上海交通大學

《操作系统》课程

学生实验报告

实验名称: Project3

姓　　名: 洪瑄锐

学　　号: 517030910227

班　　级: F1703302

手　　机: 15248246044

任课老师: 吴晨涛

2019 年 6 月 1 日

目录

一、实验要求

3-1 Multithread Sorting Application

编写一个多线程排序程序，其工作方式如下：整数列表分为两个相同大小的较小列表，两个独立的线程使用某个排序算法对每个子列表进行排序，这两个子列表由第三个线程合并。最后由父线程输出排好序的列表。

3-2 Fork-Join Sorting Application

使用 Java 的 fork-join 并行 API 实现前面的项目，该项目将以两种不同的版本开发，每个版本将实现一个不同的分而治之的排序算法：QuickSort 和 MergeSort。

二、程序设计思想及代码解释

I.Multithread Sorting Application

1.QuickSort 伪代码

```
Algorithm Partition(X,Left,Right)
Input: X (an array), Left (the left boundary of the array), Right (the right boundary)
Output: X and Middle such that X[i]≤X[Middle] for all i≤Middle and X[j]>X[Middle]
    for all j>Middle
Begin
    pivot:=X[Left];
    L:=Left; R:=Right;
    while L<R do
        while X[L]≤pivot and L≤Right do L:=L+1;
        while X[R]>pivot and R≥Left do R:=R-1;
        if L<R then exchange X[L] with X[R];
    Milddle:=R;
    exchange X[Left] with X[Middle];
end

Algorithm QUICKSORT(X,n)
Input: X (an array in the range 1 to n)
Output: X (the array in sorted order)
begin
```

Q_Sort(1,n)

Procedure Q_Sort(Left,Right)

Begin

if Left<Right then

Partition(X,Left,Right);

Q_Sort(Left,Middle-1);

Q_Sort(Middle+1,Right);

end

```
int partition(int left, int right)
{
    int pivot = array[left];
    int l = left;
    int r = right;
    int middle;
    int tmp;
    while (l < r)
    {
        while (l <= right && array[l] <= pivot)
        {
            l++;
        }
        while (r >= left && array[r] > pivot)
        {
            r--;
        }
        if (l < r)
        {
            tmp = array[l];
            array[l] = array[r];
            array[r] = tmp;
        }
    }
    middle = r;
    tmp = array[left];
    array[left] = array[middle];
    array[middle] = tmp;
    return middle;
}
```

```
void QuickSort(int left,int right)
{
    int mid;
    if (left < right)
    {
        mid = partition(left, right);
        QuickSort(left, mid-1);
        QuickSort(mid+1, right);
    }
}
```

2.Merge 伪代码

Begin

if(i<=num_1 && j<=num_2)

if(array[i]<array[j])

a[index]=array[i];

i++;index++;

else

a[index]=array[j];

j++;index++;

else

将剩余按照原顺序放入数组 a 中

End

```
void merge()
{
    int i = 0;
    int j = 50;
    int index = 0;

    while (i <= 49 && j <= 99)
    {
        if (array[i] <= array[j])
        {
            array_after[index] = array[i];
            index++;
            i++;
        }
        else
        {
            array_after[index] = array[j];
            index++;
            j++;
        }
    }
    while (j <= 99)
    {
        array_after[index] = array[j];
        j++;
        index++;
    }
    while (i <= 49)
    {
        array_after[index] = array[i];
        i++;
        index++;
    }

}
```

3.创建线程执行分别快速排序和合并排序，利用 runner 中的 param 参数辨认是哪一个线程，从而执行其特定任务。

创建线程步骤：

```
pthread_t tid;/*the thread identifier*/
pthread_attr_t attr;/*set of thread attributes*/

/*set the default attributes of the thread*/
pthead_attr_init(&attr);
/*create the thread*/
Pthread_creat(&tid,&attr,runner,th1);//th1 是为本项目设计的针对每个线程的标识
/*wait for the thread to exit*.
pthread_join(tid,NULL)
```

对于本项目来说，注意等待线程 1 和线程 2 将各自部分排序完毕后再创建线程 3 合并。

```
char *th1 = "one";
char *th2 = "two";
char *th3 = "three";

pthread_t tid1;
pthread_attr_t attr1;

pthread_t tid2;
pthread_attr_t attr2;

pthread_t tid3;
pthread_attr_t attr3;


pthread_attr_init(&attr1);
pthread_create(&tid1, &attr1, runner, th1);

pthread_attr_init(&attr2);
pthread_create(&tid2, &attr2, runner, th2);

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

pthread_attr_init(&attr3);
pthread_create(&tid3, &attr3, runner, th3);

pthread_join(tid3, NULL);
```

4. 执行函数

辨认每个线程的表示，分别工作

```
void* runner(void *param)
{
    if (strcmp(param, "one") == 0)
    {
        QuickSort(0, 49);
    }
    else if (strcmp(param, "two") == 0)
    {
        QuickSort(50, 99);
    }
    else if(strcmp(param,"three")==0)
    {
        merge();
    }

    pthread_exit(0);
}
```

II. QuickSort.java

• 围绕 ForkJionTask 类进行扩展得到不返回结果的 RecursiveAction 类。

• 在类的 compute()函数中，当数组数据数量足够小时，直接利用选择排序算法将数组排序，否则进行快速排序，大致思路为选数组最左边的数作为轴，将数组分为两半，该数左边小于该数，该数右边大于等于该数，然后使用 fork()创建新任务，将该数的左右两半分别进行快速排序，再调用 join()直至该任务完成。

•在主函数中，创建一个 ForkJoin 对象，并通过其 invoke()提交初始任务，

初始化数组 array。

```
protected void compute() {
  if (end - begin < THRESHOLD)
        {
                for(int i = end;i>begin;i--)
                {
                        int max = 0;
                        int max_index = 0;
                        for(int j = 0; j <= i; j++)
                        {
                                if(array[j]>max)
                                {
                                        max_index = j;
                                        max = array[j];
                                }
                        }
                        array[max_index] = array[i];
                        array[i] = max;
                }
  }
  else {
    // divide stage
                int l = begin;
                int r = end;
                int i = l;
                int j = r;
                int x = array[l];
                while (i < j)
                {
                        while (i < j && array[j] >= x)
                                j--;
                        if (i < j)
                                array[i++] = array[j];
                        while (i < j && array[i]< x)
                        if (i < j)
                                array[j--] = array[i];
                }
                array[i] = x;

                QuickSort leftTask = new QuickSort(begin, i - 1, array);
                QuickSort rightTask = new QuickSort(i + 1, end, array);

                leftTask.fork();
                rightTask.fork();

               leftTask.join();
              rightTask.join();
  }
}
```

```
public static void main(String[] args) {
		ForkJoinPool pool = new ForkJoinPool();
		int[] array = new int[SIZE];


		java.util.Random rand = new java.util.Random();
		for (int i = 0; i < SIZE; i++) {
			array[i] = rand.nextInt(SIZE);
		}
		System.out.print("Unsorted array:");

		for(int i = 0; i < SIZE; i++)
		{
			System.out.print(array[i]+" ");

		}

		System.out.println("\n\n");

		QuickSort task = new QuickSort(0, SIZE-1, array);

		pool.invoke(task);

		System.out.print("Sorted array:");
		for(int i = 0; i < SIZE; i++)
		{
			System.out.print(array[i]+" ");
		}
		System.out.println("");

}
```

III.MergeSort.java

• 围绕 ForkJionTask 类进行扩展得到不返回结果的 RecursiveAction 类。

• 在类的 compute（）函数中，当数组数据数量足够小的时候，直接利用插入排序算法将数组排序，否则在分割步骤使用 fork（）创建新任务，再调用 join（）直至该任务完成，最后利用 merge 函数合并两个子任务。

•在主函数中，创建一个 ForkJoin 对象，并通过其 invoke()提交初始任务，初始化数组 array。

```
    protected void merge(int begin, int end, int[] array)
  {
                        int mid = (begin + end) / 2;
                        int i = begin;
                        int j = mid + 1;
                        int k = 0;
                        int[] tmp = new int[end - begin + 1];
                        while (i <= mid && j <= end)
                        {
                                if (array[i]<array[j])
                                        tmp[k++] = array[i++];
                                else
                                        tmp[k++] = array[j++];
                        }
                        while (j <= end)
                                tmp[k++] = array[j++];
                        while (i <= mid)
                                tmp[k++] = array[i++];

                        for (i = begin; i <= end; i++)
                        {
                                array[i] = tmp[i - begin];
                        }
  }



protected void compute() {
  if (end - begin < THRESHOLD)
        {
                for (int i = begin; i < end; i++)
                        {
                                for(int j=i+1;j<=end;j++)
                                        if (array[i] > array[j])
                                        {
                                                int tmp = array[i];
                                                array[i] = array[j];
                                                array[j] = tmp;
                                        }
                        }

  }
  else {

                int mid = (begin + end) / 2;
                mergeSort leftTask = new mergeSort(begin, mid, array);
                mergeSort rightTask = new mergeSort(mid + 1, end, array);

                leftTask.fork();
                rightTask.fork();
                leftTask.join();
                rightTask.join();

                merge(begin, end, array);

  }
}
```

```
public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool();
        int[] array = new int[SIZE];

        System.out.print("Unsorted array:");
        java.util.Random rand = new java.util.Random();
        for (int i = 0; i < SIZE; i++)
         {
                array[i] = rand.nextInt(SIZE);
                System.out.print(array[i]+" ");

        }


        System.out.println("\n\n");

        mergeSort task = new mergeSort(0, SIZE-1, array);

        pool.invoke(task);

        System.out.print("Sorted array:");
        for(int i = 0; i < SIZE; i++)
        {
                System.out.print(array[i]+" ");
        }

    }
}
```

三、运行结果截图

1.Multithread Sorting Application

```
osc@ubuntu:~/final-src-osc10e/ch4$ ./pr2
Unsorted array:
52 7 43 3 62 47 7 4 91 70 59 83 60 45 31 89 40 39 7 37 80 94 87 82 71 94 49 55 80 0 89 84 7 85 87 69
 32 95 25 75 17 85 11 29 82 94 18 22 85 77 11 65 71 50 47 42 44 48 49 76 49 39 61 56 24 0 78 8 95 55
 35 64 40 98 93 74 92 63 48 29 40 11 46 63 62 94 57 6 94 7 83 43 98 96 52 74 96 30 34 44

Sorted Array:
0 0 3 4 6 7 7 7 7 7 8 11 11 11 17 18 22 24 25 29 29 30 31 32 34 35 37 39 39 40 40 40 42 43 43 44 44
45 46 47 47 48 48 49 49 49 50 52 52 55 55 56 57 59 60 61 62 62 63 63 64 65 69 70 71 71 74 74 75 76 7
7 78 80 80 82 82 83 83 84 85 85 85 87 87 89 89 91 92 93 94 94 94 94 94 95 95 96 96 98 98
```

2.QuickSort.java

```
osc@ubuntu:~/final-src-osc10e/ch4$ java QuickSort
Unsorted array:280 160 307 491 418 232 448 69 381 483 72 423 430 290 65 380 142 294 430 466 446 96 1
09 347 437 256 249 420 383 159 319 228 165 40 59 398 161 48 368 348 28 33 228 476 291 47 321 413 387
 368 26 227 265 92 354 256 218 93 59 359 205 132 482 440 219 279 28 306 448 100 398 213 309 224 158
199 288 330 217 184 40 79 272 483 62 385 427 185 373 384 125 271 343 361 196 260 260 267 264 223 180
 144 199 91 453 160 300 444 211 422 373 6 417 404 428 178 373 185 128 110 284 166 58 378 224 37 386
405 142 278 333 117 328 285 450 450 124 297 408 334 351 167 56 227 230 170 370 336 87 239 126 317 16
4 419 194 181 300 80 3 474 347 461 54 376 465 450 24 27 222 55 90 483 481 149 359 126 183 291 443 45
7 363 224 453 261 5 318 393 437 148 199 313 320 289 364 279 428 2 499 69 238 487 297 108 450 320 369
 345 120 355 21 13 409 454 323 477 168 98 308 176 312 323 229 89 466 373 347 8 65 3 232 128 226 346
172 283 87 357 218 347 296 276 149 170 48 159 472 77 94 341 426 481 24 94 18 183 89 327 333 242 75 1
73 214 266 300 197 44 40 478 31 157 417 203 265 303 9 410 218 183 38 496 365 421 210 35 40 127 450 1
97 203 299 299 338 471 276 395 207 407 265 181 263 214 20 243 116 333 482 378 75 376 108 52 110 81 3
08 146 259 214 330 271 313 385 193 157 174 385 374 347 132 403 39 316 354 443 493 256 356 61 28 362
196 160 72 270 279 223 349 492 150 111 89 45 270 409 315 323 395 296 216 445 346 131 89 120 35 169 1
75 353 102 198 199 311 399 351 26 453 35 441 102 351 66 279 80 213 411 326 200 137 334 255 3 433 434
 221 160 82 214 188 331 381 295 30 192 175 413 92 403 384 424 440 329 64 365 275 373 231 328 142 170
 279 405 431 438 89 116 90 320 104 447 257 400 405 127 110 471 20 454 227 370 237 371 132 410 100 31
3 290 386 71 382 495 251 125 37 191 1 355 87 414 301 423 466 384 154 461 172 270 127 4 446 429 95 73
 214 177 163 360 393 297 234 48 105 11 452 128 26 92 437 495 367 365 290 348 388 83 301 237 285 490
77 146 358
```

```
Sorted array:1 2 3 3 3 4 5 6 8 9 11 13 18 20 20 21 24 24 26 26 26 27 28 28 28 30 31 33 35 35 35 37 3
7 38 39 40 40 40 40 44 45 47 48 48 48 52 54 55 56 58 59 59 61 62 64 65 65 66 69 69 71 72 72 73 75 75
 77 77 79 80 80 81 82 83 87 87 87 89 89 89 89 89 90 90 91 92 92 92 93 94 94 95 96 98 100 100 102 102
 104 105 108 108 109 110 110 110 111 116 116 117 120 120 124 125 125 126 126 127 127 127 128 128 128
 131 132 132 132 137 142 142 142 144 146 146 148 149 149 150 154 157 157 158 159 159 160 160 160 160
 161 163 164 165 166 167 168 169 170 170 170 172 172 173 174 175 175 176 177 178 180 181 181 183 183
 183 184 185 185 188 191 192 193 194 196 196 197 197 198 199 199 199 199 200 203 203 205 207 210 211
 213 213 214 214 214 214 214 216 217 218 218 218 219 221 222 223 223 224 224 224 226 227 227 227 228
 228 229 230 231 232 232 234 237 237 238 239 242 243 249 251 255 256 256 256 257 259 260 260 261 263
 264 265 265 265 266 267 270 270 270 271 271 272 275 276 276 278 279 279 279 279 279 280 283 284 285
 285 288 289 290 290 290 291 291 294 295 296 296 297 297 297 299 299 300 300 300 301 301 303 306 307
 308 308 309 311 312 313 313 313 315 316 317 318 319 320 320 320 321 323 323 323 326 327 328 328 329
 330 330 331 333 333 333 334 334 336 338 341 343 345 346 346 347 347 347 347 348 348 349 351 351
 351 353 354 354 355 355 356 357 358 359 359 360 361 362 363 364 365 365 365 367 368 368 369 370 370
 371 373 373 373 373 373 374 376 376 378 378 380 381 381 382 383 384 384 384 385 385 385 386 386 387
 388 393 393 395 395 398 398 399 400 403 403 404 405 405 405 407 408 409 409 410 410 411 413 413 414
 417 417 418 419 420 421 422 423 423 424 426 427 428 428 429 430 430 431 433 434 437 437 437 438 440
 440 441 443 443 444 445 446 446 447 448 448 450 450 450 450 450 452 453 453 453 454 454 457 461 461
 465 466 466 466 471 471 472 474 476 477 478 481 481 482 482 483 483 483 487 490 491 492 493 495 495
 496 499
```

3.MergeSort.java

```
osc@ubuntu:~/final-src-osc10e/ch4$ java mergeSort
Unsorted array:79 67 30 227 197 467 77 421 57 104 436 133 344 310 390 229 433 343 446 89 139 131 340
 296 261 184 79 420 267 2 221 326 313 467 91 307 83 380 417 486 499 380 170 151 32 181 481 35 480 54
 122 390 169 32 8 422 341 2 261 282 359 100 465 343 463 354 461 438 49 270 287 13 115 388 178 324 47
3 307 211 233 82 214 255 291 117 225 445 463 357 213 284 189 367 366 106 229 245 364 141 124 289 202
 63 413 172 236 133 282 113 7 444 472 345 419 467 302 268 36 269 23 120 486 364 410 51 24 68 264 407
 93 146 253 193 366 489 195 370 31 174 136 94 420 52 54 247 416 494 130 234 194 334 275 222 413 456
157 390 362 231 366 122 168 457 380 426 4 389 54 305 114 42 141 284 132 247 267 80 222 121 355 387 2
60 490 134 79 9 152 422 121 297 40 493 333 281 433 459 115 218 427 246 162 288 412 343 209 286 105 2
5 6 201 368 48 359 161 171 404 356 71 117 265 20 412 45 41 420 26 435 444 118 126 447 8 416 287 409
228 471 111 254 323 51 472 422 447 137 29 423 438 480 6 131 358 479 190 379 397 136 465 374 109 493
103 473 419 471 100 432 165 347 321 383 50 122 4 471 34 415 48 444 320 484 23 453 36 401 394 126 389
 355 271 191 416 313 366 213 483 307 306 14 386 145 490 466 20 4 144 393 250 162 434 452 44 230 350
368 17 412 233 398 270 287 391 485 214 218 241 243 61 61 163 210 215 457 441 168 249 439 97 225 39 2
37 300 49 245 340 163 307 64 30 28 276 431 108 42 369 399 64 33 47 349 172 300 153 48 271 314 165 25
2 425 59 301 1 211 36 78 204 333 489 6 356 48 52 302 173 200 472 215 467 170 431 217 117 0 468 231 2
1 214 453 382 105 1 338 125 496 11 324 194 11 358 117 391 478 281 147 289 55 325 492 30 206 402 486
326 60 270 144 463 170 475 54 445 200 308 409 496 204 286 31 479 57 269 498 135 106 319 243 382 377
390 211 26 444 162 362 364 483 481 348 433 95 242 157 187 83 177 197 295 413 172 477 308 379 276 433
 313 409 312 491 167 224 356 345 372 249 350 303 428 408 297 196 434 59 374 83 325 331 44 365 138 45
1
```

```
Sorted array:0 1 1 2 2 4 4 4 6 6 6 7 8 8 9 11 11 13 14 17 20 20 21 23 23 24 25 26 26 28 29 30 30 30
 31 31 32 32 33 34 35 36 36 36 39 40 41 42 42 44 44 45 47 48 48 48 48 49 49 50 51 51 52 52 54 54 54 5
 4 55 57 57 59 59 60 61 61 63 64 64 67 68 71 77 78 79 79 79 80 82 83 83 83 89 91 93 94 95 97 100 100
 103 104 105 105 106 106 108 109 111 113 114 115 115 117 117 117 117 118 120 121 121 122 122 122 124
 125 126 126 130 131 131 132 133 133 134 135 136 136 137 138 139 141 141 144 144 145 146 147 151 152
 153 157 157 161 162 162 162 163 163 165 165 167 168 168 169 170 170 170 171 172 172 172 173 174 177
 178 181 184 187 189 190 191 193 194 194 195 196 197 197 200 200 201 202 204 204 206 209 210 211 211
 211 213 213 214 214 214 215 215 217 218 218 221 222 222 224 225 225 227 228 229 229 230 231 231 233
 233 234 236 237 241 242 243 243 245 245 246 247 247 249 249 250 252 253 254 255 260 261 261 264 265
 267 267 268 269 269 270 270 270 271 271 275 276 276 281 281 282 282 284 284 286 286 287 287 287 288
 289 289 291 295 296 297 297 300 300 301 302 302 303 305 306 307 307 307 307 308 308 310 312 313 313
 313 314 319 320 321 323 324 324 325 325 326 326 331 333 333 334 338 340 340 341 343 343 343 344 345
 345 347 348 349 350 350 354 355 355 356 356 356 357 358 358 359 359 362 362 364 364 364 365 366 366
 366 366 367 368 368 369 370 372 374 374 377 379 379 380 380 380 382 382 383 386 387 388 389 389 390
 390 390 390 391 391 393 394 397 398 399 401 402 404 407 408 409 409 409 410 412 412 412 413 413 413
 415 416 416 416 417 419 419 420 420 420 421 422 422 422 423 425 426 427 428 431 431 432 433 433 433
 433 434 434 435 436 438 438 439 441 444 444 444 444 445 445 446 447 447 451 452 453 453 456 457 457
 459 461 463 463 463 465 465 466 467 467 467 467 468 471 471 471 472 472 472 473 473 475 477 478 479
 479 480 480 481 481 483 483 484 485 486 486 486 489 489 490 490 491 492 493 493 494 496 496 498 499
```