# CS341 - EndSem Assignment

Name : M. Maheeth Reddy
Roll : 1801CS31

Date : 03-May-2021

## Answers

Ans 1: The given system can seperate the program into code and data. To share the programs among different users, two read-only base-limit register pairs are provided: one for instructions and one for data.

The advantages of the above scheme are:

1. Code is protected from being modified incorrectly, because the registers are read-only.
2. Code and data can be shared among different users effectively. They can be shared easily among different threads as well.

The disadvantages of the above scheme are:

1. The seperation of code and data might consume time. This is not necessary because compilers, after compiling the program, make sure code and data are seperated.
2. CPUs for such scheme would contain more registers, and hence would be costly to create.

Ans 2: Users can share code and data by allowing two entries in a page table to point to the same frame in memory.

Consider the following situation,

there are two users that share a large amount of data. If two entries in the page table point to that same data in physical memory, then we need not create a copy of that data in the physical memory for the second user. Both users can access the same data with the above scheme.

Updating a byte on one page implies that the byte in physical memory is also modified. Hence, this change will take effect on the other page as well. This could cause a problem if the shared data was supposed to be seperate

Ans 3 Demand paging is a strategy to load pages only when they are demanded during program execution. So, pages that are never accessed, will not be loaded into physical memory at all.

The hardware to support demand paging is same as that for paging and swapping:

① Page Table: This table has the ability to mark an entry invalid through a valid-invalid bit or a special value of protection bits

② Secondary Memory: This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk or NVM device. It is known as the swap device, and the section of storage used for this purpose is called swap space.

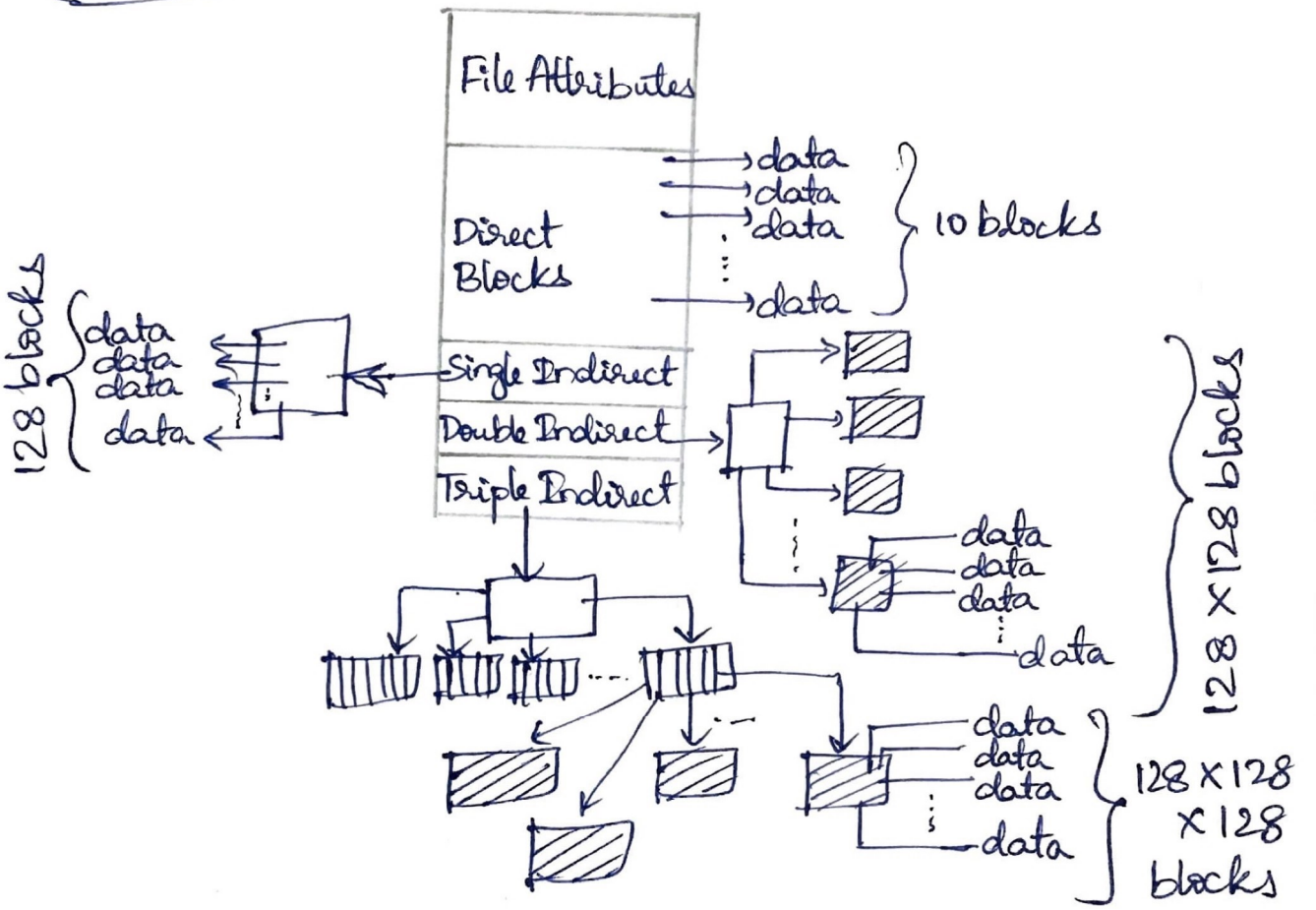
③ Translation Lookaside Buffer (TLB): This will improve the performance of the lookup operation. It acts as a cache.

Ans 4: Given, size of one data block = 1024 bytes

no. of addresses in direct block = 10

no. of addresses in each data block = 128

Structure of a file:




Max size of a file in this filesystem

= {Sum of sizes of all data blocks corresponding to this file

= {Sum of size of data blocks in direct, single indirect, double indirect and triple indirect blocks.

= [10 + 128 + 128 X 128 + 128 X 128 X 128] X 1024 bytes

= 2113674 X 1024 bytes

∴ Maximum size of a file in this file system

$= 2113674 \times 1024$ bytes $= 2164402176$ bytes

$= \frac{2164402176}{2^{30}}$ GB $\approx 2.0157$ GB

Ans 5 FIFO-based segment-replacement algorithm:

FIFO stands for First-In First-Out. So, in this case we are performing a combination of FIFO principle along with finding segments that are large enough to accomodate incoming segment.

Algorithm:

(1) In the list of segments, search for the first segment that is sufficiently large enough for accomodating the incoming segment and replace it.

(2) If there is no such segment, then,

i) If relocation is possible, perform memory rearrangement in such a way that the segments in the beginning that are collectively large enough to accomodate the incoming segment are contiguous in memory. After this, add any leftover space to the free space list.

ii) But if relocation is not possible, select a combination of segments closest to the beginning of the list such that their memories are contiguous and can accomodate the new segment. If there is any leftover space, add it to the free space list.

## LRU based segment-replacement algorithm:

LRU stands for Least Recently Used. So we will try to replace the segments that have not been used for the longest period of time.

### Algorithm:

(1) Search for a segment which is large enough to accomodate the incoming segment as has not been used for the longest period of time. Replace it and add any leftover space to the free space list.

(2) If there are no such segments, then,

i) If relocation is possible, perform memory rearrangement such that the oldest segments are contiguous in memory and replace these with the new segment.

ii) Otherwise, just select a combination of the oldest contiguous segments such that they are large enough to accomodate the incoming segment.

Ans 6: (a) CPU utilization = 13% and Disk Utilization = 97%.

Since, the Disk utilization is way more than CPU utilization, the system will spend most of the time to service page faults and ~~only~~ negligible actual processing is done. This is called Thrashing.

To allow more CPU utilization, some of the processes must be suspended. This means the degree of multi-programming is reduced.

Paging isn't helping in this case because processes are spending most of the time for paging.

(b) CPU Utilization = 87% and Disk Utilization = 3%

CPU utilization is way higher than disk utilization. This implies CPU is already busy and will cause a bottleneck eventually. So, we need not change the degree of multiprogramming.

If the degree of multiprogramming is increased now, and as the disk is not being used much, thrashing can happen. Even, the CPU utilization will decrease which is not desired. result of increasing degree of multiprogramming.

But paging is helping in this scenario because high CPU utilization (87%) means CPU is busy.

© CPU utilization is 13% and Disk Utilization = 3%

CPU utilization is low, implying that CPU is not working at full capacity. So increasing the degree of multiprogramming is sensible choice to increase CPU utilization.

The impact of paging isn't significant as there are lesser no. of processes

Ans 7: Total tape drives = 13

The requirements and allocations of tape drives for each process have been summarized in the table below:

| Process | Maximum Requirement of Tape Drives | Allocated Tape Drives | Tape Drives Needed for Execution |
|---|---|---|---|
| P1 | 11 | 6 | 11-6 = 5 |
| P2 | 5 | 3 | 5-3 = 2 |
| P3 | 8 | 2 | 8-2 = 6 |

So, the total no. of allocated tape drives = 6+3+2 = 11

∴ No. of available tape drives = 13-11 = 2

With the 2 available tape drives, only process P2 can be executed.

So, the 3 tape drives allocated to P2 become free. Now there are 2+3 = 5 available tape drives.

Only P1 can be executed with the 5 available tape drives.

Now, totally 5+6 = 11 tape drives are available with which P3 can be executed. Hence, $P_2$, $P_1$, $P_3$ is the safe sequence

Ans 8

Given,

no. of surfaces on disk pack = 16

no. of tracks per surface = 128

no. of sectors per track = 256

Amount of data stored in a sector = 512 bytes (in bit serial manner)

Total no. of sectors = no. of surfaces × no. of tracks per surface × no. of sectors per track

$= 16 \times 128 \times 256 = 2^4 \times 2^7 \times 2^8$

$\Rightarrow$ Total no. of sectors $= 2^{19}$

So, we need 19 bits to specify a particular sector

Capacity of the disk pack = Total no. of sectors × Data stored per sector

$= 2^{19} \times 512 \text{ bytes} = \frac{2^{19} \times 512}{2^{20}} \text{ MB}$

$= 256 \text{ MB}$

$\therefore$ Capacity of the disk pack is 256 MB and No. of bits required to specify a sector is 19

**Ans 9:** Given,

Data stored in a track = 16384 bytes

Rotation Time = 16ms

Average Seek Time = 40ms.

We need to calculate access time for 1024 bytes

$$\text{Access Time} = \left(\begin{matrix}\text{Seek}\\ \text{Time}\end{matrix}\right) + \left(\begin{matrix}\text{Average Rotational}\\ \text{Delay}\end{matrix}\right) + \left(\begin{matrix}\text{Transfer}\\ \text{Time}\end{matrix}\right)$$

Transfer Time for 1024 bytes

$$= \frac{\text{sectors read}}{\text{sectors per revolution}} \times \text{rotational time}$$

$$= \frac{1024}{16384} \times 16\text{ms}$$

[Ratio of Sectors read to sectors per revolution is same as that of corresponding data]

$$= 1\text{ ms}$$

$$\text{Average Rotational Time} = \frac{\text{Rotation Time}}{2} = \frac{16}{2}\text{ms} = 8\text{ms}$$

$\Rightarrow$ Access Time = 40ms + 8ms + 1ms = 49ms.

$\therefore$ The time to read a block of 1024 bytes is 49 ms

Ans 10 Given, page fault service time = 10ms

average memory access time = 20 ns

One page fault is generated every $10^6$ memory accesses,

$\Rightarrow$ Miss Ratio = $\frac{1}{10^6}$ $\Rightarrow$ Hit Ratio = $1 - \frac{1}{10^6}$

Effective Memory Access Time = (hit ratio × memory access time) + (miss ratio × page fault service time)

$= \left(1 - \frac{1}{10^6}\right) \times 20 + \frac{1}{10^6} \times 10 \times 10^6$ ns

$= 20 - 20 \times 10^{-6} + 10$ ns $= 30 - (2 \times 10^{-5})$ ns

$\Rightarrow$ Effective Memory access time is 29.99998 ns which is approximately 30 ns

Ans 11

Given, capacity = 3 page frames

initially, none of the pages are available in memory

Reference String = 1, 2, 1, 3, 7, 4, 5, 6, 3, 1

Optimal Replacement Policy is being used

Optimal Replacement Policy: Replace the page which is not used in the longest dimension of time in future

Reference String

| Frames | | 1 | 2 | 1 | 3 | 7 | 4 | 5 | 6 | 3 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F1 | | (1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| F2 | | | (2) | 2 | 2 | (7) | (4) | (5) | (6) | 6 | 6 |
| F3 | | | | | (3) | 3 | 3 | 3 | 3 | 3 | 3 |
| | | (*)1 | (*)2 | | (*)3 | (*)4 | (*)5 | (*)6 | (*)7 | | |

No pages available in memory

(*) → denotes page fault

7 page faults

In page fault (*)4, 2 was replaced by 7 in frame 2 because it is not used in future.

The reason is same for replacing 7 by 4, 4 by 5, 5 by 6 in page faults (*)5, (*)6, (*)7 respectively.

∴ There are a total of 7 page faults

Ans 12 Given, physical address has 36-bit
virtual address has 32-bit
page frame size is 4KB
page table entry size is 4 bytes

Address Translation involves three-level page table,

bits 30-31 → index into level I page table (2 bits)
21-29 → II (9 bits)
12-20 → III (9 bits)
0-11 → offset within the page (12 bits)

Now, max no. of page frames = $\frac{\text{Size of Physical address space}}{\text{Page Size}}$

$= \frac{2^{36}}{4 \times 2^{10}} = \underline{\underline{2^{24}}}$

So, 24-bits are required to index page numbers in 3rd level page table.

Now, to access second level page table entry, 9 bits are used

∴ Size of second level page table = total no. of entries × size of 1 entry
$= 2^9 \times 4$ bytes $= 2^{11}$ bytes

⇒ Size of second level page table is $2^{11}$ bytes

Hence, in a $2^{36}$ physical address space, we can have $\frac{2^{36}}{2^{11}} = 2^{25}$ second level page tables.

∴ 25-bits are required to index second level page tables.

Similarly, first level page table entry requires 9 bits for access

∴ Size of first level page table = $2^9 \times 4$ bytes = $2^{11}$ bytes.

⇒ There will be $2^{25}$ first level page tables in a $2^{36}$ physical address space. Hence, we need 25-bits for indexing.

∴ We need 25 bits to address next level page table in page table entry of first level page table

25 bits to address next level page table in page table entry of second level page table

24 bits to address next level page table in page table entry of third level page table.

Ans 13 Given,

$$\text{average time for memory access} = \begin{cases} M \text{ units, for page hit} \\ D \text{ units, for page fault} \end{cases}$$

experimental average time for memory access = X units

Let us assume page fault rate is P

We know,

$$\text{experimental memory access time} = P \times (\text{memory access time for page fault}) + (1-P) \times (\text{memory access time for page hit})$$

$$\Rightarrow X = PXD + (1-P) \times M$$

$$\Rightarrow X = pD + M - pM = M + p(D-M)$$

$$\Rightarrow X - M = p(D-M)$$

$$\Rightarrow \boxed{P = \frac{X-M}{D-M}}$$

$$\therefore \boxed{\text{Page fault rate is } \frac{X-M}{D-M}}$$

Ans 14 Given,

TLB access time = 10 ns

Main memory access time = 50 ns

TLB hit ratio = 90% = 0.9

⇒ TLB miss ratio = 1 − 0.9 = 0.1

No page faults

Effective memory access time

= Hit Ratio × (TLB access time + memory access time)

+ Miss Ratio × (TLB access time + page table access time + memory access time)

= 0.9 × (10 + 50) + 0.1 × (10 + 50 + 50) ns

(Assuming, page table lookup takes only one memory access ⇒ page table access time = 50 ns)

= 0.9 × 60 + 0.1 × 110 = 54 + 11 = 65 ns

∴ Effective Memory Access Time = 65 ns

Ans 15 Given,

No. of frames = m

Length of page reference string = p

No. of distinct page numbers = n

(a) The least no. of page faults will happen when the no. of frames, is greater than the number of distinct page numbers i.e., $m > n$. In this scenario, page faults will occur only when distinct page no.'s are retrieved. Hence, the no. of page faults will be n.

So, lower bound for page faults is n and happens when $m > n$.

(b) Max no. of page faults will happen when there is only one frame ($m = 1$), and none of the retrieved page no.'s in the reference string are same as the page in the frame. In such scenario, page faults will occur till the end of reference string. So, upper bound for page faults is p and happens when $m = 1$

NOTE: For both (a) & (b) cases, $p > m, n$