Part1_Part2_report

Muhammed Yasir Fidan

May 2021

1 Missing Parts

There are only 2 thing missing for part1 and part2. First I didn't implement inverted, my table type only regular and I didn't print estimated working set function at the end of program run. Other things implemented succesfully for part 1 and part2.
I execute my program like this
./sortArrays 3 5 6 SC regular 10000 disk.dat
Because whenever disk file change for updating file I need to rewrite file so bigger numbers than those will take a lot of time. So you can test program with this inputs otherwise you will need to wait.

2 Structures

```
23
24  struct Page{
25      int* memory;
26  };
27
28  struct PageEntry{
29      int showingPageNum;
30      bool referanced = false;
31      bool modified = false;
32      //This time info used for LRU algorithm
33      double time;
34  };
35
36  struct Info{
37      int numberOfRead;
38      int numberOfWrite;
39      int numberOfPageMiss;
40      int numberOfPageRep;
41      int numberOfDiskPageWrite;
42      int numberOfDiskPageRead;
43  };
44
45  // Info for all threads
```

Figure 1: Page and Page Table structure

Those are the structures I use in my program. Page structure used for divide virtual and physical memory to pages and Page table structure hold some informations for finding physical memory from virtual memory and some other attributes that used in page replacements algoritihms.
When I first create memory beginning virtual memory pages show beginning physical pages by default.

3 Virtual and Physical memory

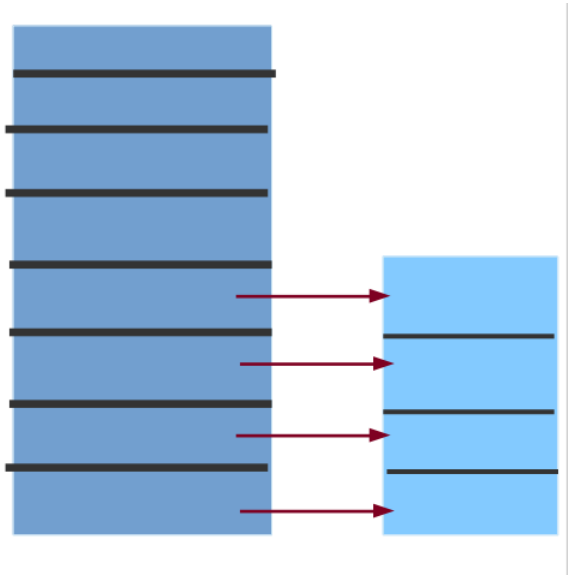

Figure 2: Page and Page Table structure

So initial virtual and physical memory look like this. After creation of this my main create 5 thread and start execution. Thread 0 is for bubble sort for sorting first half of virtual memory, thread 1 is for sorting second part of virtual memory by using quick sort. After waiting this 2 thread, thread 2 use merge function and merge this 2 sorted part and my virtual memory become complately sorted. Then thread 3 used for linear search and thread 4 used for binary search. This search methods search 5 number in virtual memory 3 of them in memory but 2 of them not. So virtual memory must at least bigger than 4 integer.

4 Threads

```
138
139     create_memory(argc,argv);
140
141     pthread_t threads[5];
142
143     pthread_create(&threads[0],NULL,&thread_bubble_sort,(void *)0);
144     pthread_create(&threads[1],NULL,&thread_quick_sort,(void *)1);
145
146     pthread_join(threads[0],NULL);
147     pthread_join(threads[1],NULL);
148
149     pthread_create(&threads[2],NULL,&thread_merge,NULL);
150     pthread_join(threads[2],NULL);
151
152     pthread_create(&threads[3],NULL,&thread_linear,NULL);
153     pthread_create(&threads[4],NULL,&thread_binary,NULL);
154
155     pthread_join(threads[3],NULL);
156     pthread_join(threads[4],NULL);
157
```

Figure 3: Threads create

Then I implement both get and set methods as mentioned in homework pdf. After create and initial filling with rand values to virtual memory, I always access physical memory by using this get and set methods. I implement get and set methods for all page replacement algorithm. So get and set methods check program executed for which page replacement algorithm and continue with its get and set methods.
In my page table structure there is a attribute called Showingpagenumber. If this is -1, it means this virtual memory page doesn't show any physical memory at this moment(Page miss). If this attribute different than -1, This page in the physical memory at this moment(hit).

5 Page Table

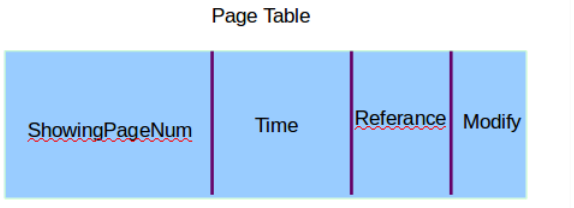

Figure 4: Page Table

This is my page table. There is 4 attribute in my page Table. First one of coruse each virtual memory show a physical memory page, so I hold this info in showPageNum attribute. Time attribute used in LRU and WSClock page replacement algorithms. Also Referance and Modify use in Page replacement algorithm and I every time a replace a page that is modified I need to write it back again disk file.

6 Statistics

I use a structure for holding statistics about each thread read,write,page miss,page replacement, number of disk read and number of disk write. I print each

```
36  struct Info{
37      int numberOfRead;
38      int numberOfWrite;
39      int numberOfPageMiss;
40      int numberOfPageRep;
41      int numberOfDiskPageWrite;
42      int numberOfDiskPageRead;
43  };
44
```

Figure 5: Information structure

5 threads statistics(this 6 attributes) at the end of program. Usually page miss,page replacement and disk read will be same because when 1 page miss occur, a page replacement will be done and of course also for taking a new page from disk a disk read will occur too.

7 Page Replacement Algorithms

I implement all 3 page replacements that told in pdf.(Second Change, LRU, WSClcok). I write get and set methods according to this algorithms. My general get and set methods simply chech page replacement algorithm and call its get and set methods.

7.1 Second Change

For Second change algorithm I create a vector. Of course if page in physical

```
63  //Used for second change
64  vector<PageEntry*>SecondChange;
```

Figure 6: Vector for Second Change page Replacement

memory we don't need to do anything, we can access data directly, but in second change algorithm if page is not in physical memory(in disk) we need replace a page. I hold my used pages in this vector for second change algorithm and check from beginning to end their referances bits from page table. If referance bit false I can remove this page from physical memory and add new page at the end of this vector, but if referance bit is true, that mean I use this page recently so just clear its referance bit erase it from beginning and add it again to the end of vector. And this algorithm goes like this until found a unreferance page.

```
530             if(SecondChange[i]->referanced == true){
531                 SecondChange[i]->referanced = false;
532                 PageEntry* temp = SecondChange[i];
533                 SecondChange.erase(SecondChange.begin());
534                 SecondChange.push_back(temp);
535                 i--;
536             }
```

Figure 7: Second change algorithm find replaced page

Of course after remove finding page in physical memory, I check this page modifyed or not. If its modified I will rewrite it to disk to make it up to date.

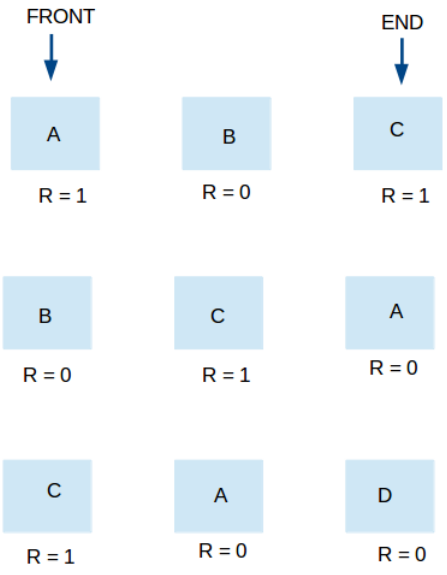

Figure 8: Second change algorithm vector

7.2 Least Recently Used(LRU)

Again if looking page in the physical memory it will be direct Hit, so there will be no page replacement or page miss, but if page is not in the physical memory LRU page replacement will work like that. FOr LRU I hold a time attribute in page table for each page. When a page replacement reguired, I traverse page table list and find the one that has smallest time. Smallest time means this page unused longest time, so I take it and replace it.

```
655         inc_PageMiss(threadType);
656         inc_PageRep(threadType);
657         //Find LRU
658         double minTime = std::numeric_limits<double>::max();
659         int index = 0;
660         for(int i=0; i<numVirtual; i++){
661             if(VirtualMemoryPT[i].showingPageNum != -1){
662                 if(VirtualMemoryPT[i].time < minTime){
663                     minTime = VirtualMemoryPT[i].time;
664                     index = i;
665                 }
666             }
667         }
```

Figure 9: LRU find min time

If changing page modified also I rewrite it to disk for make sure page is up to date in disk.

7.3 WSClock Page replacement

WSClock page replacement use both time and referance bits. I use a circular array structure for this algorithm and a threshhold value for time, initially it starts 0. I traverse this array circularly and chexk if index page R bit is 1 or 0. If R bit is 1, I make its R bit 0 and continue from the next page. If R bit is 0 I check its time amd compare it with threshhold value. If its time smaller then thresh hold time value I replace this page.

```
63  //Used for second change
64  vector<PageEntry*>SecondChange;
65  //Used for WSClock page replacement
66  double ThreshHold = 0; //ThreshHold for WSClock
67  int WSIndex = 0; //WSclock data structure index
68  PageEntry** Clock; //WSClock circular array
69
```

Figure 10: LRU find min time

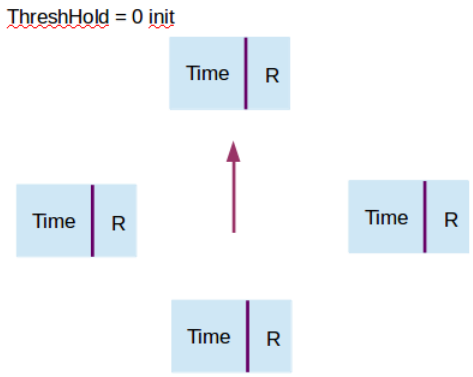

Figure 11: WSClock structure