Juan Luis de Reiset
UCID: 30050167

CPSC 457 - Assignment 5

Due date: **Thursday, April 11, 2019 at 11:30pm**.
Individual assignment. No group work allowed.
Weight: 8% of the final grade.

Q1 – Written question (5 marks)

free 100KB	P10 10KB	free 500KB	P11 20KB	free 200KB	P12 30KB	free 300KB	P13 40KB	free 600KB

The OS needs to allocate memory for 4 new processes in the following order: **P1 of 212KB, P2 of 417KB, P3 of 112KB and P4 of 426 KB**.

First Fit

free	**P10**	**P1**	**P3**	**free**	**P11**	**free**	**P12**	**free**	**P13**	**P2**	**free**
100 KB	10 KB	212 KB	112 KB	176 KB	20 KB	200 KB	30 KB	300 KB	40 KB	417 KB	183 KB

P1,P2 and P3 are allocated in memory, but then it is not possible to allocated P4, to a large enough partition.

Best fit

free	**P10**	**P2**	**free**	**P11**	**P3**	**free**	**P12**	**P1**	**free**	**P13**	**P4**	**free**
100 KB	10 KB	417 KB	83 KB	20 KB	112 KB	88 KB	30 KB	212 KB	88 KB	40 KB	426 KB	174 KB

P1, P2, P3 and P4 are allocated successfully in memory.

Worst Fit

free	**P10**	**P2**	**free**	**P11**	**free**	**P12**	**free**	**P13**	**P1**	**P3**	**free**
100 KB	10 KB	417 KB	83 KB	20 KB	200 KB	30 KB	300 KB	40 KB	212 KB	112 KB	276 KB

P1, P2 and P3 are allocated in memory, but then it is not possible to allocate P4 due to lack of a large enough partition.

Next fit

free	P10	**P1**	**free**	**P11**	**free**	**P12**	**free**	**P13**	**P2**	**P3**	**free**
100 KB	10 KB	212 KB	288 KB	20 KB	200 KB	30 KB	300 KB	40 KB	417 KB	112 KB	71 KB

P1, P2 and P3 are allocated in memory, but then it is not possible to allocate P4 due to lack of a large enough partition.

Q2 – Written question (5 marks)

Consider a virtual memory system with a page size of 512 bytes, and a page table shown below. Convert the following logical addresses to physical addresses. Show the page numbers and page offsets for each logical address.

Logical address	**Page number**	**Page offset**	**Physical address**
1027	2	3	3
2058	4	10	522
522	1	10	522
5	0	5	1541
2047	3	511	2559

	Page table
0:	3
1:	1
2:	0
3:	4
4:	1

Q3 – Written question (5 marks)

Consider a system with a 32-bit logical address space and 2KiB page size. The system supports up to 128MiB of physical memory. How many entries are there in each of the following?

a) A conventional single-level page table.

$$\text{Entries}=\frac{\textit{logical address space}}{\textit{page size}} = \frac{2^{32}}{2^{1} * 2^{10}} = \frac{2^{32}}{2^{11}} = 2^{21} = 2097152 \text{ entries}$$

b) An inverted page table.

$$\text{Entries}=\frac{\textit{physical address space}}{\textit{page size}} = \frac{128MB}{4KB} = \frac{2^{7} * 2^{20}}{2^{1} * 2^{10}} = \frac{2^{27}}{2^{11}} 2^{16} = 65536 \text{ entries}$$

Q4 – Written question (5 marks)

Consider a system where a direct memory reference takes 150ns.

a) <u>If we add a single-level page table stored in memory to this system, how much time would it take to locate and reference a page in memory?</u>

Given that the direct memory reference takes 150 ns, we can conclude that every memory access takes 150 ns. In the case of a single-level page, every instruction access requires at least two memory access (for page table lookup and fetching). Then the times it takes to locate and reference a page in memory is :

2 * 150ns = 300ns

b) <u>If we also add a TLB, and 80% of all page-table references are found in the TLB, what is the effective access time ? Assume that searching TLB takes 20ns.</u>

$$effective\ access\ time = (1-p)*(tlbs + 2*ma) + p * (tlbs + ma)$$

= (1-0.8) * (20ns + 2 * 150ns) + 0.8 * (20ns + 150ns)

= 200ns

Q5 – Written question (5 marks)

Consider the following page reference string: 1,2,1,4,2,1,5,2,4,7,5,4,1,4,7,1,4,2,1,7. Assume there are 3 available frame, all initially empty. Illustrate how pages are placed into the frames using LRU and Optimal page replacement algorithms. How many page faults would occur for each algorithm?

LRU:

1	2	1	4	2	1	5	2	4	7	5	4	1	4	7	1	4	2	1	7
1	1		1			1		4	4	4		4		4			4		7
	2		2			2		2	2	5		5		7			2		2
			4			5		5	7	7		1		1			1		1
Number of page faults:				**11**															

Optimal:

1	2	1	4	2	1	5	2	4	7	5	4	1	4	7	1	4	2	1	7
1	1		1			5			5			1					1		
	2		2			2			7			7					7		
			4			4			4			4					2		
Number of page faults:				**7**															

Q6 - Programming question (20 marks)

Write a program (`pagesim.c` or `pagesim.cpp`) that simulates three page replacement algorithms: Optimal, LRU and Clock. Your program will read in a reference string from standard input, and then run a simulation using all three algorithms. The number of available frames will be specified on the command line, and your simulation will start with all frames empty. For the clock algorithm you can use the single refererence bit implementation.

At the end of the simulation your program will output the content of the frames and the number of page faults for each placement algorithm. Your must format your output to match the sample output below:

Example input file test1.txt:	Sample output:

1 2 3 4 1 2 5 1 2 3 4 5	**$ ./pagesim 4 < test1.txt**
	Optimal:
	  - frames: 4 2 3 5
	  - page faults: 6
	LRU:
	  - frames: 5 2 4 3
	  - page faults: 8
	Clock:
	  - frames: 4 5 2 3
	  - page faults: 10

You can make the following assumptions:

- Number of available frames will be between 1 and 20 (inclusive).
- Number of entries in the reference string will be at most 5000.
- Frame numbers will be non-negative integers smaller than 100.

Q7 - Programming question (30 marks)

For this question you will implement a program (`fat.c` or `fat.cpp`) that will check the consistency of a file allocattion table with respect to the entries of a directory. Your program will read input from standard input, and will output results to standard output.

Input

The input will contain a simplistic representation of the filesystem. It will contain the following, all separated by white space:

- block size – an integer in range [1, 1024]
- number of entries in the directory – an integer in range [0, 50]
- number of entries in FAT – an integer in range [1, 200000]
- the entries in the directory – one entry per line, each containg:
    - filename – a string of up to 128 characters, any chars allowed except white space
    - first block pointer – an index into the FAT, an integer in range [-1, 200000), where '-1' denotes a NULL pointer
    - actual files size in bytes – an integer in range $[0, 2^{30}]$
- the entries in the FAT – a list of integers separated by white space
    - each entry represents a pointer to the next entry in the FAT
    - each entry is an integer in a range [-1, 200000)
    - -1 denotes a NULL pointer (end of chain)
    - the entries in FAT are numbered starting from 0

Sample input file `test1.txt`:

```
10 3 11
A.jpg 0 31
B.txt 6 23
C.zip -1 0
5 9 5 3 -1 1 8 0 6 -1 0
```

The above input describes a filesystem that has a block size of 10, contains FAT with 11 entries, and holds 3 files: `A.jpg`, `B.txt` and `C.zip`.

File `A.jpg` contains 31 bytes, and it is stored in blocks {0, 5, 1, 9}. It has the correct number of blocks, contains no cycles, and does not share blocks with any other file.

File `B.txt` has 23 bytes, and it is stored on blocks {6, 8}. The blocks belonging to file `B.txt` form a cycle, which is a problem that your program will need to detect. File `B.txt` also has an incorrect number of blocks for its size, which is another problem your program needs to report.

File `B.txt` does not share blocks with any other file. If it did, you would need to detect and report that as well.

File `C.zip` is empty. No blocks are allocated to this file. There are no problems with this file.

Finally, the total number of unused blocks on the filesystem is 5. This is a number you will need to calculate and report.

Output

After reading in the input, your program will check the consistency of the filesystem and report its findings to standard output. For every file you need to determine whether there are any issues with that file. You need to check for the following issues:

- Does the file contain the right number of blocks, or are there too many or too few?
- Do the blocks allocated to the file contain a cycle?
- Does the file share its blocks with any other file?

You then need to report all issues you found for every file.

You will also determine how many of the blocks are unused in the filesystem. Unused blocks are the ones not allocated to any file.

Sample output:

Here is an output that your program should produce for the above sample input:

```
$ ./fat < test1.txt
Issues with files:
      jpg:
      txt: not enough blocks, contains
   cycle C.zip:
Number of free blocks: 5
```

A skeleton code you can use as a starting point for this question is included in Appendix A.

Submission

You should submit 3 files for this assignment:

- Answers to the written questions combined into a single file, called either `report.txt` or `report.pdf`. Do not use any other file formats!
- Your solution to Q6 called `pagesim.c` or `pagesim.cpp`.
- Your solution to Q7 called `fat.c` or `fat.cpp.`

Since D2L will be configured to accept only a single file, you will need to submit an archive, eg. `assignment5.tgz`. To create such an archive, you could use a command similar to this:

```
$ tar zcvf assignment5.tgz report.pdf pagesim.cpp fat.cpp
```