CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
No.132, AECS Layout
I.T. Park Road, Bangalore 560 037
T: +91 80 28524466/77, F : +91 80 28524630
E: info@cmr.ac.in I www.cmr.ac.in

Operating Systems (15CS64)
Internal Assessment Test – 3
Question paper Solution

1. Consider the following snapshot of a system:

	Allocation				Max				Available			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

i) Find out need matrix.

ii) If a request from process P1 arrived for (0,4,2,0) can the request be granted immediately?

iii) Is the system is in safe state.

4(b) For the given snapshot :

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	5	2	0
P2	1	0	0	0	1	7	5	0				
P3	1	3	5	4	2	3	5	6				
P4	0	6	3	2	0	6	5	2				
P5	0	0	1	4	0	6	5	6				

Using Banker's algorithm:

i) What is the need matrix content?

ii) Is the system in safe state?

iii) If a request from process P2(0,4,2,0) arrives, can it be granted? (10 Marks)

Ans:

Solution (i):

• The content of the matrix *Need* is given by

Need = Max - Allocation

• So, the content of Need Matrix is:

	Need			
	A	B	C	D
P1	0	0	0	0
P2	0	7	5	2
P3	1	0	0	2
P4	0	0	2	0
P5	0	6	4	2

Solution (ii):

• Applying the Safety algorithm on the given system,

Step 1: Initialization

Work = Available i.e. Work =1 5 2 0

....P1.....P2.....P3.....P4.....P5....

Finish = | false | false | false | false | false |

Step 2: For i=1

Finish[P1] = false and Need[P1]<=Work i.e. (0 0 0 0)<=(1 5 2 0) → true

So P1 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P1] =(1 5 2 0)+(0 0 1 2)=(1 5 3 2)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | false | false | false |

Step 2: For i=2

Finish[P2] = false and Need[P2]<=Work i.e. (0 7 5 2)<=(1 5 3 2) → false

So P2 must wait.

Step 2: For i=3

Finish[P3] = false and Need[P3]<=Work i.e. (1 0 0 2)<=(1 5 3 2) → true

So P3 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P3] = (1 5 3 2)+(1 3 5 4)=(2 8 8 6)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | false | false |

Step 2: For i=4

Finish[P4] = false and Need[P4]<=Work i.e. (0 0 2 0)<=(2 8 8 6) → true

So P4 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P4] =(2 8 8 6)+(0 6 3 2)=(2 14 11 8)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | true | false |

Step 2: For i=5

Finish[P5] = false and Need[P5]<=Work i.e. (0 6 4 2)<=(2 14 11 8) → true

So P5 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P5] =(2 14 11 8)+(0 0 1 4)=(2 14 12 12)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | true | true |

Step 2: For i=2

Finish[P2] = false and Need[P2]<=Work i.e. (0 7 5 2) <=(2 14 12 12) → true

So P2 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P2] =(2 14 12 12)+(1 0 0 0)=(3 14 12 12)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | true | true | true | true |

Step 4: Finish[Pi] = true for 1<=i<=5

Hence, the system is currently in a safe state.

The safe sequence is <P1, P3, P4, P5, P2>.

Conclusion: Yes, the system is currently in a safe state.

Solution (iii): P2 requests (0 4 2 0) i.e. Request[P2]= 0 4 2 0

• To decide whether the request is granted, we use Resource Request algorithm.

Step 1: Request[P2]<=Need[P2] i.e. (0 4 2 0) <= (0 7 5 2) → true.

Step 2: Request[P2]<=Available i.e. (0 4 2 0) <= (1 5 2 0) → true.

Step 3: Available = Available – Request[P2] = (1 5 2 0) - (0 4 2 0)= (1 1 0 0)

Allocation[P2] = Allocation[P2] + Request[P2] = (1 0 0 0) + (0 4 2 0)= (1 4 2 0)

Need[P2] = Need[P2] – Request[P2] = (0 7 5 2) - (0 4 2 0)= (0 3 3 2)

• We arrive at the following new system state

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	1	0	0
P2	1	4	2	0	1	7	5	0				
P3	1	3	5	4	2	3	5	6				
P4	0	6	3	2	0	6	5	2				
P5	0	0	1	4	0	6	5	6				

• The content of the matrix *Need* is given by

Need = Max - Allocation

• So, the content of Need Matrix is:

	Need			
	A	B	C	D
P1	0	0	0	0
P2	0	3	3	2
P3	1	0	0	2
P4	0	0	2	0
P5	0	6	4	2

• Applying the Safety algorithm on the given system,

Step 1: Initialization

Work = Available i.e. Work =1 1 0 0

....P1.....P2.....P3.....P4.....P5....

Finish = | false | false | false | false | false |

Step 2: For i=1

Finish[P1] = false and Need[P1]<=Work i.e. (0 0 0 0)<=(1 1 0 0) → true

So P1 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P1] =(1 1 0 0)+(0 0 1 2)=(1 1 1 2)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | false | false | false |

Step 2: For i=2

Finish[P2] = false and Need[P2]<=Work i.e. (0 3 3 2)<=(1 1 1 2) →false

So P2 must wait.

Step 2: For i=3

Finish[P3] = false and Need[P3]<=Work i.e. (1 0 0 2)<=(1 1 1 2) → true

So P3 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P3] = (1 1 1 2)+(1 3 5 4)=(2 4 6 6)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | false | false |

Step 2: For i=4

Finish[P4] = false and Need[P4]<=Work i.e. (0 0 2 0)<=(2 4 6 6) → true

So P4 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P4] =(2 4 6 6)+(0 6 3 2)=(2 10 9 8)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | true | false |

Step 2: For i=5

Finish[P5] = false and Need[P5]<=Work i.e. (0 6 4 2)<=(2 10 9 8) → true

So P5 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P5] =(2 10 9 8)+(0 0 1 4)=(2 10 10 12)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | false | true | true | true |

Step 2: For i=2

Finish[P2] = false and Need[P2]<=Work i.e. (0 3 3 2) <=(2 10 10 12) → true

So P2 must be kept in safe sequence.

Step 3: Work = Work + Allocation[P2] =(2 10 10 12)+(1 4 2 0)=(3 14 12 12)

....P1.....P2.....P3.....P4.....P5...

Finish = | true | true | true | true | true |

Step 4: Finish[Pi] = true for 0<=i<=4

Hence, the system is currently in a safe state.

The safe sequence is <P1, P3, P4, P5, P2>.

Conclusion: Since the system is in safe sate, the request can be granted.

2. For the following page reference calculate the page fault that occur using FIFO, LRU and optimal for 3 and 4 frames respectively. 5,4,3,2,1,4,3,5,4,3,2,1,5.

5(c) For the following page reference, calculate the page faults that occur using FIFO and LRU for 3 and 4 page frames respectively

5, 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5. (10 Marks)

Ans:

(i) LRU with 3 frames:

Frames	5	4	3	2	1	4	3	5	4	3	2	1	5
1	5	5	5	2	2	2	3	3	3	3	3	3	5
2		4	4	4	1	1	1	5	5	5	2	2	2
3			3	3	3	4	4	4	4	4	4	1	1
No. of Page faults	√	√	√	√	√	√	√	√			√	√	√

No of page faults=11

(ii) LRU with 4 frames:

Frames	5	4	3	2	1	4	3	5	4	3	2	1	5
1	5	5	5	5	1	1	1	1	1	1	2	2	2
2		4	4	4	4	4	4	4	4	4	4	4	5
3			3	3	3	3	3	3	3	3	3	3	3
4				2	2	2	2	5	5	5	5	1	1
No. of Page faults	√	√	√	√	√			√			√	√	√

No of page faults=9

(iii) LRU with 4 frames::

Frames	5	4	3	2	1	4	3	5	4	3	2	1	5
1	5	4	3	2	1	4	3	5	5	5	2	1	1
2		5	4	3	2	1	4	3	3	3	5	2	2
3			5	4	3	2	1	4	4	4	3	5	5
No. of Page faults	√	√	√	√	√	√	√	√			√	√	

No of page faults=10

(iv) FIFO with 4 frames:

Frames	5	4	3	2	1	4	3	5	4	3	2	1	5
1	5	4	3	2	1	1	1	5	4	3	2	1	5
2		5	4	3	2	2	2	1	5	4	3	2	1
3			5	4	3	3	3	2	1	5	4	3	2
4				5	4	4	4	3	2	1	5	4	3
No. of Page faults	√	√	√	√	√			√	√	√	√	√	√

No of page faults=11

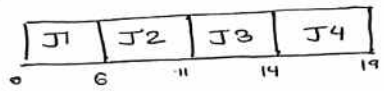
3. Consider the following set of process with arrival and burst time.
A larger priority number has a highest priority

Jobs	Arrival Time ms	Burst Time ms	Priority
J1	0	6	4
J2	3	5	2
J3	3	3	6
J4	5	5	3

Draw the Gantt chart and calculate waiting time and turnaroundtime using i)FCFS ii) SJF iii)SRTF iv) priority v) Round Robin (Q=3)

Answers:

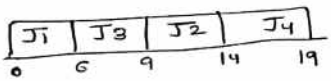
I) i) FCFS



Avg. waiting time.
$J_1 = 0-0=0$, $J_2 = 6-3=3$
$J_3 = 11-3=8$, $J_4 = 14-5=9$
$\Rightarrow (0+3+8+9)/4$
$= 6$

Avg. TAT
$\Rightarrow \frac{6+11+14+19}{4}$
$= 12.5$

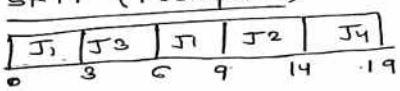
ii) SJF (Non-preemptive)



Avg. waiting time
$J_1 = 0-0=0$, $J_2 = 9-3=6$
$J_3 = 6-3=0$, $J_4 = 14-5=9$
$\Rightarrow \frac{0+6+3+9}{4}$
$= 4.5$

Avg. TAT
$\Rightarrow \frac{6+9+14+19}{4}$
$= 12$

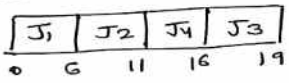
iii) SRTF (Premptive)



Avg. waiting time
$J_1 = 6-3-0=3$, $J_2 = 9-3=6$
$J_3 = 3-3=0$, $J_4 = 14-5=9$
$\Rightarrow \frac{3+6+0+9}{4}$
$= 4.5$

Avg. TAT
$J_1 = 9-3=6$, $J_2 = 14$
$J_3 = 6$, $J_4 = 19$
$\Rightarrow \frac{6+14+6+19}{4}$
$= 11.25$

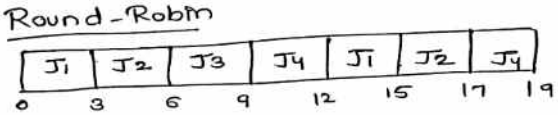
iv) Priority



Avg. waiting time.
$J_1 = 0-0=0$, $J_3 = 11-3=9$
$J_2 = 6-3=3$, $J_4 = 16-5=11$
$\Rightarrow \frac{0+3+9+11}{4}$
$= 5.6$

Avg. TAT.
$\Rightarrow \frac{6+11+16+19}{4}$
$= 13$

v) Round-Robin



Avg. waiting time.
$J_1 = 12-3=9$, $J_2 = 15-3-3=9$
$J_3 = 6-3=3$, $J_4 = 17-9-5=3$
$\Rightarrow \frac{9+3+9+3}{4}$
$= 6$

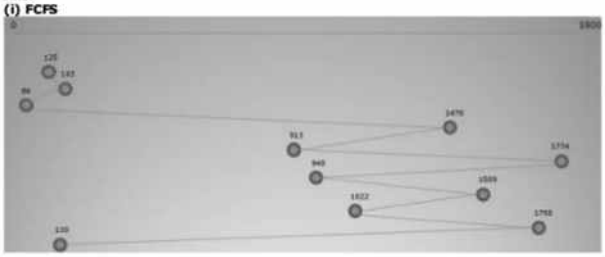
Avg. TAT.
$J_1 = 15-3=12$, $J_2 = 17-3=14$
$J_3 = 9$, $J_4 = 19-12=7$
$\Rightarrow \frac{12+9+14+7}{4}$
$= 10.5$

4. Suppose that the disk drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current(location) head position, What is the total distance(in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms? i) FCFS ii) SSTF iii)SCAN iv)LOCK v) C-SCAN

7(a) Suppose that the disk drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current (location) head position, what is the total distance (in cylinders) that the disk-arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms? (15 Marks)
(i) FCFS; (ii) SSTF; (iii) SCAN; (iv) LOCK; (v) C-SCAN

Ans:

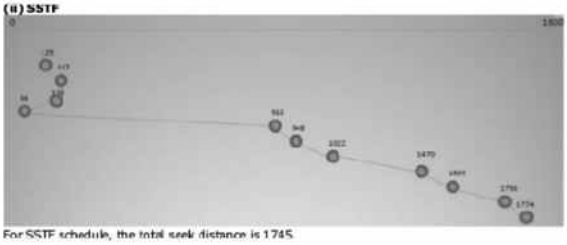
(i) FCFS



From cylinder	To cylinder	Seek Time
143	86	57
86	1470	1384
1470	913	557
913	1774	861
1774	948	826
948	1509	561
1509	1022	487
1022	1750	728
1750	130	1620
Total Seek Time		7081

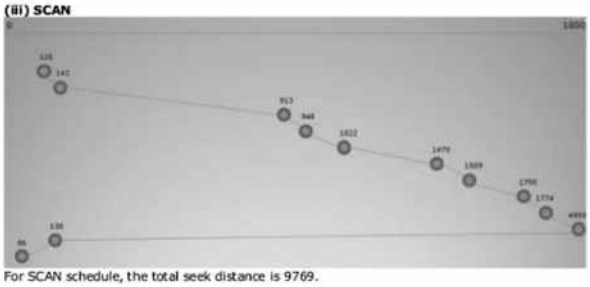
For FCFS schedule, the total seek distance is 7081.

(ii) SSTF



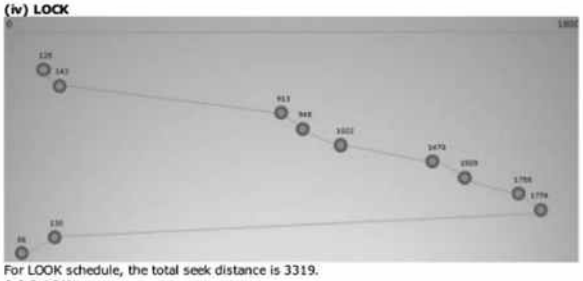
For SSTF schedule, the total seek distance is 1745.

(iii) SCAN



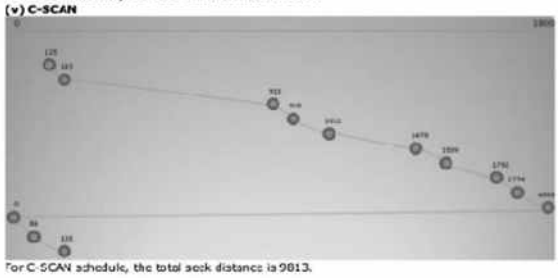
For SCAN schedule, the total seek distance is 9769.

(iv) LOCK



For LOOK schedule, the total seek distance is 3319.

(v) C-SCAN



For C-SCAN schedule, the total seek distance is 9813.

PART – B (choose any 4 questions from this part) [5*4 = 20]

5. What are the major methods used for allocating a disk space? Explain each with suitable examples.

Allocation Methods

• The direct-access nature of disks allows us flexibility in the implementation of files.
• In almost every case, many files are stored on the same disk.
• Main problem:
How to allocate space to the files so that
→ disk-space is utilized effectively and

→ files can be accessed quickly.
• Three methods of allocating disk-space:
1. Contiguous
2. Linked and
3. Indexed
• Each method has advantages and disadvantages.
• Some systems support all three (Data General's RDOS for its Nova line of computers).

Contiguous Allocation

• Each file occupies a set of contiguous-blocks on the disk (Figure 6.17).
• Disk addresses define a linear ordering on the disk.
• The number of disk seeks required for accessing contiguously allocated files is minimal.
• Both sequential and direct access can be supported.
• Problems:
1. Finding space for a new file
□ □External fragmentation can occur.
2. Determining how much space is needed for a file.

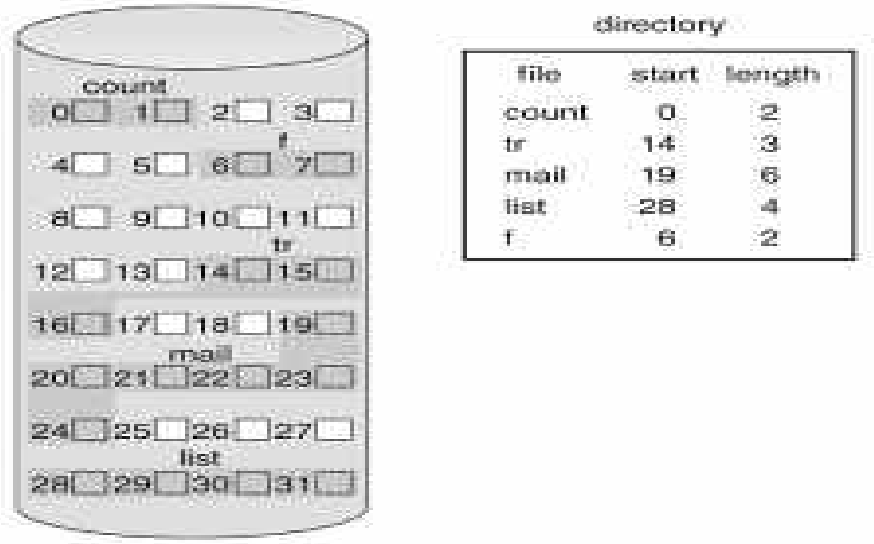


Fig : Contiguous allocation of disk-space

□ □If you allocate too little space, it can't be extended.

Two solutions:

i) The user-program can be terminated with an appropriate error-message. The user must then allocate more space and run the program again.

ii) Find a larger hole,

copy the contents of the file to the new space and release the previous space.

• To minimize these drawbacks:
1. A contiguous chunk of space can be allocated initially and
2. Then when that amount is not large enough, another chunk of contiguous space

(known as an ‘extent’) is added.
Figure 6.17 Contiguous allocation of disk-space

Linked Allocation

• Each file is a linked-list of disk-blocks.
• The disk-blocks may be scattered anywhere on the disk.
• The directory contains a pointer to the first and last blocks of the file (Figure 6.18).
• To create a new file, just create a new entry in the directory (each directory-entry has a pointer to the disk-block of the file).
1. A **write** to the file causes a free block to be found. This new block is then written to and linked to the eof (end of file).
2. A **read** to the file causes moving the pointers from block to block.
• Advantages:
1. No external fragmentation, and any free block on the free-space list can be used to satisfy a request.
2. The size of the file doesn't need to be declared on creation.
3. Not necessary to compact disk-space.
• Disadvantages:
1. Can be used effectively only for sequential-access files.
2. Space required for the pointers.
Solution: Collect blocks into multiples (called ‘clusters’) & allocate clusters rather than blocks.
3. Reliability: Problem occurs if a pointer is lost(or damaged).
Partial solutions: i) Use doubly linked-lists.

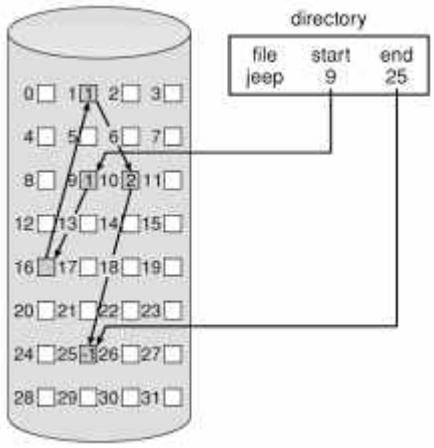


Fig : Linked allocation of disk-space

ii) Store file name and relative block-number in each block.
Figure 6.18 Linked allocation of disk-space Figure 6.19 File-allocation table
• FAT is a variation on linked allocation (FAT=File Allocation Table).

• A section of disk at the beginning of each partition is set aside to contain the • The table
→ has one entry for each disk-block and
→ is indexed by block-number.
• The directory-entry contains the block-number of the first block in the file.
• The table entry indexed by that block-number then contains the block-number of the next block in the file.
• This chain continues until the last block, which has a special end-of-file value as the table entry.
• Advantages:
1. Cache can be used to reduce the no. of disk head seeks.
2. Improved access time, since the disk head can find the location of any block by reading the info in the FAT.

Indexed Allocation

• Solves the problems of linked allocation (without a FAT) by bringing all the pointers together into an
index block.
• Each file has its own index block, which is an array of disk-block addresses.
• The ith entry in the index block points to the ith file block (Figure 6.20).
• The directory contains the address of the index block.

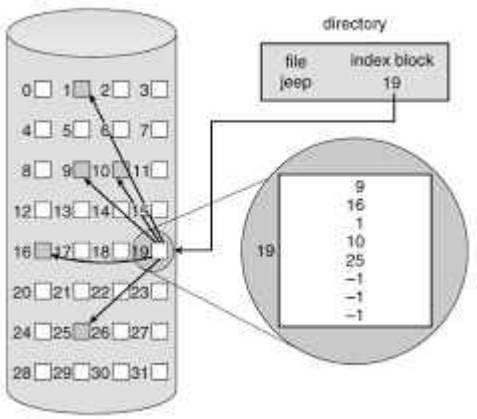


Fig : Indexed allocation of disk space

• When the file is created, all pointers in the index-block are set to nil.
• When writing the ith block, a block is obtained from the free-space manager, and its address put in
the ith index-block entry,
• Problem: If the index block is too small, it will not be able to hold enough pointers for a large file,

6. Explain briefly the access matrix with domains as objects.

Access Matrix

- The model of protection that we have been discussing can be viewed as an ***access matrix,*** in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Domain switching can be easily supported under this model, simply by providing "switch" access to other domains:

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure 14.4 - Access matrix of Figure 14.3 with domains as objects.

- The ability to ***copy*** rights is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e. for the same object. There are two important variations:

o If the asterisk is removed from the original access right, then the right is ***transferred,*** rather than being copied. This may be termed a ***transfer*** right as opposed to a ***copy*** right.

o If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propagated further. That is the new domain does not also receive the right to copy the access. This may be termed a ***limited copy*** right, as shown in Figure 14.5 below:

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Figure 14.5 - Access matrix with *copy* rights.

- The ***owner*** right adds the privilege of adding new rights or removing existing ones:

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Copy and owner rights only allow the modification of rights within a column. The addition of ***control rights***, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains. For example in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

7. What do you mean by a free space list? With suitable examples, explain any 2 methods of implementation of a free space list.

• A ***free-space list*** keeps track of free disk-space (i.e. those not allocated to some file or directory).
• To create a file,
1. We search the free-space list for the required amount of space.
2. Allocate that space to the new file.
3. This space is then removed from the free-space list.
• To delete a file, its disk-space is added to the free-space list.

Bit Vector

• The free-space list is implemented as a bit map/bit vector.
• Each block is represented by a bit.
1. If the block is free, the bit is 1.
2. If the block is allocated, the bit is 0.
• For example, consider a disk where blocks 2, 3, 4, 5 and 7 are free and the rest of the blocks are allocated. The free-space bit map will be 00111101
• Advantage:
1. Relative simplicity & efficiency in finding the first free block, or ‘n’ consecutive free blocks.
• Disadvantages:
1. Inefficient unless the entire vector is kept in main memory.
2. The entire vector is written to disc occasionally for recovery.

Linked List

• The basic idea:

1. Link together all the free disk-blocks (Figure 6.21).
2. Keep a pointer to the first free block in a special location on the disk.
3. Cache the block in memory.
• The first block contains a pointer to the next free one, etc.
• Disadvantage:
1. Not efficient, because to traverse the list, each block is read.
• Usually the OS simply needs a free block, and uses the first one.

Grouping

• The addresses of n free blocks are stored in the 1st free block.
• The first n-1 of these blocks are actually free.
• The last block contains addresses of another n free blocks, etc.
• Advantage:
1. Addresses of a large no of free blocks can be found quickly.

Counting

• Takes advantage of the fact that, generally, several contiguous blocks may be allocated/freed simultaneously.
• Keep the address of the first free block and the number 'n' of free contiguous blocks that follow the first block.
• Each entry in the free-space list then consists of a disk address and a count.

8. Discuss the directory implementation using a) Linear List b) Hash table.

Directory Implementation

1. Linear-list
2. Hash-table

Linear List

• A linear-list of file-names has pointers to the data-blocks.
• To create a new file:
1. First search the directory to be sure that no existing file has the same name.
2. Then, add a new entry at the end of the directory.
• To delete a file:
1. Search the directory for the named-file and
2. Then release the space allocated to the file.
• To reuse the directory-entry, there are 3 solutions:
1. Mark the entry as unused (by assigning it a special name).
2. Attach the entry to a list of free directory entries.
3. Copy the last entry in the directory into the freed location & to decrease length of directory.
• Problem: Finding a file requires a linear-search which is slow to execute.
Solutions:
1. A cache can be used to store the most recently used directory information.

2. A sorted list allows a binary search and decreases search time.
• Advantage:
1. Simple to program.
• Disadvantage:
1. Time-consuming to execute.

Hash Table

• A linear-list stores the directory-entries. In addition, a hash data-structure is also used.
• The hash-table
→ takes a value computed from the file name and
→ returns a pointer to the file name in the linear-list.
• Advantages:
1. Decrease the directory search-time.
2. Insertion & deletion are easy.
• Disadvantages:
1. Some provision must be made for collisions i.e. a situation in which 2 file-names hash to the same location.
2. Fixed size of hash-table and the dependence of the hash function on that size.

9. Discuss the steps in handling a page fault, with the help of a neat diagram.

Steps in handling a page-fault

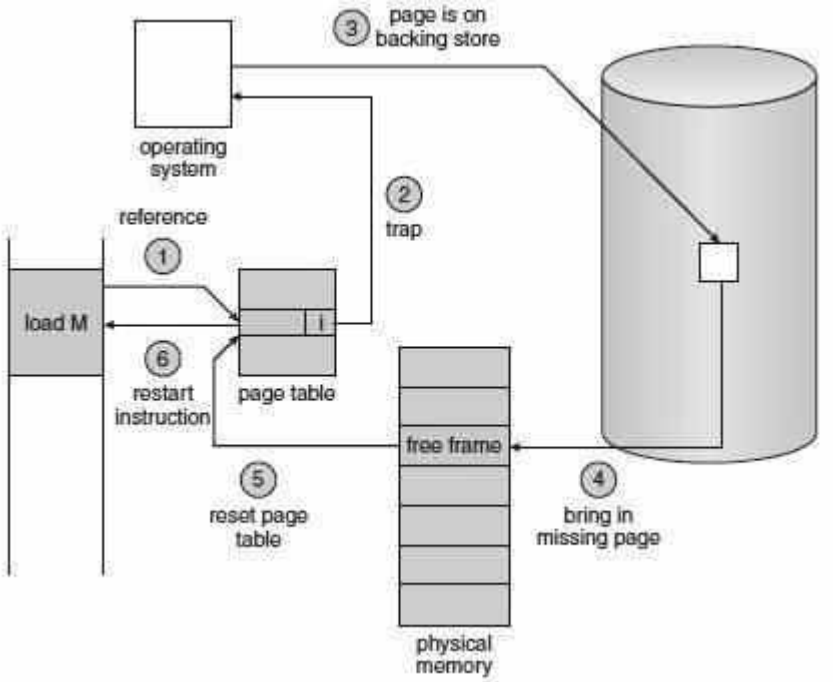


Fig : Steps in handling a page-fault

A **page-fault** occurs when the process tries to access a page that was not brought into

memory**.**
• Procedure for handling the page-fault (Figure)

Steps:

Steps 1. Check an internal-table to determine whether the reference was a valid or an invalid memory access.
Steps 2. If the reference is invalid, we terminate the process.If reference is valid, but we have not yet brought in that page, we now page it in.
Steps 3. Find a free-frame (by taking one from the free-frame list, for example).
Steps 4. Read the desired page into the newly allocated frame.
Steps 5. Modify the internal-table and the page-table to indicate that the page is now in memory.
Steps 6. Restart the instruction that was interrupted by the trap.

10. Explain the different IPC mechanism available in LINUX.

Interprocess Communication

- Like UNIX, Linux informs processes that an event has occurred via signals
- There is a limited number of signals, and they cannot carry information:
- Only the fact that a signal occurred is available to a process
- The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait.queue** structures

Passing Data Between Processes

- The pipe mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read at the other.
- Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
- To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism

Shared Memory Object

- The shared-memory object acts as a backing store for shared-memory regions in the same way as a file can act as backing store for a memory mapped memory region
- Shared-memory mappings direct page faults to map in pages from a persistent shared-memory object
- Shared-memory objects remember their contents even if no processes are currently

mapping them into virtual memory

11. Write short notes on process scheduling and kernel synchronization.

Process Scheduling (Cont)

 Linux uses two process-scheduling algorithms:
 A time-sharing algorithm for fair preemptive scheduling between multiple processes
 A real-time algorithm for tasks where ea e a go o as s e e absolute priorities are more important than fairness
 A process's scheduling class defines which algorithm to apply
 For time-sharing processes, Linux uses a prioritized, credit based algorithm
 The crediting rule
priority
2
credits := credits + factors in both the process's history and its priority
 This crediting system automatically prioritizes interactive or I/Obound processes
 Linux implements the FIFO and round-robin real-time scheduling classes; in both cases, each process has a priority in addition to its scheduling class
 The scheduler runs the process with the highest priority; for equalpriority processes, it runs the process waiting the longest
 FIFO processes continue to run until they either exit or block
 A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robing processes of equal priority automatically time-share between themselves

kernel synchronization

To avoid performance penalties, Linux's kernel uses a synchronization architecture that allows long critical sections to run without having interrupts disabled for the critical section's entire duration
 Interrupt service routines are separated into a *top half* and a *bottom half.*
 The top half is a normal interrupt service routine, and runs with recursive interrupts disabled
 The bottom half is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves

 This architecture is completed by a mechanism for disabling selected bottom halves while executing normal, foreground kernel code