47/54 +1 48

6/17/2014

CS570 Summer 2014 Exam #2

Nam[redacted]

Use

1. 4/4 2: 4/4 3: 4/4 4: 4/4 5: 6/6 6: 8/9 7: 4/4 8: 4/6 9: 9/9 B4 = 50

1. Consider a virtual memory system using paged memory with 32-bit physical addresses and 32-bit virtual addresses and 18 bits of the virtual address was for the offset.
   a. What size should the pages be (how big do they need to be)? 255 KB
page #	offset
14 bits	18 bits

   The offset tells you how big it should be, $2^{18}-1 = 262143$ bytes ≈ 255.99 KB
   b. How many pages can the process have (maximum number)? 16384 pages.
   page # = 14 bits, represented so $2^{14} - 1 = 16,383$ unique pages 0 ~ 16,383

2. Given an O/S which has divided all of fast memory into 3 frames, and if this O/S uses the LRU algorithm for page replacement/swapping, then how many page faults are generated if the following sequence of pages are requested for use (show your work on back!): 7

   2 5 2 | 3 5 4 | 2 5 1 | 2 3 2

3. Repeat #2, but use the FIFO algorithm for page replacement/swapping instead. How many page faults? 9

   2 5 2 | 3 5 4 | 2 5 1 | 2 3 2

4. The Optimal Page Replacement algorithm is the only one that can't be implemented in a run-time system. Since it can't be implemented in a run-time system, why do we study it and use this algorithm?

   We use this algorithm as a base case to base all our other page replacement algorithms such as FIFO, LRU, and Clock to see how well they match the ideal case which is OPT.

5. Some operating systems provide a system call, "rename()", to give a file a new name. Is there any difference between using this call to rename a file and just copying the file to a new file with the new name, followed by deleting the old one (include an explanation of why/how)

   There is a difference between using the rename() system call versus copying a file to a new file and renaming it. The rename system call does not change the modification time stamp, and it doesn't create a new file, and it doesn't create another inode link to that data. The copy does the reverse of that, and it can fail if disk is full.

2) LRU algorithm Main Memory

2 5 2 3 5 4 2 5 X 2 3 2

cache

# of Iterations →	1	2	3	4	5	6	7	8	9	10	11	12
	2	5	2	3	5	4	2	5	1	2	3	2
	X	2	5	2	3	5	4	2	5	1	2	3
	X	X	X	5	2	3	5	4	2	5	1	1
	F	F		F		F	F		F		F	

F = page fault

Total Pf = 7 ✓

3) FiFo Algorithm

2 5 2 3 5 4 2 5 X X 3 2

# of iterations →	1	2	3	4	5	6	7	8	9	10	11	12
	2	5	5	3	3	4	2	5	1	1	3	2
	X	2	2	5	5	3	4	2	5	5	1	3
	X	X	X	2	2	5	3	4	2	2	5	1
	F	F		F		F	F	F	F		F	F

Pf total = 9 ✓

(2) 45,054

page # = $\frac{45,054}{4096}$ offset = 45,054 mod 4096

Page # = 10.99 offset = 4094 ✓

= 10 9 logical units.

(3) page # = $\frac{131,072}{4096} = 32$

offset = 131,072 mod 4096 31

= 0 ✓ logical units.

6. Your process needs to access the following three virtual addresses (given in base 10), compute the virtual page number and the offset within that page for each address if page size = 4 KB: 20,092, 45,054, 131,072
(1) (2) (3)

4KB = 4 · 1024 = 4096

(1) 20,092

Page # = $\frac{20,092}{4096} = 4.9 = 4$ 3 −1 offset = 20,092 mod 4096 = 3708 ✓ logical units.

Teacher Leonard Guy wrong +1

7. Each entry in a typical page table has several control bits to enable the h/w and operating system to manage the pages with. Name the correct control bits for each below:

P in memory or not M modify bit?

1. This page table entry is valid: valid bit
2. The page at this entry has been written to: ~~accessed bit~~
3. The page at this entry may be read fm/written to: ~~read and write only~~
4. The page at this entry has been read fm/written to: ~~modify bit~~

8. List three advantages of using page-segmented memory over segmented memory. Be specific and explain why it's an advantage:

less fragmentation - no external fragmentation, but there is internal fragmentation ✓

less searching - since there is a base size, no length needs to be kept track of. ✓

less complex protection algorithm. - ~~self explanatory~~ paging is more organized so protection is easier.

−2

9. A UNIX filesystem has 512 byte blocks and 4 byte disk addresses. What is the maximum filesize assuming inodes have 10 direct, one single, one double, and one triple indirect addresses in each inode?

$4 \overline{)512}$ $\frac{128}{}$ ✓ $= 2^7$

From the direct 10 · 2^9 = 5 · 2 · 2^9

0) = 5 kB

10 direct

1) 1 single

2) 1 double

3) 1 triple

1) From the one single $2^7 \cdot 2^9 = 2^{16} = 2^6$ KB ✓

2) From the one double $2^7 \cdot 2^7 \cdot 2^9 = 2^{23} = 2^3 \cdot 2^{20} = 2^3$ MB ✓

3) From the one triple $2^7 \cdot 2^7 \cdot 2^7 \cdot 2^9 = 2^{30} = 16$B ✓

The maximum file size is easily calculated by summing parts 0 through 3:

max file size = 5KB + 2^6KB + 2^3MB + 16B

= 5(1024) + 2^6(1024) + $2^3(1024)^2$ + 1$(1024)^3$

= 1,082,201,088 bytes ✓

which is approximately ≈ 16B ✓ ↑ divide by (1024^3)

7. Present Bit, Modified Bit, Protection Bits, Referenced Bits

Explain how "ls ~" works? directory inode etc...

1) The shell is the running process.
2) The shell issues a fork system call
3) A nearly identical process is created, and the child process is created to run the new born process ls.
4) ls then has to list the directory, the home directory as indicated by the ~ tilde
5) The ~ tilde translates to an absolute address /usr/home
6) The directory root is obtained from the file system probably ext-3. As indicated by /
7) The inode is obtained for /usr
8) The directory /usr is obtained
9) The inode for /usr/home is obtained
10) The directory for /usr/home is obtained
11) We know, from lecture that file names are kept and maintained by the directories.
12) ls simply reads this as a data stream and after some internal formatting as coded in ls it displays the output to the stdout by default.