情報科学演習第四期課題　マルチスレッド

高口 奨一郎
学籍番号：6319045

2021 年 1 月 13 日

1　課題１

1.1　課題の説明

座席予約システムのプログラムを作成せよ．指定された席数は，必ず連続した席にすること．もし，空席があっても指定された連続した席が存在しない場合は予約失敗扱いとして構わない．その結果，最終的に空席が残っても問題ない．

1.2　アルゴリズムの説明

座席を二次元配列で表す。空席には-1 を配列に格納し、予約済みの座席は予約者の id を配列に格納する。予約のリクエストがあれば、リクエストを受けた人数分の連続した座席が残っているか調べる。調べ方については条件分岐を用いて-1 が格納されている配列を探し出しカウンターを +1 する。-1 が連続するとカウンターが増えていくような仕組みを作り、カウンターの値がリクエストされた人数に達せば予約を実行する。そうでなければ予約は失敗になる。また、このプログラムでは複数のスレッドを用いて予約管理を行うためリクエストから予約完了までの過程は矛盾が生じないよう、ロックを行う。なお、今回の課題では授業で与えられたプログラムを用いて予約のリクエストは複数の顧客がランダム席数の予約を 10 回試みることとする。

1.3　プログラムの説明

予約を行う関数 reserve、その他与えられたプログラムを改変した箇所についてプログラムの説明を行う。

```
1     synchronized boolean reserve(int id, int num){
2        for(int i=0;i<seat.length;i++){
3         int counter=0;
4          for(int j=0;j<seat[0].length;j++){
5           if(seat[i][j]==-1){
6            counter++;
7             if(counter==num){
8              for(int empty=j-num+1;empty<j+1;empty++){
9               seat[i][empty]=id;
10              count++;
11             }
```

```
12              return true;
13              }
14            }
15           }
16         }
17        return false;
18    }
```

for 文を用いて座席を表す配列の要素にアクセスする。配列の要素が-1 であれば counter の値を 1 増やす。隣の席（配列）が空席でなければ（-1 でなければ）counter の値を 0 に戻す。counter の値が与えられた整数の num（予約リクエストの人数）の値と等しければ選択された座席に予約者の id を入れ（配列の要素に予約者の id を代入する）予約人数をカウントする変数 count の値を 1 増やす。また、予約が完了すれば true を返す。これによって run 関数内で true が出力された場合に予約内容を表すメッセージが出力される。予約が失敗すれば false を返す。これについても同様に run 関数内で予約失敗を表すメッセージを返す。また、今回のプログラムでは予約可能かどうか調べる段階から予約完了までのすべての過程についてロックを行うことが必要であるため、synchronized メソッドとした。

```
1     public void run(){
2           for(int i=0;i<10;i++){ １０回行う//
3            int num = (int)(Math.random()*6+1);
4             if(rs.reserve(id, num)){
5                   System.out.println("ID:"+id+"  reserved "+num+" seats.");
6             }
7           else{
8            System.out.println("ID:"+id+"  couldn't reserved "+num+" seats.");
9           }
10          }
11    }
```

予約失敗が正常に行われているかを明確にするために予約失敗の場合にメッセージを出力する処理を行う。

1.4 実行結果

以上のように正しく動作していることがわかる。

```
PS C:¥Users¥s_kou¥java> javac -encoding utf-8 Reservation.java
PS C:¥Users¥s_kou¥java> java Reservation
ID:0  reserved 2 seats.
ID:0  reserved 4 seats.
ID:0  reserved 4 seats.
ID:1  reserved 2 seats.
ID:0  reserved 3 seats.
ID:2  reserved 5 seats.
ID:1  reserved 4 seats.
ID:2  reserved 5 seats.
ID:2  reserved 4 seats.
ID:0  reserved 6 seats.
ID:3  reserved 1 seats.
ID:2  reserved 2 seats.
ID:1  reserved 6 seats.
ID:2  reserved 6 seats.
ID:3  reserved 2 seats.
ID:3  reserved 4 seats.
ID:3  reserved 6 seats.
ID:3  reserved 5 seats.
ID:3  reserved 1 seats.
ID:0  reserved 1 seats.
ID:3  couldn't reserved 6 seats.
ID:2  reserved 5 seats.
ID:4  reserved 4 seats.
ID:4  reserved 4 seats.
ID:4  couldn't reserved 4 seats.
ID:4  couldn't reserved 2 seats.
ID:1  reserved 2 seats.
ID:4  couldn't reserved 6 seats.
ID:2  couldn't reserved 5 seats.
ID:3  reserved 2 seats.
ID:0  couldn't reserved 6 seats.
ID:3  couldn't reserved 4 seats.
ID:2  couldn't reserved 4 seats.
ID:2  couldn't reserved 2 seats.
ID:2  couldn't reserved 3 seats.
ID:4  couldn't reserved 6 seats.
ID:1  couldn't reserved 4 seats.
ID:4  couldn't reserved 1 seats.
ID:4  couldn't reserved 4 seats.
ID:3  couldn't reserved 2 seats.
ID:0  couldn't reserved 3 seats.
ID:4  couldn't reserved 1 seats.
ID:4  couldn't reserved 3 seats.
ID:1  couldn't reserved 2 seats.
ID:0  couldn't reserved 3 seats.
ID:1  couldn't reserved 3 seats.
ID:0  couldn't reserved 3 seats.
ID:1  couldn't reserved 4 seats.
ID:1  couldn't reserved 6 seats.
ID:1  couldn't reserved 6 seats.
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 2 2 2 2 2 0 0 0 0 0 0
2 2 2 2 2 2 1 1 1 1 1 2 2 2 2
2 2 3 0 3 3 2 2 2 2 2 2 1 1 3
4 4 4 4 2 2 2 2 2 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 4 4 4 4
reserved number of person90
```

1.5 考察

1.5.1 考察課題

排他制御について理解を深めるため reserve メソッドを synchronized メソッドにしなかった場合にはどのようなことが起こるのかについて考察を行った。まず、以下のように printSeat メソッドを改変した。

1.5.2 プログラムの説明

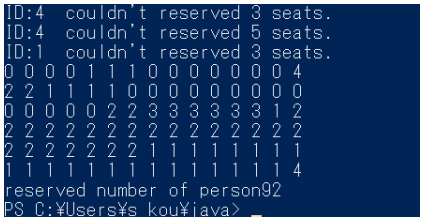
```
1       void printSeat(){
2               for(int i=0;i<seat.length;i++){
```

```
3               for(int j=0;j<seat[i].length;j++){
4                       System.out.print(seat[i][j]+" ");
5               }
6               System.out.println();
7              }
8       System.out.println("reserved number of person"+count);
9      }
```

上の関数で求めた予約が確定した人数の総数を出力する処理を行った。

1.5.3 実行結果

synchronized メソッドにした場合と同じように正しく動作する場合もあるが以下のように本来予約人数は 90 人以下であるはずであるのに対してそれより多くの人が予約をしているというような結果になった。つまり予約が重複している席があるということである。



これは予約を複数のスレッドが行うため、あるスレッドが連続している席が存在しているかカウントしている、または予約を行っている段階であれば他のスレッドが同じ席にアクセスすればその席は空席であるというデータを受け取ってしまうからであると考えられる。実際に自ら作成したプログラムを用いて実験を行うことでロックの重要性をよく理解することができた。

1.6 プログラムソース

```
1  public class Reservation{
2      int seat[][];
3      public int count = 0;
4      public Reservation(int n, int m){
5        seat = new int[n][m];
6               for(int i=0;i<n;i++){
7                for(int j=0;j<m;j++){
8                       seat[i][j]=-1;
9                }
10              }
```

```
11      }
12
13      synchronized boolean reserve(int id, int num){
14         for(int i=0;i<seat.length;i++){
15          int counter=0;
16           for(int j=0;j<seat[0].length;j++){
17            if(seat[i][j]==-1){
18             counter++;
19              if(counter==num){
20               for(int empty=j;empty>j-num;empty--){
21                seat[i][empty]=id;
22                count++;
23               }
24              return true;
25              }
26            }
27           }
28         }
29        return false;
30      }
31
32      void printSeat(){
33             for(int i=0;i<seat.length;i++){
34              for(int j=0;j<seat[i].length;j++){
35                     System.out.print(seat[i][j]+" ");
36              }
37              System.out.println();
38             }
39       System.out.println("reserved number of person"+count);
40      }
41
42      public static void main(String args[]){
43             int thread_num = 5;
44             Reservation rs = new Reservation(6,15);
45             Passengers ps[] = new Passengers[thread_num];
46             for(int i=0;i<thread_num;i++){
47              ps[i] = new Passengers(i,rs);
48             }
49             for(int i=0;i<thread_num;i++){
50              try{
51                     ps[i].join();
52              }catch(InterruptedException e){
53              }
54             }
55             rs.printSeat();
```

```
56      }
57  }
58
59
60  class Passengers extends Thread{
61    int id;
62    Reservation rs;
63     public Passengers(int n, Reservation rs){
64            this.id = n;
65            this.rs = rs;
66            this.start();
67     }
68
69     public void run(){
70            for(int i=0;i<10;i++){
71             int num = (int)(Math.random()*6+1);
72              if(rs.reserve(id, num)){
73                     System.out.println("ID:"+id+"  reserved "+num+" seats.");
74              }
75        else{
76         System.out.println("ID:"+id+"  couldn't reserved "+num+" seats.");
77        }
78
79            }
80     }
81  }
```

2　課題２

2.1　課題の説明

整数型配列のソートのプログラムを作成せよ．ソートは並列で行い，以下にサンプルとして記載する逐次バブルソートとの速度比較を考察せよ．配列の生成等は以下の Sort クラスを利用し，実際のソート処理を行う BubbleSort クラスの部分を他の並列ソートに変更すること．最低でもこの逐次バブルソートよりは速度向上させること．

2.2　アルゴリズムの説明

与えられた配列を二つに分割し、分割したそれぞれの配列についてマージソートを行う。この作業を二つのスレッドを用いて並列で行うことにより、実行速度の向上を図る。ソート済みの二つの配列をマージすればソート終了となる。マージソートについては与えられた配列を要素数が 1 になるまで分解し、分解した配列を整列させながら併合（マージ）する。

2.3　プログラムの説明

MergeSort クラス、その他与えられたプログラムを改変した箇所についてプログラムの説明を行う。

2.3.1　Sort

```
1  public class Sort {
2      private int array[];
3      private int arrayh[];
4      private int arrayr[];
5      public Sort(int n){
6       array = new int[n];
7       arrayh = new int[n/2];
8       arrayr = new int[n-n/2];
9        for(int i=0;i<n;i++){
10             array[i] = (int)(Math.random()*Integer.MAX_VALUE);
11             }
12       long start = System.currentTimeMillis();
13       for(int i=0;i<n/2;i++){
14               arrayh[i] = array[i];
```

```
15              }
16        int m = n/2;
17        for(int i=m;i<n;i++){
18                arrayr[(i-m)] = array[i];
19              }
20
21      MergeThread m1=new MergeThread(arrayh);
22      MergeThread m2=new MergeThread(arrayr);
23      m1.start();
24      m2.start();
25      try{
26       m1.join();
27       m2.join();
28      }catch(InterruptedException e){
29      }
30      /** ソート結果を得るメソッド */
31            array=merge(m1.array,m2.array,array);
32            //printArray(array);
33            long end = System.currentTimeMillis();
34            System.out.println("sort?: "+sortCheck(array)+
35                               ", Processing time: "+(end-start)+"ms");
36      }
```

3,4 行目は最初に分割した配列を格納するための配列である。9-19 行目において与えられた配列の左側を arrayh、右側を arrayr とする。21,22 行目においてこの二つの配列を MergeThread に渡し、ソートを実行する。最後に 31 行目においてソートされた二つの配列を merge することで配列のソートが完了する。なお、join メソッドを用いているのは二つの配列のソートが完了する前に二つの配列を merge することを防ぐためである。

2.3.2 merge

```
1       public int[] merge(int[] arrayh, int[] arrayr,int[] array){
2        int i=0;
3        int j=0;
4        while(i<arrayh.length || j<arrayr.length){
5         if(j==arrayr.length || (i<arrayh.length && arrayh[i]<arrayr[j])){
6          array[i+j]=arrayh[i];
7          i++;
8         }
9         else{
10         array[i+j]=arrayr[j];
11         j++;
```

```
12        }
13       }
14       return array;
15      }
```

merge メソッドはソート済みの二つの配列を併合した配列をもとの配列（array）に格納するメソッドである。条件分岐を用いて arrayh、arrayr の小さい方から順番に要素を取り出して配列 array に格納する。

2.3.3 merge

```
1  private int[] mergeSort(int[] array){
2    int n = array.length;
3     if(n>1){
4       int[] array1=new int[n/2];
5       int[] array2=new int[n-n/2];
6       for(int i=0;i<n/2;i++){
7         array1[i] = array[i];
8       }
9       int m = n/2;
10      for(int i=m;i<n;i++){
11        array2[(i-m)] = array[i];
12      }
13      mergeSort(array1);
14      mergeSort(array2);
15      merge(array1,array2,array);
16    }
17    return array;
18  }
19 }
20 }
```

mergeSort メソッドは与えられた配列を二つに分割ことを再帰的に繰り返すことで配列の大きさを 1 にする。これらの配列に対して merge メソッドを繰り返し呼び出すことによって与えられた配列をソートすることができる。

2.3.4 MergeThread

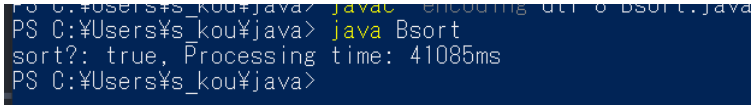
```
1 class MergeThread extends Thread{
2   public int[] array;
3   public MergeThread(int[] n){
4   array=n;
5   }
6   public void run(){
```

```
7     MergeSort m1=new MergeSort(array);
8     array=m1.array;
9     }
10 }
```

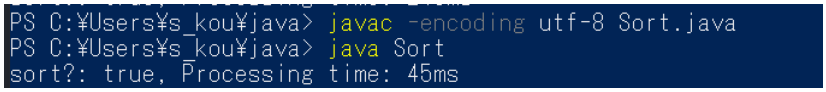
MergeThread クラスは複数のスレッドを使用することを可能にするために Thread クラスを継承している。run 関数内で MergeSort に array を渡すことでマージソートを実行している。

2.4 実行結果

2.4.1 遂次バブルソートを用いた場合



2.4.2 並列処理を行うマージソートを用いた場合



以上のように正しくソートが実行され、与えられた遂次バブルソートより実行速度が向上していることが分かる。

2.5 考察

2.5.1 考察課題

マージソートとは与えられた配列を分割し、併合していくものであるから並列アルゴリズムと相性がよいアルゴリズムであると考え、スレッドの数を増やせば増やすほど実行速度が向上するのではないかと考え、実験してみた。

2.5.2 実験方法

上で用いたプログラムにおいてスレッドの数を 1,2,4,8 として実行時間の計測を行う。

配列の要素数	1000	100000	1000000	10000000
スレッド数 1	7	60	248	3299
スレッド数 2	8	66	214	1671
スレッド数 4	8	70	164	1176
スレッド数 8	8	71	135	890

2.5.3 実行結果

要素数によって異なる結果が得られた。要素数が少ない場合にはスレッドの数が増えてもあまり実行時間に差はないかむしろスレッドの数が少ない方が実行時間が短くなった。要素数がある一定の数字を超えるとスレッドの数が多ければ多いほど実行速度は飛躍的に向上した。要素数を 10000000 とするとスレッドの数が 8 倍になると実行時間は約 4 分の 1 に短縮された。これは要素数が少なければ与えられた配列をスレッドの数に分割する処理やスレッドを立ち上げる処理に時間を要するためにこのような結果が得られたのではないかと考える。逆に要素数が多い場合には実行時間におけるソートに要する時間の割合が高くなるためにスレッドの数を増やせば増やすほど実行時間の短縮に繋がったのではないかと考える。

2.5.4 結論

実行時間の比較を行う前はスレッドの数を増やせば増やすほど実行時間の短縮に繋がるのではないかと考えていたが、マルチスレッドプログラミングをする上ではソートする要素数やコンピュータの性能、コア数などによって一概にスレッドの数を増やせばいいものではないことを体感することができた。これがプログラミングの面白いところであると感じた。コンピュータの性能やプログラムの他の側面にも目を向けてこれからも学習していく。

2.6 プログラムソース

```
1  public class Sort {
2      private int array[];
3      private int arrayh[];
4      private int arrayr[];
5      public Sort(int n){
6              array = new int[n];
7       arrayh = new int[n/2];
```

```
 8       arrayr = new int[n-n/2];
 9        for(int i=0;i<n;i++){
10              array[i] = (int)(Math.random()*Integer.MAX_VALUE);
11              }
12         long start = System.currentTimeMillis();
13         for(int i=0;i<n/2;i++){
14                arrayh[i] = array[i];
15              }
16         int m = n/2;
17         for(int i=m;i<n;i++){
18                arrayr[(i-m)] = array[i];
19              }ここからソート呼び出し
20             //
21      MergeThread m1=new MergeThread(arrayh);
22      MergeThread m2=new MergeThread(arrayr);
23      long start = System.currentTimeMillis();
24      m1.start();
25      m2.start();
26      try{
27       m1.join();
28       m2.join();
29      }catch(InterruptedException e){
30      }
31      /** ソート結果を得るメソッド */
32            array=merge(m1.array,m2.array,array);
33            //printArray(array); 配列表示//
34            long end = System.currentTimeMillis();
35            System.out.println("sort?: "+sortCheck(array)+
36                               ", Processing time: "+(end-start)+"ms");
37      }
38
39      /** ソートチェック */
40      public static boolean sortCheck(int array[]){
41             for(int i=0;i<array.length-1;i++){
42              if(array[i]>array[i+1])return false;
43             }
44             return true;
45       }
46
47      /** 確認用の配列表示メソッド */
48      public static void printArray(int array[]){
49             for(int i=0;i<array.length;i++){
50              System.out.print(array[i]+" ");
51             }
52             System.out.println();
```

```
53      }
54
55      public int[] merge(int[] arrayh, int[] arrayr,int[] array){
56       int i=0;
57       int j=0;
58       while(i<arrayh.length || j<arrayr.length){
59        if(j==arrayr.length || (i<arrayh.length && arrayh[i]<arrayr[j])){
60         array[i+j]=arrayh[i];
61         i++;
62        }
63        else{
64         array[i+j]=arrayr[j];
65         j++;
66        }
67       }
68       return array;
69      }
70
71      public static void main(String args[]){
72              new Sort(100000);
73      }
74
75
76  /**マージソート
77
78  */
79
80  class MergeSort {
81    public int array[];
82    MergeSort(int[] n){
83    array = n;
84    sort();
85    }
86    private void sort(){
87     array=mergeSort(array);
88    }
89
90  /** ソートコンストラクタから自動で実行される  */
91   private int[] mergeSort(int[] array){
92     int n = array.length;
93      if(n>1){
94        int[] array1=new int[n/2];
95        int[] array2=new int[n-n/2];
96        for(int i=0;i<n/2;i++){
97          array1[i] = array[i];
```

```
98        }
99        int m = n/2;
100       for(int i=m;i<n;i++){
101         array2[(i-m)] = array[i];
102       }
103       mergeSort(array1);
104       mergeSort(array2);
105       merge(array1,array2,array);
106     }
107     return array;
108   }
109 }
110 }
111
112 class MergeThread extends Thread{
113   public int[] array;
114   public MergeThread(int[] n){
115   array=n;
116   }
117   public void run(){
118   MergeSort m1=new MergeSort(array);
119   array=m1.array;
120   }
121 }
```