start[2]!=i)
            merge[cnt++]=arr[start[2]++];

        else
            merge[cnt++]=Small(&start[0],&start[1],&start[2]);

    }


    for(int j=0;j<i;j++)
        printf("%d ",merge[j]);
    puts("");

    pthread_exit(0);

}
```

Merge

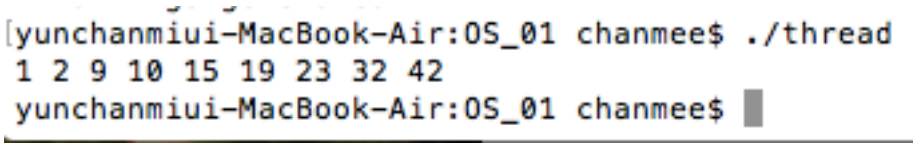
- Merge를 통해 각 각 정렬한 공간을 비교하여 또다시 Sort하는 과정이다
- merge 배열은 재정렬된 배열을 의미한다.
- start 배열에 각 파라미터의 가장 처음을 넣어준다.
- 만약 나눈 세 부분이 모두 다시 정렬 되지 않았다면 Small 함수를 통해 3개의 위치에서 배열값을 반환받아 작은 값을 merge 배열에 추가한다.
- 만약 한 부분을 제외한 TwoSmall 함수를 통해 두 부분만 정렬되지 않았다면 두 위치에서 배열 값을 비교 한 후 작은 값을 merge 배열에 추가한다.
- 한 부분을 제외한 모든 부분이 정렬되었을 경우 나머지 값들은 정렬 되어 있기 때문에 그 값을 merge 배열에 추가한다.
- 모든 배열이 정렬 된 후 while문을 빠져 나온 후 정렬된 배열을 출력한다.

IV.실행 화면

1)



2)



V.평가 및 개선 방향

-> Thread를 직접 생성해 봄으로서 다양한 POSIX 함수들이 사용 되는 법을 알게 되었다.

-> User Thread를 관리하는 Thread library에 대해 알게되었다.

-> 데이터를 분산시켜 동시에 수행하는 Data-Parallelism을 사용하였다.

-> LINUX 환경에 대해 더 익힐 수 있는 시간이 되었다.

-> thread를 사용하여 정렬 하는 것이 그냥 정렬을 하는 것보다 빨랐기에 여러개의 thread로 나눌 수록 더 빨라 질 수 있을 것 같다.

-> 여러개의 thread를 사용 할 경우 발생되는 Overhead를 주의하며 사용한다.

-> 배열을 정렬 할 시 O(N^2)의 수행시간이 걸리는 Bubble Sort보다 Quick Sort를 사용하면 평균 수행 시간이 더 빨라 질 수 있다.