CS342 Operating Systems - Spring 2023
Homework #2 and Solution

Assigned: May 28, 2023.
Due date: June 5, 2023, 23:59. Document version: 1.2

Q1. Suppose we have 4 resource types, A, B, C, D, and 6 processes, P0...P5, in a system. The current Allocation and Maximum Demand matrices are shown below. We have, in total, 15 instances of A, 6 instances of B, 9 instances of C, 10 instances of D. Prove that the current state is safe. If, at this state, a request from P5 (3,2,3,3) is made, should it be granted or not? Prove your answer.

	Alloc				MaxDemand			
	A	B	C	D	A	B	C	D
P0	2	0	2	1	9	5	5	5
P1	0	1	1	1	2	2	3	3
P2	4	1	0	2	7	5	4	4
P3	1	0	0	1	3	3	3	2
P4	1	1	0	0	5	2	2	1
P5	1	0	1	1	4	4	4	4

Answer:
Existing = 15, 6, 9, 10
Available is: 6 3 5 4
Need is:
7 5 3 4
2 1 2 2
3 4 4 2
2 3 3 1
4 1 2 1
3 4 3 3
With current available pool of resources, P1's need can be satisfied. Pool becomes: 6 4 6 5.
With that available pool, P2's need can be satisfied. Pool becomes: 10 5 6 7.
With that available pool, the needs of all other processes can be satisfied. Hence this state is safe.

If P5 makes a request :3 2 3 3. New state will be:
Avail: 3 1 2 1
Need:
7 5 3 4
2 1 2 2
3 4 4 2
2 3 3 1
4 1 2 1
0 2 0 0

Alloc is:
2 0 2 1
0 1 1 1
4 1 0 2
1 0 0 1
1 1 0 0
4 2 4 4

In this state:
We can not satisfy any of the needs with the current available pool of resources. Therefore, the state is unsafe. Hence the request of P5 should not be granted.

Q2. Assume we have a system that is using single-level paging. Assume page table for a process is always in the memory.
a) If a physical memory access takes 150 ns, what is the effective access time to memory (EAT) without TLB?

Answwer: We need 2 physical memory accesses. Hence it is 2 x 150 ns = 300 ns.

b) Assume we have a TLB used. The TLB search takes 10 ns, no matter it is a hit or miss. If hit rate is 85%, what is the effective access time to memory?

Answer: With TLB, if there is a miss, we need 10 + 150 + 150 = 310 ns.
If there is a hit, we need 10 + 150 = 160 ns.
EAT = 0.85x160+0.15x310= 182.5 ns.

c) If two level paging would be used, what would be your answer for questions a) and b)?

If we have two-level page table used, we need a) 150x3 = 450 ns without TLB; b) With TLB, EAT = 0.85(10+150)+0.15 (10 +450) = 205 ns.

Q3. Assume a system is using four-level paging, 48-bit virtual addresses and 48-bit physical addresses. A virtual address is divided into five pieces as follows: [`9, 9, 9, 9, 12`]. That means, the first 9 bits are index to the first-level table. Offset is 12 bits. Assume each page table entry (any level) is 8 bytes.

a) How many bits in a page table entry are used to store a frame number?

Answer: 48-12 = 36 bits.

b) How much memory is consumed by 1st, 2nd, 3rd, and 4th level page tables for a process that has 6 MB of virtual memory used, starting at address `0x000000300000`. That means the process has one virtual memory area used, starting at virtual address `0x000000300000` (included), and ending at virtual address `0x0000000900000` (not included). In other words, the range of legal addresses is [`0x000000300000`, `0x0000000900000`).

Answer: A leaf level table can map 2^9 x 2^12 = 2 MB of contiguous virtual memory. The indicated vma start at 3 MB and goes up to 9 MB. Therefore we need 1 leaf (4th level table) for the following virtual address ranges:
[3 MB, 4 MB).
[4 MB, 6 MB)
[6 MB, 8 MB)
[8 MB, 9 MB)
Hence we need 4 4th level tables. We also need 1 first level, 1 second level, and 1 third level table. In total we need 7 tables, each 4 KB long. We need 28 KB space. 28 KB is consumed by page table.

c) How much memory is consumed by 1st, 2nd, 3rd, and 4th level page tables for a process that has a code segment of 128 KB at virtual address `0x000001000000`, a data segment of 2 GB starting at virtual address `0x000800000000` and a stack segment of 4 MB starting at virtual address `0x0f0000000000`. Assume for this question that stack segment also grows upward (towards higher addresses).

Answer:
A 4th level can map: 2^21 bytes. 2 MB.
A 3rd level can map: 2^30 bytes. 1 GB.
A 2nd level can map: 2^39 bytes. 512 GB.
We need 1 1st level table.
--- For code segment, we need 1 2nd level, 1 3rd level, and 1 4th level table. Code segment starts at 2^24 = 16 MB.
--- Data segment starts at 2^35 = 32 GB. Therefore we need the same 2nd level page table, but different (different than the 3rd level table used for code segment) 3rd level tables. 2 3rd level tables will be needed (since data segment is 2 GB and). We need 2^31/2^21 = 2^10 = 1024 4th level tables.
--- Stack segment starts at address 15x2^40. Therefore for stack we need a different 2nd level table. We also need different other level tables. We need 1 3rd level table and 2 4th level tables.

In total we need:
1st level: 1 table

2nd level: 1 + 1 = 2 tables
3rd level: 1 + 2 + 1 = 4 tables
4 level: 1 + 2^10 + 2 = 1024 + 3 = 1027 tables.
In total: 1027 + 4 + 2 + 1 = 1034 tables.
It makes 1034 x 4 KB = **4136 KB**. ~= **4 MB**. This is the total memory consumed by the page tables (1st, 2nd, 3rd, and 4th level) of the process.

Q4. Consider the following page reference string of a process.

2 4 1 3 4 6 3 6 1 2 1 5 2 5 4 1 2 4 3

Assume the process has 3 frames that it can use, all empty initially.
a) Assume second chance (i.e., clock) algorithm is used as page replacement algorithm. Assume reference bits (R bits) are cleared after every 5 references (i.e., some time between every 5th and 6th reference - after the 5th reference and before the 6th reference). Show the memory state (the pages in memory and their R bit values) after each page reference. Also indicate which reference causes a page fault. Assume after a page fault, when the new page is loaded, its R bit is set to 1.

Answer:
In second change we maintain a FIFO list of page numbers. The R bits of the pages are considered as well while making page replacement decisions.
f: page fault.

2: 2(1) f
4: 2(1) 4(1) f
1: 2(1) 4(1) 1(1) f
3: 4(0) 1(0) 3(1) f
4: 4(1) 1(0) 3(1)

4(0) 1(0) 3(0) // reset the R bits

6: 1(0) 3(0) 6(1) f
3: 1(0) 3(1) 6(1)
6: 1(0) 3(1) 6(1)
1: 1(1) 3(1) 6(1)
2: 3(0) 6(0) 2(1) f

3(0) 6(0) 2(0) // reset

1: 6(0) 2(0) 1(1) f
5: 2(0) 1(1) 5(1) f
2: 2(1) 1(1) 5(1)
5: 2(1) 1(1) 5(1)
4: 1(0) 5(0) 4(1) f

1(0) 5(0) 4(0) // reset

1: 1(1) 5(0) 4(0)
2: 4(0) 1(0) 2(1) f
4: 4(1) 1(0) 2(1)
3: 2(1) 4(0) 3(1) f
total number of page faults: 11.

b) Solve the same question for Optimal algorithm.

	2	4	1	3	4	6	3	6	1	2	1	5	2	5	4	1	2	4	3
	f	f	f	f		f				f		f			f				f
F1	2	2	2	3	3	3	3	3	3	3	3	5	5	5	4	4	4	4	3
F2		4	4	4	4	6	6	6	6	2	2	2	2	2	2	2	2	2	2
F3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

total number of page faults: 9.

Q5. Suppose that a disk drive has 10,000 cylinders, numbered 0 to 9,999. The drive is currently serving a request at cylinder (track) 4500, and the previous request was at cylinder 3900. The queue of pending requests, in FIFO order, is as follows:

5200 2000 9500 4300 1500 5100 9600 4000 4600

Starting from the current head position, what is the total distance (in cylinders/tracks) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms: FCFS, SCAN , SSTF.

Answer:
a) FCFS order used.
4500 5200 2000 9500 4300 1500 5100 9600 4000 is the order.
total distance = 700+3200+7500+5200+2800+3600+4500+5600+600 = 33700.
b) SCAN order is:
4500 4600 5100 5200 9500 9600 9999 4300 4000 2000 1500
total distance = 13998.
c) SSTF order is:
4500 4600 4300 4000 5100 5200 2000 1500 9500 9600
total distance = 100+300+300+1100+100+3200+500+8000+100 = 13700.

Q6. A disk has the following parameters:

Size: 512 GB
RPM: 3600
avg seek time: 4 ms
max transfer rate: 30 MB/s.

a) Assume block size 4 KB. What is the I/O time to read one random block from this disk? How many such transfers can we complete per second? What is the I/O data rate (i.e., throughput).

Answer:
4KB/50MB/s = 0.13 ms. This is the transfer time for 4 KB. Number of rotations per second is 3600 / 60 = 60. avg rotational latency is: 1000 x (1/60) / 2 = 8.3 ms. Then total time to transfer 4 KB is: 4 ms (seek time) + 8.3ms + 0.13ms = 12.43ms. Number of I/Os per second = 80. Throughput: 80 x 4 = 320 KB/sec = **0.3125 MB/s.**

b) Assume we will read 512 blocks (each 4 KB), that are contiguous on the disk, sequentially. How many such transfers can we complete per second? What is the I/O data rate (i.e., throughput).

Answer:
512x4KB / 30x1024 KB = 66.6 ms. Time for I/O = 4 + 8.3 + 66.6 ms = 78.9 ms. Number of I/Os per second = 12.6. I/O rate: 12.6 x 512x4K = **25.2 MB/s.**

Q7. Consider a file system that uses inodes to represent files. Disk blocks are 4 KB in size, and a pointer to a disk block requires 8 bytes. Assume in an inode we have 10 direct disk block pointers, one single indirect pointer, one double indirect pointer, and one triple indirect pointer. That means, combined index allocation scheme is used to keep track of the blocks allocated to a file.

a) What is the maximum size of a file that can be stored in this file system?

In an inode, we can store: 4 KB / 8 = 512 pointers. Total file size is: 10+2^9+2^18^2 + 2^27 ~= 2^27 blocks. That makes 2^27 x 2^12 = 2^39 Bytes = 512 GB. So, max file size is around 512 GB.

b) How many second level index blocks are required for a file X of size 8 GB.

A leaf index block can map 2^9 x 2^12 = 2^21 = 2 MB of file data. For 8 GB, we need 2^33 / 2^21 = 2^12 = 4096 leaf index blocks. We also have 10 direct pointers in the inode, which maps 10x4KB = 40KB; but this will not affect the answer (40 KB < 2 MB).

-- 1-level structure has one leaf index block (which is not second-level).
-- 2-level structure has 2^9 = 512 leaf index blocks (all second-level).
-- 3-level structure has many leaf (3rd level) index blocks. We need to use 4096-1-512 of them. This will require 7 second-level index blocks from 3-level structure to be used ceiling((4096-1-512)/512)= 7.

In total we need: 512 + 7 = **519** second level index blocks. 512 of them will be from the 2-level structure. 7 of them will be from the 3-level structure.

c) If nothing, except the inodes, is cached in memory, how many disk block accesses are required to access a byte i) at offset 2^15, ii) at offset 2^23, iii) at offset 2^32 of the file X?

i) offset 2^15. 2^15 / BS = 2^15/2^12 = 8. It is in the file block 8. Hence the respective block is pointed directly by the inode. Therefore we need to do one disk access (1 disk block accessed) to retrieve the data block containing the byte at that offset.

ii) offset = 2^23. This is 8 MB. A single index node can map 2 MB of contiguous file data. The first ~2 MB of the file is mapped by the single-level index structure. Hence the address of the block containing the offset is in the two-level index structure. Therefore we need to access the 1st level index block of the two-level index structure and then a 2nd level index block, and then the data block. We need 3 disk accesses. That means we need to access 3 blocks (these blocks do not have to contiguous). Two of them are index blocks. The third one is the data block that contains the desired byte.

iii) offset = 2^32 = 4 GB. A two-level index structure can map 2^30 Bytes = 1 GB. The first ~1 GB of the file is mapped by the two-level index structure. Since we want to access a byte at 4 GB, the address of the block containing that byte is in the 3-level index structure. Hence we need to access the 1st level index block of the 3-level structure, then a 2nd level index block, then a 3rd level index block and then the data block. In total 4 disk accesses are required. That means 4 different blocks (can be at any place on the disk) need to be retrieved. Three of them are index blocks, and one of them is a data block.