83/100

COSC 450 Operating System Midterm #2

5/2/2024

Name: Kyle Tranfaglia.

1. (5 pt.) A system use the paging for managing virtual memory. The system has four page frames. The time of loading, time of last access, and the reference bit **R**, and modified bit **M** for each page are as shown below (the times are in clock ticks):

Page	Loaded	Last referenced	R	M
0	245	255	0	0
1	220	265	0	1
2	119	270	1	0
3	115	280	1	1

a) Which page will FIFO (First In First Out) replace? use lowest loaded

Page 3 ✓

b) Which page will NRU (Not Recently Used) replace? use R, M classes

Page 0 ✓

c) Which page will LRU (Least Recently Used) replace? use lowest referenced

Page 0 ✓

d) Which page will Second chance replace? use fifo, R=1 means second chance

Page 3 - saved, set R=0

Page 2 - saved, set R=0

Page 1, R=0, replace → Page 1 ✓

2. (10 pt.) Let's assume that a LINUX system use bitmap for maintain free disk block information. Let assume the bitmap was completely lost due to the crash. Is it possible to recover bitmap? If possible, discuss your algorithm to recover the bitmap in detail. If not, discuss why.

Yes, it is possible. Size of block & number of blocks are stored in super block in partition. We can use this to scan all the i-nodes. ✓

Algorithm:

New Bitmap (size = # of blocks from super block)
Reset (set to 0) ✓
for each I-Node do ✓
for each I-Node Entry do
Reset to 1 (depending on block info) ✓

3. (10 pt.) Suppose that a machine generates 38-bit virtual addresses and 32-bit physical addresses.

   a. What is the main advantage of a multilevel page table over a single-level one?

   multilevel page tables can store much more virtual addresses than physical adress spaces. The layering increases virtual address storage without drastically increasing the size of a single page table which can improve searching & less wasted space

   ✓

   b. With a two-level page table, 16-KB pages, and 4-byte entries, how many bits should be allocated for the top-level page table field and how many for the next level page table field? Explain your answer.

   -4 ✗

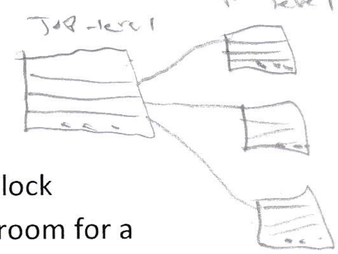
This is because top level is one page table mapping to 2k-1 page tables as each entry in top-level maps to a table in second level

2^{38} bytes / $2^4 \cdot 2^{10}$ → $2^{38} / 2^{14} = 2^{24}$ bytes / 4byte

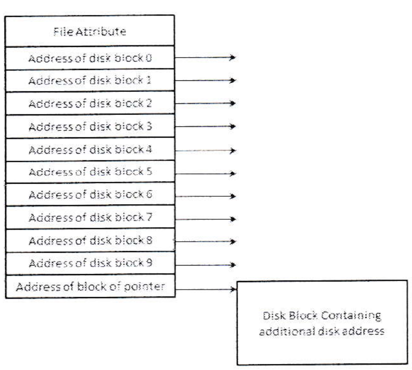
$= 2^{20}$ page tables … 2^{20} bytes $\cdot\ 2^3 = 2^{23}$ bits

1 for top page table

$2^{23} - 1$ for next level page table



4. (5 pt.) LINUX like system use i-node to maintain the file system. Attributes and block addresses are saved in i-node. One problem with i-nodes is that if each one has room for a fixed number of disk addresses, what happens when a file grows beyond this limit? One solution is to reserve the last disk address not for a data block, but instead for the address of block containing more disk-block addresses as shown following picture.
Picture shows that i-node contains 10 direct addresses and these were 64 bits each. A block size is 4 KB. If a file use i-node and one extra block to save block information, what world the largest possible file size could be?



4KB / 64 bit → 4KB / (64 / 2^3)

→ $2^2 \cdot 2^{10}$ / 8 → $2^{12} / 2^3$ → 2^9 bytes

2^9 = 512 + 10 = 522 bytes ✓

4KB * 522 → $2^{12} \cdot 522$ = 2138112 bytes ✓

or

$2^2 \cdot 2^{10} \cdot 522$

2088 KB

5. (10 pt.) Consider the following resource allocation snapshot with five processes (P_0 P_1, P_2, P_3, P_4) and five resources (A, B, C, D). Existing matrix for resources is E = (3, 14, 12, 12)

	Allocated				Maximum Need			
	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2
P_1	1	0	0	0	1	7	5	0
P_2	1	3	5	4	2	3	5	6
P_3	0	6	3	2	0	6	5	2
P_4	0	0	1	4	0	6	5	6

a) What are the values of matrix request matrix R and available matrix A.

$C = \begin{pmatrix} 0012 \\ 1000 \\ 1354 \\ 0632 \\ 0014 \end{pmatrix}$ $E = (3, 14, 12, 12)$ $R = \begin{pmatrix} 0000 \\ 0750 \\ 1002 \\ 0020 \\ 0642 \end{pmatrix}$ $A = (1, 5, 2, 0)$

$A = E - C \rightarrow (3,14,12,12) \xrightarrow{P_0} (3,14,11,10) \xrightarrow{P_4} (2,14,11,10) \xrightarrow{P_2} (1,11,6,6) \xrightarrow{P_3} (1,5,3,4) \xrightarrow{P_4} (1,5,2,0)$

b) Show that it is not deadlock

No deadlock!

$A = (1, 5, 2, 0)$

$C = \begin{pmatrix} 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 1 & 3 & 5 & 4 \\ 0 & 6 & 3 & 2 \\ 0 & 0 & 1 & 4 \end{pmatrix}$ $R = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{pmatrix}$

$(1,5,2,0) \xrightarrow[P_0]{+(0,0,0,2)} (1,5,2,2) \xrightarrow[P_2]{+(1,3,5,4)} (2,8,7,6) \xrightarrow[P_3]{+(0,6,3,2)} (2,14,11,8) \xrightarrow[P_4]{+(0,0,1,4)} (2,14,12,12) \xrightarrow[P_1]{+(1,0,0,0)} (3,14,12,12) = E$

c) If a resource request from P_1 arrives for (0, 4, 2, 0), Can the request be granted immediately? You need show safe or unsafe state.

$C = \begin{pmatrix} 0002 \\ 1420 \\ 1354 \\ 0632 \\ 0014 \end{pmatrix}$ add from R

$R = \begin{pmatrix} 0000 \\ 0310 \\ 1002 \\ 0020 \\ 0642 \end{pmatrix}$ subtract request

$A = (1,5,2,0) - (0,4,2,0) \rightarrow (1,1,0,0)$

$(1,1,0,0) \xrightarrow[P_0]{+(0,0,0,2)} (1,1,0,2) \xrightarrow[P_2]{+(1,3,5,4)} (2,4,5,6) \xrightarrow[P_3]{+(0,6,3,2)} (2,10,8,8)$

$\xrightarrow[P_1]{+(1,4,2,0)} (3,14,10,8) \xrightarrow[P_4]{+(0,0,1,4)} (3,14,12,12) = E$

Safe!

6. (5 pt.) Discuss each of followings.
   a. What are four necessary conditions for a deadlock
      1) mutual exclusion
      2) circular wait
      3) Hold & Wait
      4) No preemption
   b. Four strategies for dealing with a deadlock
      1) Ignore
      2) Detection & Recovery
      3) Avoidance with dynamic Allocation
      4) Attack one of the four necessary conditions for deadlock

7. (5 pt.) About Log-Structured File System
   a. Log-Structured File system can be apply based on the assumption. What is this assumption? Files are cached to RAM once opened
   b. Linux use i-node for saving blocks information for a file. To open a file, operating system checks the directory for the file to get i-node number. Since i-nodes are located in special location, operating system does not need search for i-node. In LSF (Log-Structured File) system, i-node is not located in specific location. Briefly discuss how LSF operating system could access a file. I-Nodes do not have fixed location I-Nodes are written to log. LSF uses structure called I-node map to get current I-Node location. So, LSF would use I-node map to get current I-Node to access a file.

8. (10 pt.) A computer system generates a 64-bit virtual address for a process. This system has 32GB RAM and page size is 16 KB.

   a. If each entry in the page table needs 64 bits per entry, calculate the possible size of the page table by bytes.

   2^{64} bytes / 16KB $\Rightarrow 2^{64} / 2^4 \cdot 2^{10} \rightarrow 2^{64}/2^{14}$
   $= 2^{50} \cdot 64 bit \Rightarrow 2^{50} \cdot 2^6 = 2^{56} bit / 2^3 = \boxed{2^{53} \text{ bytes}}$

   b. Page frame number information for each page must be saved in the page table. **How many bits** does it need to save page frame number information?

   32 GB / 16 KB $\rightarrow 2^5 \cdot 2^{30} / 2^4 \cdot 2^{10} = 2^{35} / 2^{14} = 2^{21}$

   $\boxed{21 \text{ bits}}$

9. (5 pt.) The deadlock detection algorithm needs three matrixes for deadlock detection.
   - Available resource matrix (A).
   - Current allocation matrix (C)
   - Request matrix (R)

   Consider the following state of a system with four processes, P_1, P_2, P_3, and P_4, and five types of resources R_1, R_2, R_3, R_4, and R_5 with matrixes

$$C = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 \end{bmatrix},\ R = \begin{bmatrix} 1 & 1 & 0 & 2 & 1 \\ 0 & 1 & 0 & 2 & 1 \\ 0 & 2 & 0 & 3 & 1 \\ 0 & 2 & 1 & 1 & 0 \end{bmatrix},\ A = [0\ 1\ 0\ 2\ 1]$$

By using the deadlock detection algorithm, show that where there is a deadlock or not in the system.

$A = (0,1,0,2,1)$

$+ (0,1,0,1,0)$ $\quad + (0\ 0\ 0\ 0\ 1)$ Deadlocks!

$(0,1,0,2,1) \xrightarrow{P_1} (0,2,0,3,1) \xrightarrow{P_2} (0,2,0,3,2) \rightarrow$

10. (5 pt.) A system has five processes P_1, P_2, P_3, P_4, P_5, and three types of resources R_1, R_2, R_3. The current allocation and needs more per each process are as follows:

	Allocated	Need More	Available
	R_1 R_2 R_3	R_1 R_2 R_3	R_1 R_2 R_3
P_1	0 1 0	7 4 3	3 3 2
P_2	2 0 0	1 2 2	
P_3	3 0 2	6 0 0	
P_4	2 1 1	0 1 1	
P_5	0 0 2	4 3 1	

Suppose P_5 request 3 more resource R_1 and 2 more resource R_2. Does this request lead to an unsafe state?

-4

P_5 has allocated $(0,0,2)$ and requests $(3,2,0)$

	Max Need R_1 R_2 R_3	Request
P_1	7 5 3	7 3 0
P_2	3 2 2	1 2 2
P_3	9 0 2	3 0 0
P_4	2 2 2	0 1 1
P_5	4 3 3	4 3 1

$A = (3,3,2) - (3,2,0) = (0,1,2) \xrightarrow[P_4]{+(2,1,1)} (2,2,2)$

$\xrightarrow[P_2]{+(2,0,0)} (4,2,2) \xrightarrow[P_3]{+(3,0,2)} (7,2,2)$ can't fulfill P_5 or P_1!. Not safe!

11. (10 pt.) In the file system of an operating system, two widely utilized methods exist for managing free blocks: a linked list and a bitmap. Consider a scenario where the block size is 8-KB and the file system employs 32-bit disk block numbers.

a. How many maximum blocks are required to keep track of a 128-GB disk using a linked list?

$2^3 \cdot 8KB / 32bit \Rightarrow 2^6 \cdot 2^{10} / 2^5 bit \Rightarrow 2^{16} / 2^5 = 2^{11} - 1 = 2048 - 1$

$128GB / 8KB = 2^7 \cdot 2^{30} / 2^3 \cdot 2^{10} \Rightarrow 2^{37} / 2^{13} = 2^{24}$ blocks

$= 2047$ block info

$2^{24} / 2047 = \boxed{8196 \text{ blocks}}$

b. How many blocks are required to keep track of a 128-GB disk using a bitmap?

from 2^{24} bitblocks $/ 2^3 \Rightarrow 2^{21}$ bytes $/ 2^3 \cdot 2^{10} = 2^{21} / 2^{13} = \boxed{2^8 \text{ blocks}}$

c. What is the maximum disk size supported by this operating system?

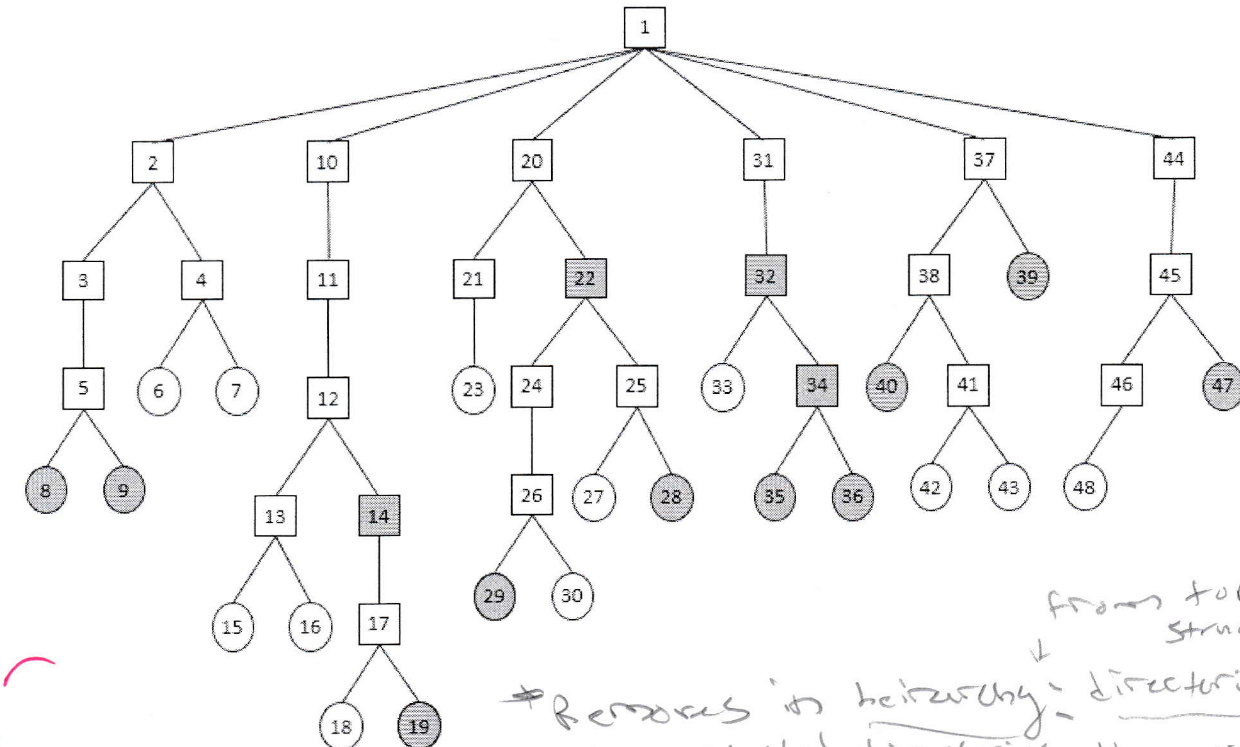
2^{32} bytes $/ 8KB = 2^{32} / 2^3 \cdot 2^{10} \Rightarrow 2^{32} / 2^{13} = \boxed{2^{14} \text{ bytes}}$

$\downarrow$

$\boxed{512 KB}$

-3

12. (10 pt.) The following figure shows a file tree with directories (squares) and files (circles). The shaded items have been modified since the base date and thus need to be dumped. The unshaded ones have not been modified since the base date. The dump algorithm maintains a bitmap indexed by i-node number. Bits per i-node. Bits will be set and cleared in this map as the algorithm proceeds. We discussed logical dump algorithm which has four phases. Show bitmap after each phase by shading.



from top of structure ↓

*Removes in heirarchy: directories containing, then shaded directories, then files tend

-5

Phase 1) X

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Shade all shaded file, directories, and directories containing shaded

Phase 2)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Remove directories with shaded files ^

Phase 3)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Remove directories shaded X

Phase 4)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Remove shaded files X

13. (10 pt.) One way to use contiguous allocation of the disk and not suffer from holes is to compact the disk every time a file is removed. Since all files are contiguous, copying a file requires a seek and rotational delay to read the file, followed by the transfer at full speed. Writing the file back requires the same work. Assuming a seek time of 6 msec, a rotation delay of 5 msec, a transfer rate of 16 MB/sec, and an average file size of 8 KB, how long does it take to read a file into main memory and then write it back to disk at a new location? Using these numbers, how long would it take to compact half of a 32 GB disk. (1 KB = 2^{10} B, 1 GB = 2^{30} B, 1 sec = 1000 msec, 1 hour = 3600 sec)

Formula: seek time + rotation delay + (Avg file size / transfer rate)

$\Rightarrow 6+5 = 11$ msec

11 msec + (8KB / (16MB/sec))

needs to be ms

$\Rightarrow$ 11 msec + (8KB / 16MB/sec) $\cdot 10^3$

$= 11 + \left(\frac{2^{13}}{2^{24}}\right) \cdot 10^3$

$= 11 + \left(\frac{10^3}{2^{11}}\right) = 0.4883 + 11$

$= 11.4883$

Read + write $= 2 \cdot 11.4883 = 22.9766$ msec

half of 32GB $\rightarrow$ 16GB   16GB / 8KB $\rightarrow 2^4 \cdot 2^{30} / 2^3 \cdot 2^{10}$
$= 2^{34} / 2^{13} = 2^{21}$

$\Rightarrow 2^{21} \cdot 22.9766$

$= 4.82 \times 10^7$ msec / 10^3 msec/sec $= 48185.4$ sec

48185.4 sec / 3600 sec/hr $= \boxed{13.3848 \text{ hours}}$