EOPSY 21L LAB-4

Memory Management

MAY 18, 2021

KESHAV DANDEVA
Student ID: 302333

Introduction

Memory management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize overall system performance. Memory management keeps track of each memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

A computer can address more memory than the amount physically installed on the system. This extra memory is called **virtual memory** and it is a section of a hard disk that is set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Algorithms

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a **page fault** occurs, and a free page cannot be used for allocation purpose accounting to reason that pages are not available, or the number of free pages is lower than required pages.

First In First Out (FIFO) algorithm

Oldest page in main memory is the one which will be selected for replacement. Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Optimal Page algorithm

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists and has been called OPT or MIN. Replace the page that will not be used for the longest period. Use the time when a page is to be used.

Least Recently Used (LRU) algorithm

Page which has not been used for the longest time in main memory is the one which will be selected for replacement. Easy to implement, keep a list, replace pages by looking back into time.

Task

Create a command file for the provided simulator that maps any 8 pages of physical memory to the first 8 pages of virtual memory, and then reads from one virtual memory address on each of the 64 virtual pages. Step through the simulator one operation at a time and see if you can predict which virtual memory addresses cause page faults. What page replacement algorithm is being used? Locate in the sources and describe the page replacement algorithm in use.

MEMORY.CONF FILE

In this file, I defined the first 8 pages of physical memory to map with first 8 pages of virtual memory with the command 'memset'. The rest of the settings were kept as default except the 'addressradix'. The addressradix is set to 10 so that we can view the output 'tracefile' similar to the commands file i.e., in binary notation.

```
// memset  virt page #  physical page #  R (read from)  M (modified) inMemTime (ns) lastTouchTime (ns)
memset 0 0 0 0 0 0
memset 1 1 0 0 0 0
memset 2 2 0 0 0 0
memset 3 3 0 0 0 0
memset 4 4 0 0 0 0
memset 5 5 0 0 0 0
memset 6 6 0 0 0 0
memset 7 7 0 0 0 0


// enable_logging 'true' or 'false'
// When true specify a log_file or leave blank for stdout
enable_logging true

// log_file <FILENAME>
// Where <FILENAME> is the name of the file you want output
// to be print to.
log_file tracefile

// page size, defaults to 2^14 and cannot be greater than 2^26
// pagesize <single page size (base 10)> or <'power' num (base 2)>
pagesize 16384

// addressradix sets the radix in which numerical values are displayed
// 2 is the default value
// addressradix <radix>
addressradix 10

// numpages sets the number of pages (physical and virtual)
// 64 is the default value
// numpages must be at least 2 and no more than 64
// numpages <num>
numpages 64
```

SCRIPT.SH FILE

I created this script to create the commands file that will read from one virtual memory address on each of the 64 virtual pages. This is a simple bash script that will print READ 0, READ 16348 and so on in the commands file. We use multiples of 16348 as this is the default page size set in the memory.conf file.

```
#!/bin/bash

pagesize=0

for i in {0..63}
do
    printf "READ $pagesize\n" >> commands
    pagesize=$(expr $pagesize + 16384)
done
```

COMMANDS FILE

This file will command the simulator to read from one virtual memory address on each of the 64 virtual pages. Below are the screenshots of the first 10 lines and last 10 lines of the command file created.

```
1   READ 0
2   READ 16384
3   READ 32768
4   READ 49152
5   READ 65536
6   READ 81920
7   READ 98304
8   READ 114688
9   READ 131072
10  READ 147456
```

...

```
54  READ 868352
55  READ 884736
56  READ 901120
57  READ 917504
58  READ 933888
59  READ 950272
60  READ 966656
61  READ 983040
62  READ 999424
63  READ 1015808
64  READ 1032192
```

TRACEFILE

This is the output file containing the log of operations. It lists the executed commands and their output. It shows one line per operation executed. The format of each line is:

command address ... status

where:

command is READ or WRITE,
address is a number corresponding to a virtual memory address, and
status is okay or page fault.

Below are the screenshots of the first 10 lines, middle 10 lines and last 10 lines of the file.

```
READ 0 ... okay
READ 16384 ... okay
READ 32768 ... okay
READ 49152 ... okay
READ 65536 ... okay
READ 81920 ... okay
READ 98304 ... okay
READ 114688 ... okay
READ 131072 ... okay
READ 147456 ... okay
```

…

```
READ 442368 ... okay
READ 458752 ... okay
READ 475136 ... okay
READ 491520 ... okay
READ 507904 ... okay
READ 524288 ... page fault
READ 540672 ... page fault
READ 557056 ... page fault
READ 573440 ... page fault
READ 589824 ... page fault
READ 606208 ... page fault
```

…

```
READ 868352 ... page fault
READ 884736 ... page fault
READ 901120 ... page fault
READ 917504 ... page fault
READ 933888 ... page fault
READ 950272 ... page fault
READ 966656 ... page fault
READ 983040 ... page fault
READ 999424 ... page fault
READ 1015808 ... page fault
READ 1032192 ... page fault
```

Conclusion

As we can see from the output file, first 31 pages of the virtual memory were mapped correctly and the problem started from the 32nd page i.e., **524288.** In the memory.conf file, we specified to map first 8 pages only and in the result, we see not only those pages but pages till 31 were mapped correctly. The page fault appeared at 32nd page as it is being referenced although it is not mapped. We can see this in the simulator as well that it was mapped to -1 physical page (which doesn't exist). And because of that all the pages after that caused page fault.

So, according to the page replacement algorithm (defined in the beginning of the report), we observe that the first page fault i.e., 32nd page takes the first physical page i.e., page 0 as replacement which is first physical page in the queue. This leads to the conclusion that the algorithm in use here is the **FIFO (First In First Out)**. This information can also be found in the **PageFault.java** file.