# INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

| END SEMESTER EXAMINATION | | | | | | | | | | SPRING SEMESTER | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Roll Number | | | | | | | | | | Section | | Name | |
| Subject Number | C | S | 3 | 1 | 2 | 0 | 2 | Subject Name | | | | *Operating Systems* | |
| Department / Center of the student | | | | | | | | | | | | Additional Sheets | |

## Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items of any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the Invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging Information with others or any such attempt will be treated as '**unfair means**'. Do not adopt unfair means and do not indulge in unseemly behavior.

***Violation of any of the above instructions may lead to severe punishment.***

**Signature of the Student**

*To be filled in by the examiner*

| Question Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Marks obtained | | | | | | | | | | | |

| Marks Obtained (in words) | Signature of the Examiner | Signature of the Scrutineer |
|---|---|---|
| | | |

# CS31202 Operating Systems

**End-Semester Examination**

**25-Apr-2026, 03:00pm – 05:00pm**

**Maximum Marks: 80**

---

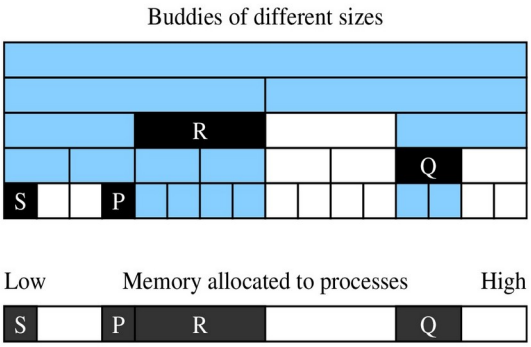
## Instructions to students

- Please write in the spaces provided in the question paper itself. Be brief and precise.
- For rough work and leftover answers (if needed), you can use the extra blank pages provided at the end. You are given sufficient extra space, so try to avoid taking supplementary sheets from the invigilators.
- If you use extra page(s) for any leftover answer, please give a pointer (number of the extra page) from the given space where you begin the answer. Without this, we will not look into the extra pages.
- Do not write anything on this page. Questions start from the next page (Page 3).

**1. [*Contiguous memory allocation*]**

A contiguous memory-allocation scheme works as follows. Let the total available memory have a size of $2^t$ (unless otherwise specified, all sizes in this question are in bytes). This is broken into two adjacent contiguous chunks (called buddies) of size $2^{t-1}$ each. Each buddy is broken down into two adjacent sub-buddies (contiguous again) of size $2^{t-2}$ each, and so on. When a process $P$ with memory requirement $m$ (bytes) comes, the smallest power $2^k \geqslant m$ is calculated. A buddy of size $2^k$ is allocated to the process, provided that such a buddy is available. A buddy is available (also called free) if it itself is not allocated to a process, and no part of it (that is, no sub-buddy of a smaller size) is allocated to any process. If no buddy of size $2^k$ is available, allocation fails.

The figure to the right shows the (same) memory broken up into buddies of different sizes. There are four processes P, Q, R, and S currently in the memory. Buddies allocated to theses processes are blackened. Buddies that are not available for allocation are shaded. The unshaded buddies are available.




In order to keep track of the free buddies of different sizes, the OS uses a bit array for all buddies at each level of buddy sizes. The bit indicates whether a buddy is available (1) or not available (0). Assume that the low end of the memory is to the left, and the high end is to the right. For the situation shown in the adjacent figure, the arrays at the top five levels (numbered 0 through 4) are [0], [00], [0010], [00001101], and [0110000011110011].

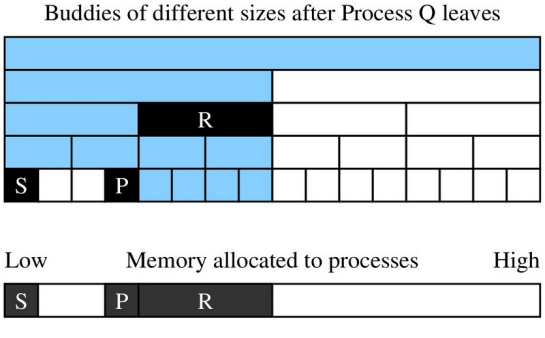
**(a)** Explain the general procedure how the free-buddies arrays are modified by the OS when a new process of a specified memory requirement arrives and is allocated a free buddy $B$ at the appropriate level $l$. Mention the indices at which the free-buddies arrays are modified, and how (say, $B$ is at index $i$ of the level-$l$ array). **[4]**

Suppose that the new process is of memory requirement $m$ satisfying $2^{k-1} < m \leqslant 2^k$. If there are no free buddies in the array at level $l = t - k$, then allocation fails. So suppose that one or more free buddies of size $2^k$ is/are available. One such buddy $B$ is allocated to the process. That buddy is marked unavailable.

Assume that array indexing is zero-based. Suppose that the allocated buddy $B$ is at index $i$ in the level $l$ array. All sub-buddies under the allocated buddy are marked unavailable. More precisely, we mark the buddies $2i$ and $2i + 1$ in the level $l + 1$ array as unavailable, the buddies $4i$, $4i + 1$, $4i + 2$, and $4i + 3$ at level $l + 2$ as unavailable, and so on.

Moreover, the parent buddy of $B$ (at index $i / 2$ in the $l - 1$ level array), the grandparent buddy of $B$ (at index $i / 4$ in the $l - 2$ level array), the great grandparent buddy (at index $i / 8$ in the $l - 2$ level array), and so on are all marked as unavailable (because a part of them is allocated to the new process).

**(b)** If a process $P$ that was allocated a buddy of size $2^k$ terminates, the buddy allocated to $P$ is freed. If the adjacent buddy is also free, these are combined to a free buddy of size $2^{k+1}$. If the adjacent buddy of the combined buddy is also free, these are combined to free a buddy of size $2^{k+2}$, and so on. The figure to the right explains this procedure if Process Q (in the above figure) terminates. The free-buddies arrays now become [0], [01], [0011], [00001111], and [0110000011111111]. Explain the general procedure how these arrays (mention the indices) are modified by the OS when a process terminates. **[4]**



Suppose that the departing process was allocated a buddy $B$ at level $l$. That buddy is first marked available.

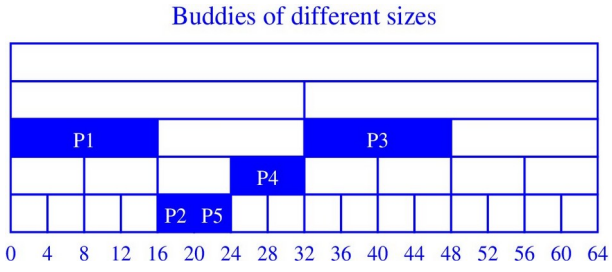
Assume that the freed buddy $B$ is at index $i$ in the level $l$ array. All sub-buddies under the freed buddy $B$ are marked available. More precisely, we mark the buddies $2i$ and $2i + 1$ in the level $l + 1$ array as available, the buddies $4i$, $4i + 1$, $4i + 2$, and $4i + 3$ at level $l + 2$ as available, and so on.

Moreover, we now handle merging of free buddies. If $i = 2j$, then consider the buddy $B'$ at index $2j + 1$ in the level $l$ array, whereas if $i = 2j + 1$, then consider the buddy $B'$ at index $2j$ in the level $l$ array. $B$ is now marked free. If $B'$ is also free, we mark the buddy $P$ (parent) at index $j$ of the level $l - 1$ array as available. We now repeat the same process with $B$ replaced by $P$. This continues until no more merging is possible, or we have reached the top level (level 0).

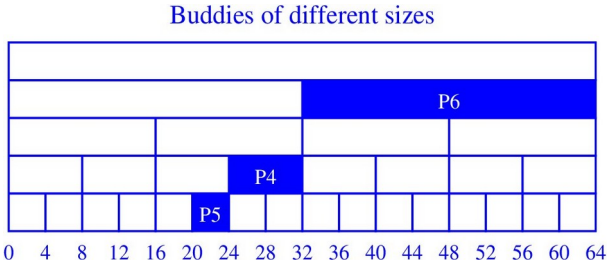
**(c)** Assume that the free buddy given to a process is at the lowest possible location in memory (among all the free buddies at that level). A machine has 64 MB memory. Initially, the memory is completely free. Then, the following processes arrive in the given sequence.

P1 of memory requirement 12 MB arrives.
P2 of memory requirement 3 MB arrives.
P3 of memory requirement 15 MB arrives.
P4 of memory requirement 6 MB arrives.
P5 of memory requirement 4 MB arrives.

Draw the allocated buddies after these five allocations. For this exercise, you do not need buddies of sizes 2 MB or less. Show only the buddies of sizes 4 MB – 64 MB. Draw the allocation as in the figures on the last page. Show only the buddies of sizes of interest. Mark the allocated buddies by process names. There is no need to shade the unavailable buddies, or to draw the memory map as shown at the bottom of the figures on the previous page. **[4]**



**(d)** If a process cannot be allocated memory, it waits until one or more running processes leave, creating a free buddy of the desired size to accommodate the waiting process. Suppose that the five processes P1 through P5 of Part (c) are still in memory, and then a new process P6 of memory requirement 21 MB arrives. Since there is not enough free space left to accommodate P6, it has to wait. Suppose that the processes P1 through P5 leave in the same sequence as they arrive (that is, P1 leaves first, then P2, then P3, and so on). How many of the existing processes must leave before P6 can be allocated memory? Draw the allocated buddies after the allocation of P6. **[4]**




**(e)** Explain the fragmentation issues associated with this contiguous memory-allocation scheme. That is, explain what type(s) of fragmentation (external/internal/both) can happen, and what is the worst-case memory loss (in percentage) due to each possible type of fragmentation. **[4]**

Suppose that the memory requirement of a process is $m$. It is given a buddy of size $2^k$ satisfying $2^{k-1} < m \leqslant 2^k$. Then, the space lost due to internal fragmentation is $2^k - m$ bytes. This is maximized when $m = 2^{k-1} + 1$, and the fraction of memory loss is

$$(2^{k-1} - 1) / (2^{k-1} + 1) \approx \frac{1}{2}.$$

That is, up to (almost) 50% memory may be lost due to internal fragmentation.

External fragmentation is also possible. Assume that the memory size is $2^t$. A (hypothetical) process of size 1 appears when the memory is empty. It is given a buddy (at level $t$) of size 1. But then, all ancestors of that buddy are now unavailable. If a process of memory requirement $2^{t-1} + 1$ arrives, it cannot be allocated any buddy (indeed it requires the buddy of size $2^t$ at level 0), although $2^t - 1$ bytes of the memory are still available. Only a new process of size $\leqslant 2^{t-1}$ can still be allocated memory. So external fragmentation is also

$$(2^{t-1} - 1) / (2^t - 1) \approx \frac{1}{2},$$

that is, about 50%. Note that if the process of memory requirement $2^{t-1} + 1$ arrives first, then it is given the entire memory of size $2^t$, and even the 1 byte process cannot be accommodated now. This time it is internal fragmentation of about 50%.

**2. [IA-32 *segmentation*]**

A process P uses only local segments. A segment of size ⩽ 1 MB is called small, and a segment of size > 1 MB is called large. In the linear address space, small segments are 16-byte aligned (that is, start from an address that is a multiple of 16). Large segments, on the other hand, are page aligned, that is, start at page boundaries and are given whole numbers of pages. Recall that IA-32 uses 4 KB pages. A process P consists of the following segments.

| Segment Number | Segment size (bytes in decimal) | Segment Type |
|---|---|---|
| 0 | 1234 | Small |
| 1 | 56789 | Small |
| 2 | 1234567 | Large |
| 3 | 13579 | Small |
| 4 | 2345678 | Large |
| 5 | 9876543 | Large |

In the linear address space, the segments are placed in the sequence 0, 1, 2, 3, 4, 5. Segment 0 starts at location 0. After that, the other segments are placed at the lowest possible memory locations following the alignment constraints mentioned above. In connection with this situation, answer the following questions.

**(a)** In the following table, work out the starting and the last addresses of the segments in the linear address space. **[6]**

| Segment Number | Segment size (bytes in decimal) | Starting Address | Last Address |
|---|---|---|---|
| 0 | 1234 | 0 | 1233 (0x4d1) |
| 1 | 56789 | 1248 (0x4e0) | 58036 (0xe2b4) |
| 2 | 1234567 | 61440 (0xf000) | 1296006 or 1298431 (0x13c686 or 0x13cfff) |
| 3 | 13579 | 1298432 (0x13d000) | 1312010 (0x14050a) |
| 4 | 2345678 | 1314816 (0x141000) | 3660493 or 3661823 (0x37dacd or 0x37dfff) |
| 5 | 9876543 | 3661824 (0x37e000) | 13538366 or 13541375 (0xce943e or 0xce9fff) |

(No need to show the numbers in hexadecimal)

**(b)** Convert the logical address (4, 321098) (this is in the (segment number, offset) form) to the linear address as a byte number, and also in the (page number, offset) form. **[2]**

1314816 + 321098 = 1635914 = 399 × 4096 + 1610 = (399, 1610) = (0x18f, 0x64a).
(No need for showing the hex equivalent)

**(c)** Calculate the number of entries in the page table of P. **[2]**

$\lceil 13538366 / 4096 \rceil = 3306.$

**3. [*Virtual memory*]**

A process has 10 pages (numbered 0 – 9), and is given 4 frames (F0, F1, F2, and F3, to be allocated initially in this sequence). Pure demand paging is used with page replacement based on a working set window of size 4. If the working-set size goes below 4, the loaded pages that are not in the working set are not immediately removed. Page replacement is deferred until a page fault happens. If multiple pages are candidates for replacement, the least recently used one is replaced. The page table is organized in the form of a hash table of size 7 with open addressing (not chaining). The table stores only the loaded pages, and uses the hash function $h(p) = (3p+5)\%7$ to calculate indices in the table, where $p$ is a page number. Each entry in the hash table can be empty, or deleted, or a pair (page number, allocated frame). If a page is removed from a frame, its entry is deleted from the hash table. The entry for a newly loaded page is inserted in an empty or a deleted cell. If an occupied cell is found at the hash index, the (circularly) next index is probed. If that entry is occupied too, the next index is probed, and so on. Show how the page-replacement algorithm works on the reference string 6,5,9,8,9,8,5,1,0,3,7,3,4,0,2. Present your answer in the table below. Mark the deleted cells in the page table until replaced by the entry of a new page. Index 0 of the page table is on the left. **[10]**

| Page referenced | | Working set window | Working set | Page fault (Y/N) |
|---|---|---|---|---|
| 6 | | 6 | { 6 } | Y |
| | Page table | 6, F0 | | |
| 5 | | 65 | { 5, 6 } | Y |
| | Page table | 6, F0 | 5, F1 | |
| 9 | | 659 | { 5, 6, 9 } | Y |
| | Page table | 6, F0 | 9, F2 | 5, F1 |
| 8 | | 6598 | { 5, 6, 9, 8} | Y |
| | Page table | 8, F3 | 6, F0 | 9, F2 | 5, F1 |
| 9 | | 5989 | { 5, 8, 9 } | N |
| | Page table | 8, F3 | 6, F0 | 9, F2 | 5, F1 |
| 8 | | 9898 | { 8, 9 } | N |
| | Page table | 8, F3 | 6, F0 | 9, F2 | 5, F1 |
| 5 | | 8985 | { 5, 8, 9 } | N |
| | Page table | 8, F3 | 6, F0 | 9, F2 | 5, F1 |
| 1 | | 9851 | { 1, 5, 8, 9 } | Y |
| | Page table | 8, F3 | 1, F0 | 9, F2 | 5, F1 |
| 0 | | 8510 | { 0, 1, 5, 8 } | Y |
| | Page table | 8, F3 | 1, F0 | DEL | 0, F2 | 5, F1 |
| 3 | | 5103 | { 0, 1, 3, 5 } | Y |
| | Page table | 3, F3 | DEL | 1, F0 | DEL | 0, F2 | 5, F1 |
| 7 | | 1037 | { 0, 1, 3, 7 } | Y |
| | Page table | 3, F3 | DEL | 1, F0 | DEL | 0, F2 | 7, F1 |
| 3 | | 0373 | { 0, 3, 7 } | N |
| | Page table | 3, F3 | DEL | 1, F0 | DEL | 0, F2 | 7, F1 |
| 4 | | 3734 | { 3, 4, 7 } | Y |
| | Page table | 3, F3 | DEL | DEL | 4, F0 | DEL | 0, F2 | 7, F1 |
| 0 | | 7340 | { 0, 3, 4, 7 } | N |
| | Page table | 3, F3 | DEL | DEL | 4, F0 | DEL | 0, F2 | 7, F1 |
| 2 | | 3402 | { 0, 2, 3, 4 } | Y |
| | Page table | 3, F3 | DEL | DEL | 4, F0 | 2, F1 | 0, F2 | DEL |

**4. [*Linked allocation of files with and without FAT*]**

A solid state disk has a total capacity of 16 TB with a block size of 4 KB. The disk access time is 100 microseconds per block, and the main memory has a bandwidth of $10^7$ bytes/s. Each directory entry consists of a file name (32 bytes) and a pointer (8 bytes) to the corresponding File Control Block (FCB). The FCB of a file contains file attributes and the address of the first data block of the file. The FCB occupies 4 KB.

**(a)** Assume that the system maintains a FAT which initially resides on the disk. Compute the time to load the FAT to the main memory. **[3]**

Disk size = $2^{44}$ bytes, block size = $2^{12}$ bytes
Total number of blocks = $2^{32}$
FAT entry = 4 bytes
FAT size = $2^{32} \times 4$ = 16 GB
Time to load FAT = (16 GB / 4 KB) × 0.0001 sec = 419.4304 sec

**(b)** Consider a file A of size 8 MB stored on the disk. When a sequential read request for file A is issued, the system first spends 10 ms to locate the file name in the directory (this is in the main memory). It then reads the corresponding directory entry (in order to retrieve the FCB pointer from it), reads the entire FCB (this is also residing in the main memory), and finally reads the data blocks of the file sequentially from the disk. Consider two file allocation methods: (i) FAT allocation (assume that the FAT has been cached in main memory), and (ii) linked allocation (without FAT). Calculate the total times required to read the entire file A sequentially using (i) FAT and (ii) linked allocation (without FAT). Clearly show all the components of the access times, and identify which method is faster. **[4]**

File size = 8 MB = 8192 KB, Block size = 4 KB
Number of blocks in the file = 8192 / 4 = 2048

Directory search = 10 ms
Read FCB pointer (8 bytes) from directory entry. Main memory bandwidth = $10^7$ bytes/s.
Time to read the directory entry = 40 / $10^7$ sec = 0.004 ms

FCB Size = 4 KB = 4096 bytes
Time to read FCB = 4096 / $10^7$ sec = 0.4096 ms

Total Directory Access Time:
10 + 0.004 + 0.4096 = 10.4136 ms

(i) FAT

Lookup per block:
Each FAT entry = 4 bytes
Total FAT traversal time = 2048 × 4 / $10^7$ sec = 0.8192 ms

Disk Access Time:
Disk access per block = 100 μs = 0.1 ms
Time to access the complete file: 2048 × 0.1 = 204.8 ms

Total Time (FAT) = 10.4136 + 0.8192 + 204.8 = 216.0328 ms

(ii) Linked allocation without FAT

Number of blocks in the linked allocation without FAT
$= \lceil 8 \times 2^{20} / (4096 - 4) \rceil = 2051$

Total Time (Linked Allocation)
= Directory access time + time for accessing the 2051 blocks
$= 10.4136 + 2051 \times 0.1 = 215.5136$ ms

FAT is faster.

**(c)** Determine the total time required to read only the 1200-th block of file A using (i) FAT and (ii) linked allocation (without FAT). Assume that the FAT (if applicable), the directory, and the FCB are all cached in the main memory. As in Part (b), the file name is first searched in the directory, then the corresponding directory entry is read, and the FCB is accessed. This is finally followed by the required number of disk-block reads. In both the cases, clearly show all the components of the access times, and identify which method is faster. **[3]**

(i) FAT Allocation (FAT cached):
Time for traversal of FAT in memory = 4800 bytes / $10^7$ bytes/s = 0.00048 sec = 0.48 ms
Time to read data block from disk = 0.1 ms
Directory access time = 10.4136 ms
Total time = 10.4136 ms + 0.48 ms + 0.1 ms = 10.9936 ms

(ii) Linked Allocation (without FAT):
Total time = 10.4136 ms + 1200 × 0.1 ms = 130.4136 ms

FAT is faster.

**5.** **[*Indexed allocation of files*]**

Consider a Unix file system organized using inodes. Each inode reserves the first 180 bytes for attributes, and subsequently contains direct block pointers along with one singly indirect pointer, one doubly indirect pointer, and one triply indirect pointer. Each disk block is of size 8 KB. The number of direct pointers is such that the entire inode block is fully utilized. The file system resides on a (magnetic) disk with 100 tracks, where each seek takes 20 ms per track movement. The disk-rotation speed is 3600 rpm, and each disk track holds 1 MB of data. All inodes are stored on track 0, and are loaded into the main memory before any data-block access.

**(a)** If the size of the disk-block pointer is 32 bits, calculate the maximum file size supported by this system. **[3]**

Inode size = 8 KB
Attributes = 180 bytes, indirect pointers = 12 bytes
Direct pointer space = 8192 – 192 = 8000 bytes
Number of direct pointers = 8000 / 4 = 2000

Number of pointers per indirect block = 8192 / 4 = 2048

Total number of blocks addressable = $2000 + 2048 + 2048^2 + 2048^3$ blocks

Approximately 64TB.

**(b)** Determine the number of disk accesses required to read the 34,000,000-th byte of a file. **[2]**

Block number = $\lfloor 34000000 / 8192 \rfloor = 4150$

Coverage
Direct pointers: 2000
Single indirect pointer: 2048
Total: 4048 < 4150
⇒ The requested block is in double indirect pointers

Accesses:
Double indirect: 2
Data block: 1
⇒ 3 disk access

**(c)** In this system, a file of size 100 KB is stored on the disk. Assume that after the first 180 bytes (equivalent to 45 block-pointer entries) reserved for the attributes, the $(45+x)$-th entry of the inode (that is, the $x$-th entry of the data block of the file) points to a data block located on track number $((7x+5)\%n)$ of the HDD if $x$ is even, and on track number $((x^2+2)\%n)$ if $x$ is odd, where $n$ = 100 (the tracks of the HDD are numbered $0-99$). If the SCAN disk-scheduling algorithm is used, compute the total time required to read the entire file, assuming that the file is already open. The disk head is initially positioned at track 80, and is moving toward higher-numbered tracks. **[7]**

File size = 100 KB ⇒ 13 blocks

Tracks: [5,3,19,11,33,27,47,51,61,83,75,23,89]

SCAN order (from 80 → up → down):
83, 89, 99, 75, 61, 51, 47, 33, 27, 23, 19, 11, 5, 3

Total movement: 115 tracks

Seek time: 115 × 20 = 2300ms

Disk rotation: 3600 rpm = 60 rps
Time per rotation: 16.67 ms
Rotational latency: 13 × 8.33 = 108.3 ms

Data Transfer Time:
Each track holds = 1 MB, Block size = 8 KB
Blocks per track = 1MB / 8KB = 128
Transfer time per block: 16.67/128=0.13ms
For 13 blocks: 13 × 0.13=1.7ms

Total file read Time = 2300 + 108.3 + 1.7 = 2.41 sec

## 6. [*Free-block management*]

Consider two file systems called CIC and CSE implemented on two identical disk partitions of size 16 TB each, and both implement the contiguous block-allocation method for file storage, but with different block sizes and different free-space managers. In the CIC file system, each disk block is of size 4 KB, and free space is managed using the bitmap method. In the CSE file system, each disk block is of size 64 KB, and free space is managed using the grouping method (multiple free-block pointers per block). Assume that the entire disk partition (16 TB) of each file system is available only for file storage. The free-space managers reside on a different partition, and require storage in both the cases. In the CIC system, the size of the bitmap is constant, whereas in the CSE system, the size of the grouping information depends on the number of free blocks. Users want to store video files each of the fixed size 64 MB, on both the file systems. Calculate the number of video files that will be stored in the CSE file system such that the total space consumed by the bitmap in the CIC system becomes equal to the total space consumed by the grouping method in the CSE system. Clearly show all the intermediate steps. **[10]**

Disk size: $2^{44}$ bytes

CIC: Total number of blocks = $2^{44} / 2^{12} = 2^{32}$
Bitmap size = $2^{32}$ bits = $2^{29}$ bytes

CSE : Total number of blocks = $2^{44} / 2^{16} = 2^{28}$
Number of pointers per block = 64KB / 4 = $2^{14}$

Let the number of free blocks be $F$ on the CSE system.
Grouping needs $F / 2^{14}$ blocks
Space used = $F / 2^{14} \times 2\text{^}16 = 4F$ bytes

$\Rightarrow 4F = 2^{29}$
$\Rightarrow F = 2^{27}$ free blocks on CSE

Number of occupied blocks = $2^{28} - 2^{27} = 2^{27}$
Size of each file = 64 MB = $2^{26}$ bytes
Number of blocks per file = $2^{26} / 2^{16} = 2^{10}$

$\Rightarrow$ Number of files on CSE = $2^{27} / 2^{10} = 2^{17}$

**7. [*Storage devices*]**

**(a)** Consider a 640 KB (magnetic) HDD with a single platter. The file system on the HDD implements contiguous allocation of file storage with extents (that is, if one contiguous chunk is insufficient to accommodate a file completely, it continues to another contiguous chunk). The disk has 256 sectors per track, and a total of 5 tracks T0, T1, T2, T3, and T4. Each disk block consists of 2 consecutive sectors. Blocks are laid out sequentially on the track (in the sequence block 0, block 1, and so on) starting from track 0, and progressing towards higher-numbered tracks. Files are allocated using contiguous allocation starting with the lowest block number, using the worst-fit strategy. The file manager tries to fit each file within a single track, and avoids spanning one file across multiple tracks (unless it is forced to use multiple extents). Initially, the disk is empty. A process creates the following files in sequence: file A of size 60 blocks, file B of size 40 blocks, file C of size 85 blocks, file D of size 90 blocks, file E of size 100 blocks, and file F of size 70 blocks. Next, the process deletes file B, and then it creates a file M of size 65 blocks. Show the allocation of files in different tracks and blocks after these eight file operations. Show your calculations, and finally present the track contents (including free blocks) in the table below. **[5]**

| Track | Block range (starting block – ending block) | Content (file or free space) |
|---|---|---|
| T0 | 0 – 59<br>60 – 124<br>125 – 127 | File A<br>File M<br>Free |
| T1 | 128 – 212<br>2133 – 255 | File C<br>Empty |
| T2 | 256 – 345<br>346 – 383 | File D<br>Free |
| T3 | 384 – 483<br>484 – 511 | File E<br>Free |
| T4 | 512 – 581<br>582 – 639 | File F<br>Free |

**Note:** There may be multiple entries against each track.

**(b)** Now, the file M of Part (a) grows by an additional 50 blocks (at the end, and in one single go). If sufficient contiguous space is not available adjacent to its current allocation of M, the system implements extent-based contiguous allocation (with worst-fit strategy). However, it always prefers the minimum number of extents. Show the allocation of the enlarged file M in different tracks and blocks. Clearly write the track and the block numbers of only the enlarged file M. **[3]**

Step 1: Use adjacent space

T0: 60 – 124 → 60 – 127, (+3 blocks)
Remaining: 50 – 3 = 47 blocks

Step 2: Allocate new extent (worst-fit, single extent)
T4: 582 – 628
Location of M

T0: 60 – 124 → 60 – 127, (+3 blocks)
T4: 582 – 628

Extra page for leftover answers (if any) and/or rough work

Extra page for leftover answers (if any) and/or rough work

Extra page for rough work

Extra page for rough work

Extra page for rough work

Extra page for rough work