CS-341
End-Semester Assignment

Name: P.V. Sriram
Roll: 1801CS37

1) The given system can separate 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, one for data.

Advantages

1) Data sharing is easier between threads which need same data.
   For eg. Only one copy of an editor or a compiler needs to be kept in the memory, and this code can be shared by all processes needing access to the editor or compiler code.
2) Protection of code against erroneous modification.

Disadvantages

1) The only disadvantage is that code and data must be separated, which is usually adhered to in a compiler-generated code.
2) It is more expensive to create such a CPU because it would contain more registers.

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 separate. This is similar to copy-on-write in Linux.

Hence, advantage is to avoid deep and expensive copies as long as it is possible. Changing a byte would invoke copying entire frame.

3) Demand Paging

Demand paging is like simple paging and swapping all rolled up into one. We consider the page to be the unit of I/O. Instead of swapping all of the pages at once, when a context-switch occurs, we defer loading or storing any page until it becomes absolutely necessary. If we attempt to access a virtual address that lies within a page in memory, we win. If not, this is a page fault. When a page fault occurs, we load the page into main memory from the backing store. If there isn't enough available main memory to load the page, a page in memory is first written to the backing store.

At a minimum, demand paging requires hardware to support the logical to physical address translation. Required hardware includes a translation look-aside buffer to hold page table entries. In addition, each page must have an associated valid bit. While it is not required, additional bits such as a dirty bit and a reference bit can simplify the implementation of the page replacement algorithm.

In addition, if the architecture supports DMA I/O, a lock bit is required.

Finally, the machine must implement retractable instructions, which will complicate CPU design.

4) Given,

In a UNIX system,

Block size = 1024

Each block contains addresses of 128 blocks.

Required, maximum file size.

Maximum file size of system = Summation of size of all the data blocks whose address belong to the file

For 10 direct address of data blocks,

Size = 10 × 1024 = 10240 bytes

For 1 singly indirect data block = 128 × 1024 = 131072 bytes

For 1 doubly indirect data block = 128 × 128 × 1024 = 16777216 bytes

For 1 triply indirect data block = 128 × 128 × 128 × 1024 = 2147483648 bytes

⇒ Max file size = 10240 + 131072 + 16777216 + 2147483648

= 2103674 × 1024 Bytes

= 2.01 GB

5) FIFO

FIFO stands for First In - First Out. Therefore, in this scenario we need to accomodate the incoming segment as soon as a suitable slot is available

(i) Linearly iterate through the list of segments. Search for the first segment which has enough space to accomodate the incoming segment and replace it.

(ii) If there is no such segment then,

a) 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.

b) 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

LRU stands for last recently used. Naturally, we would look for such a segment which wasn't used in a long time as well as fits the incoming segment, replace such a segment.

(i) First, linearly iterate through the segment list, and identify such a segment which was not used the longest as well as can fit the incoming segment. Replace it.

(ii) If such a segment is not found,

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

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

6) The degree of multiprogramming is the maximum number of process that a single processor system can accomodate efficiently.

Thrashing

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.

9) • CPU utilisation 13%, Disk utilisation 97%

Disk utilization at 97% is too high while the CPU at 13% is too low this causes thrashing. Processes that are being executed spend much time on the disk in attempts to be paged. The degree of multiprogramming cannot be increased because process need to be suspended to increase CPU utilization.

b) CPU utilization 87%, Disk utilization 3%. CPU at 87% and disk at 3% means that the CPU is going to cause the bottleneck eventually. Disk is being underused and increasing the degree of multiprogramming would inevitably cause thrashing.

c) CPU utilization 13%, Disk utilization 3%. CPU and disk space are both very low. The obvious answer would be an increase the degree of multiprogramming would increase CPU utilization.

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

7)

	Allocated	Max	Need
P1	6	11	5
P2	3	5	2
P3	2	8	6

Maximum need for the process P1, P2, P3 is 11, 5, 8 and current allocation is 6, 3, 2 respectively. Total 13 tape drive were present out of which 11 are allocated and left with 2 drivers.

Now P1 needs 5, P2 needs 2, P3 needs 6. Only P2's need can be fulfilled. Now after P2 execute 5 tapes are free which can fulfill P1's demand after that P1 will execute.

∴ Execution order → **P2, P1, P3**

8) Given,

Surfaces = 16

Tracks = 128/surface

Sectors = 256 / Track

Data = 512 B / Sector

∴ Total capacity = $16 \times 128 \times 256 \times 512$

$= 2^4 \times 2^7 \times 2^8 \times 2^9 = 2^{4+7+8+9} = 2^{28}$ bytes

∴ 256 MB

No. of Sectors = $16 \times 128 \times 256 = 2^4 \times 2^7 \times 2^8$

$= 2^{19}$ sectors

∴ 19 bits are required to uniquely identify sectors

9) Given,

Disk with Track size = 16384 bytes

Rotation time per track = 16 ms

⇒ Transfer rate = 16834/16 = 1024 bytes/ms

Also given,

Block size = 1024 bytes

Average seek time = 40 ms

Average latency = $\frac{1}{2}$ × (Total Transversal time for Track) = $\frac{1}{2}$ × 16 = 8 ms

Total time = Average latency + Average seek time + Transfer time

= 40 + 8 + $\frac{1024}{1024}$ = 40 + 8 + 1 = 49 msec

10) Given,

Page fault service time = 10 ms

Memory Access time = 20 ns

Page fault rate = $1/10^6 = 10^{-6}$

Effective access time = (Page fault rate) x (Page fault service time)

+

(Hit rate) x (Mem Access time)

$\Rightarrow \quad 10^{-6} \times 10^{6} \times 10 + (1 - 10^{-6}) \times (20)$

$\approx \quad 10 + 20 \approx \boxed{30 \text{ ns}}$

11) Given,

Frame size = 3

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

Also given,

Optimal page replacement policy

	1	2	1	3	7	4	5	6	3	1
x	x	x	x	3	3	3	3	3	3	3
x	x	2	2	2	7	4	5	6	6	6
x	1	1	1	1	1	1	1	1	1	1
	X	X	✓	X	X	X	X	X	✓	✓

No. of page faults = ~~8~~ 7

12) Given,

Virtual address size = 32 bit

Physical address size = 36 bit ⇒ Physical memory size = 2^{36} bytes

~~Page~~ Page frame size = 4 KB = 2^{12} bytes

Also given,

offset for page = 12 bits ⇒ page size = 2^{12} bytes

∴ No. of page tables for processes = $2^{36}/2^{12} = 2^{24}$ bytes

∴ We need 24 bits for third level.

Similarly,

Given offset for 3rd level entry = 9 bits = 2^9 entries

∴ No. of entries in 2nd level table = 2^{36}/table size

Page table size = (No. of entries) × (Entry size)

= 2^9 × 4bytes = 2^{11}

↑

(Virtual memory = 32 bits)

∴ No. of entries in 2nd level table = $2^{36}/2^{11}$ = 2^{25} bytes

∴ We need 25 bits for second level

Similarly,

No. of entries for 1st level = $2^{36}/2^9 \times 4 = 2^{25}$

∴ We need 25 bits for First level

∴ 25, 25, 24

13) Given,

An OS uses demand paging.

Also given,

Average Memory access = M units (Hit)

= D units (Miss)

Given effective time taken for memory access is X

assume 'K' is the page fault rate.

Average Memory access = (1 - page fault rate) × Mem Access when no page fault

+

(page fault rate) × Mem Access at page fault

$\Rightarrow X = (1-K)M + KD$

$X = M - KM + KD \Rightarrow X - M = K(D-M)$

$$K = \frac{(X-M)}{D-M}$$

14) Given,

A paging scheme which uses Translation Look-aside Buffer.

Also given,

TLB access = 10 ns

Main memory access = 50 ns

Hit ratio = 90%

Effective access time = Hit ratio × (TLB Access + Mem Access)

+

Miss ratio × (TLB Access + 2 × Mem Access)

= 0.9 × (10 + 50) + 0.1 (10 + 100)

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

15) Given,

Frames = m

Page-reference string = P

Distinct page numbers = n

a) Lower Bound

This occurs when the best case happens.

i.e) $n \leq m \leq P$

meaning all the pages will be within frames.

∴ In this case, page faults = $\boxed{n}$

b) Upper Bound

This occurs when the worst case happens

i.e) $m < n \leq P$

and incidently all the new entries would not be present in frames ⇒ All page faults

⇒ $\boxed{P}$