Ömer Oktay Gültekin

2190413 Section 1

HW 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

Need

A	B	C	D
7	5	3	4
2	1	2	2
3	4	4	2
2	3	3	1
4	1	2	1
3	4	3	3

Existing

A	B	C	D
15	6	9	10

Available

A	B	C	D
6	3	5	4

we need to find a sequence of allocating Needs to prove that the state is safe.

→ Try to find a row in Need; that is ≤ Available.

P_4. run completion. Available becomes = [6 3 5 4] + [1100] = [7 4 5 4]

P_3. run completion. Available becomes = [7 4 5 4] + [1001] = [8 4 5 5]

P_2. run completion. Available becomes = [8 4 5 5] + [4102] = [12 5 5 7]

P_1. run completion. Available becomes = [12 5 5 7] + [0 1 1 1] = [12 6 6 8]

P_0. run completion. Available becomes = [12 6 6 8] + [2 0 2 1] = [14 6 8 9]

P_5. run completion. Available becomes = [14 6 8 9] + [1 0 1 1] = [15 6 9 10]

A sequence $P_4, P_3, P_2, P_1, P_0, P_5$ is found. The state is SAFE.

- P 5 request resources [3 2 3 3]

⇒ [3 2 3 3] ≤ [6 3 5 4] ⇒ enough resources available.

⇒ Checking if the new state is safe if we allocate request:

Allocation

	A	B	C	D
P1	2	0	2	1
P2	0	1	1	1
P3	4	1	0	2
P4	1	0	0	1
P5	1	1	0	0
P6	4	2	4	4

Need

A	B	C	D
7	5	3	4
2	1	2	2
3	4	4	2
2	3	3	1
4	1	2	1
0	2	0	0

Available

A	B	C	D
3	1	2	1

- since no row satisfies: Need; ≤ Available in the safe state, the new state is NOT SAFE.
- Request of P_5 should NOT be allocated.

Q2. Assume we have 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?
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?
c) If two level paging would be used, what would be your answer for questions a) and b)?

a) We need 2 physical memory accesses. Therefore, it is $2 \times 150\ ns = \boxed{300\ ns}$. //

b) 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
$EAT = 0.85 \times 160 + 0.15 \times 310\ ns = \boxed{182.5\ ns}$

c) If two level paging would be used,
a) $150 \times 3\ ns = \boxed{450\ ns}$ (2 accesses for mapping, 1 access for data) is needed without TLB.
b) with TLB, $EAT = 0.85 \times 160\ ns + 0.15 \times (10+450)\ ns = \boxed{205\ ns}$

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?

A physical address is 48-bits. $48-12 = 36$ bits are used to store a frame number $\Rightarrow$ PFN = 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).

start address : 0x 000000300000 = 0... | 0... | 0...1 | 10... (offset | 9 bit | 9 bit | 9 bit | 9 bit)

End address : 0x 000000900000 = 0... | 0... | 0...100 | 10.... (offset | 9 bit | 9 bit | 9 bit | 9 bit)

Level 1 = 1 Table, Level 2 = 1 Table(0), Level 3 = 1 Table (0),
Level 4 = 4 (001, 010, 011, 100) $\Rightarrow$ Total table no = 1+1+1+4 = 7
Every table has 2^9 entry, each 8 byte.
$\Rightarrow$ $\boxed{\text{Memory consumed} = 7 \times 2^9 \times 2^3 = 28\ KB}$

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).

Code segment:

Start Address : 0x000001000000 (offset)
size : + 0x20000
End Address: 0x000001020000

Pieces: 0 0 8 0 / 0 0 8 32

Data segment:

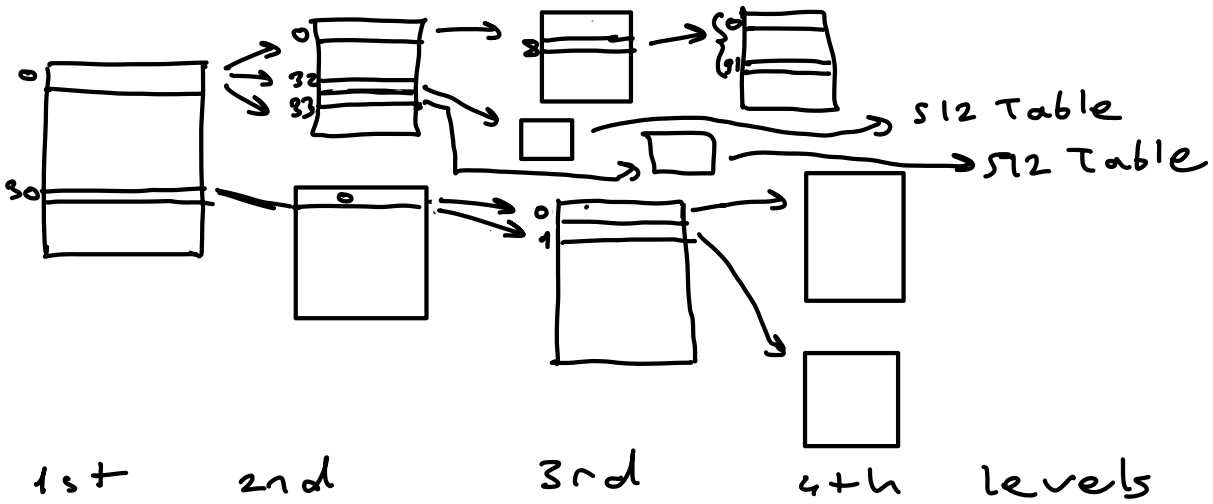
Start Address : 0x000800000000 (offset)
size : + 0x80000000
End Address: 0x000880000000

Pieces: 0 32 0 0 / 0 34 0 0 (not included)

Stack segment:

Start Address : 0x0f0000000000 (offset)
size : + 0x400000
End Address: 0x0f0000400000

Pieces: 30 0 0 0 / 30 0 2 0



1st 2nd 3rd 4th levels

Total Memory consumed = Table size x num of tables used

$= 512 \times 8\,B \times (\underset{1st}{1} + \underset{2nd}{2} + \underset{3rd}{4} + \underset{4rd}{1027})$

$= 2^{12} \times 1034\,B = 4136 \times 2^{10} = 4136KB$

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). 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.

b) Solve the same question for Optimal algorithm.

a) victim pointer (hand) points to the page shown with •.

2: 2•(1) fault

4: 2(1) 4•(1) fault

1: 2•(1) 4(1) 1(1) fault

3: 3(1) 4•(0) 1(0) fault

4: 3(1) 4•(1) 1(0)

6: 3(0) 6(1) 1•(0) fault

3: 3(1) 6(1) 1•(0)

6: 3(1) 6(1) 1•(0)

1: 3(1) 6(1) 1•(1)

2: 1•(0) 6(0) 2(1) fault

1: 1(1) 6•(0) 2(0) fault

5: 1(1) 5(1) 2•(0) fault

2: 1(1) 5(1) 2•(1)

5: 1(1) 5(1) 2•(1)

4: 1•(0) 5(0) 4(1) fault

1: 1•(1) 5(0) 4(0)

2: 2(1) 5•(0) 4(0) fault

4: 2(1) 5•(0) 4(1)

3: 2(1) 3(1) 4•(1) fault

b) 2: 2 _ _ Fault

4: 2 4 _ Fault

1: 2 4 1 Fault

3: 3 4 1 Fault

4: 3 4 1

6: 3 6 1 Fault

3: 3 6 1

6: 3 6 1

1: 3 6 1

2: 3 2 1 Fault

1: 3 2 1

5: 5 2 1 Fault

2: 5 2 1

5: 5 2 1

4: 4 2 1 Fault

1: 4 2 1

2: 4 2 1

4: 4 2 1

3: 4 2 3 Fault

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.

FCFS: The tracks are visited in the order shown below:

$|4500-5200| = 700$

$|5200-2000| = 3200$

$|2000-9500| = 7500$

$|9500-4300| = 5200$

$|4300-1500| = 2800$

$|1500-5100| = 3600$

$|5100-9600| = 4500$

$|9600-4000| = 5600$

$|4000-4600| = 600$

Total : **33700**

SCAN: The track numbers in sorted order: 1500 2000 4000 4300 [4500 >>>] 4600 5100 5200 9500 9600

① The head will move from track 4500 to the end of the platter and the requests on the way.

$4500 \rightarrow 9999 = 5499$

② Head move back up to 1500

$9999 - 1500 = 8499$

Total head movement: $8499 + 5499 = \boxed{13.998}$

SSTF: $4500 \rightarrow 4600 \rightarrow 4300 \rightarrow 4000 \rightarrow 5100 \rightarrow 5200 \rightarrow$

$100 + 300 + 300 + 1100 + 100 + 3200$

$2000 \rightarrow 1500 \rightarrow 9500 \rightarrow 9600$

$+ 500 + 8000 + 100 \Rightarrow$ Total = **13.700**

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).

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).

a) • Transfer time of a block $= \frac{4\ KB}{30\ MB/s} = \frac{2}{15}\ ms$

• Average rotational latency $= \frac{1}{2} \cdot \frac{1}{\frac{RPM}{60}} = \frac{1}{2} \cdot \frac{60}{3600} = \frac{1}{120}\ s = \frac{25}{3}\ ms$

• Total time to transfer 4 KB $= T_{seek} + T_{rotation} + T_{transfer}$

$= 4 + \frac{25}{3} + \frac{2}{15}\ ms = \frac{125+60+2}{15} = \frac{187}{15}\ ms$

• The number of disk I/Os per second $= \frac{1}{T} = \frac{15}{187} \cdot 1000$ (to convert to second)

≈ 80.

• Throughput $= \frac{80\ I/O}{sec} \cdot \frac{4\ KB}{I/O} = 0.32\ MB/s$

b) If we read 512 blocks, $A = 4\ KB \times 512 = 2048\ KB$

$= 2\ MB$ total data.

• Total required time to read this data $= \frac{2\ MB}{\frac{30\ MB}{s}} \approx 0.067\ sec.$

$= 67\ ms$

• Time for one I/O $= 4 + \frac{25}{3}(8.33) + 67\ ms = 79.33\ ms = T$.

• Number of disk I/Os per second $= \frac{1}{T} = \frac{1}{0.07933} = 12.6$

• Throughput $= \frac{12.6\ I/O}{sec} \times \frac{2\ MB}{I/O} = 25.2\ MB/s$

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

a) In an inode we can store $4\,KB / 8 = 512$ pointers.
Total file size is: $10 + 512 + 512^2 + 512^3 \cong 2^{27}$ blocks.
That makes $2^{27} \times 2^{12} = 2^{39}$ bytes = **512 GB** = max file size supported.

b) A leaf index block can map $2^9 \times 2^{12}$ bytes $= 2 \cdot 2^{20} = 2\,MB$.
For 8 GB, we need $2^{33} / 2^{21} = 2^{12} = 4096$ leaf index blocks. For the single level we can have at most 1 leaf index block. For the double level (using double indirect pointer), 512 index block we can have.
For the triple level 512^2 index block we can have.
$\Rightarrow 4096 - 513 = 3583$ 3rd level leaf required
$\Rightarrow$ we need 7 more second level inode block for 3583 leaf block $\Rightarrow$ **512 + 7 = 519** second level index block required.

c) 1) offset 2^{15} is in which block? $\frac{2^{15}}{2^{12} = 8} = 8$. It is in the file block 8. Hence the respective block is pointed directly by the inode. Therefore we need to do **1 disk access** to retrieve the data block containing the byte at that offset.

2) $2^{23} = 8$ MB. A single index block can map 2 MB data. Therefore, 2-level index structure is needed. We will access, first, second index block and then the data block. We need 3 disk accesses.

3) $2^{32} = 4$ GB. $4\text{ GB} / 2\text{ MB} = \frac{2\text{ GB}}{\text{MB}} = 2.2^{10} = 2048$

3 level structure is needed $\Rightarrow$ 4 disk access.