Date
Page

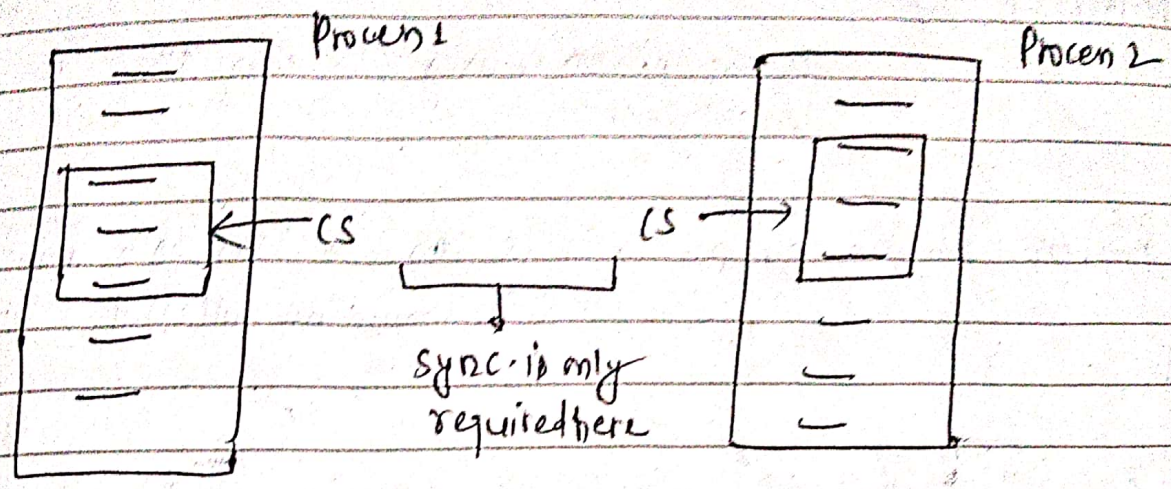
Threads vs Processes:

Process	Thread
1. Program in execution.	1. Path of execution within a process.
2. Processes are used to execute large heavyweight jobs.	2. Threads are used to execute smaller lightweight jobs.
3. Each process runs in a separate address space in CPU.	3. Threads share address space with other threads within the same process.
4. Process must use Interprocess Communication to communicate with sibling processes.	4. Threads can communicate directly with other threads of its process.
5. Processes only have control over child processes.	5. Threads have greater control over other threads of same process.

Scanned with CamScanner

Critical Section: The part of program where shared memory is accessed.

The critical section is the code segment where the shared variables can be accessed.



Where synchronization required? → only in critical section

Race condition: → caused due to lack of synchronization.

A race condition is an undesirable situation, which occurs when the final result of the concurrent processes depends on the sequence in which they complete their execution.

Process P_1

$R_1 = a$

$R_1 = R_1 + 2$

$a = R_1$

Process P_2

$R_3 = a$

$R_3 = R_3 + 5$

$a = R_3$

Scanned with CamScanner

Requirements of critical section problem solution:

Any solutions to the critical section problem should meet following requirements

(1) Mutual Exclusion:

If one process is using C.S. then other process cannot use that C.S.

(2) Progress: If no any process in CS and at least one process wants to enter in C.S., then it should be allowed.

(3) Bounded waiting → waiting of processes should be bounded (waiting should not be infinite).

Software solutions:

Solution 1: Using lock:

Boolean lock = false

P_0	P_1
while (true) {	while (true) {
while (lock);	while (lock);
lock = true;	lock = true;
CS	CS
lock = false;	lock = false
Remainder section	RS
}	}

Scanned with CamScanner

- Doesn't satify mutual exclusion.

↳ If a process preempted after while (lock); state.

→ satisfy progress.

Solution 2: Using turn:

```
int turn = 0;

P0                                  P1
while (true) {                      while (true) {
                                      while (turn != 1);
  while (turn != 0);                    CS;
    CS                                  turn = 0;
    turn = 1;
                                        RS;
  RS;                               }
}
```

→ Mutual exclusion is satisfied.

→ Progress is not satisfied. Let's turn=0, but P0 is not interested to go in critical section, P1 cannot go to critical section neither.

2 Process will execute only in strict alternate manner.

Scanned with CamScanner

Date
Page

Solution: 3 Peterson's Solution:

Boolean flag[2] → flg[0] = false
flg[0] = false

int turn;

```
while (true) {

    Flag[0] = true;
    turn = 1;
    while (flag[1] && turn = 1);

    CS
    Flag[0] = false;
    RS;
}
```

Producer

```
while(true) {

    Flag[1] = true;
    turn = 0;
    while( flag[0] &&
        turn == 0;

    CS
    flag[1] = false
    RS;
}
```

Consumer

Provides all 3 requirements satisfied.

Scanned with CamScanner

Date
Page

Hardware Solutions [Synchronization Hardware]

↓

instruction support by CPU.

↓

can be used to provide synchronization

↓

privileged instruction

(1) Test and set
(2) swap

Test and set:

returns the current value flag and sets it to true

Scanned with CamScanner

Synchronization tools:

(1) Semaphore
(2) Monitor

(1) Semaphore:

→ integer value which can be accessed using following functions only:

- wait() / P() / Degrade()
- signal() / V() / Upgrade.

atomic

```
wait (s) {                    signal(s) {

    while (s<=0);                 s++;
    s--;
}                             }
```

Types of semaphore:

1. Binary semaphore : only 2 values 0 or 1 (mutex lock)
2. Counting semaphore : unlimited domain of values

Scanned with CamScanner

Critical section solution:

```
S = 1

while (True) {

    wait (s); → atomic function (Both lines or no lines)
      C.S
    Signal (s);

}
```

(if बिचमा रोकिए भने revert हुन्छ)

Characteristics of Semaphores:

1. Used to provide mutual exclusion → Binary
2. Used to control access to resources → counting
3. Solution using semaphore can lead to have deadlock.
4. Solution using semaphore can lead to have starvation → indefinite waiting
5. Solution using semaphore can be busy-waiting solutions.
6. Semaphores may lead to priority inversion.
7. Semaphores are machine-independent.

Scanned with CamScanner

Synchronization Hardware:

Test and set lock:

- A hardware solution to synchronization problem.
- There is a shared lock variable which can ~~either~~ take value 0 or 1.
- Before entering into the critical section, a process inquires about lock.
- If it is locked (lock = 1), it keeps on waiting till it becomes free.
- If it is not locked, it takes the lock and executes critical section.

```
do {
  Acquire lock
  critical section
   Release lock
  remainder section.
}while (TRUE);
```

```
enter-region

TSL REGISTER, LOCK    // copy lock to register & set lock 1
CMP REGISTER, #0      // was lock zero?
JNE enter_region      // if not zero, lock was set so loop
RET                   // return to caller; critical region entered
```

```
leave-Region
MOVE LOCK, #0   /store a 0 in lock
RET             /return to caller
```

Scanned with CamScanner

Semaphore

Producer Consumer problem solution :

Producer :

```
do {
    : PRODUCE ITEM

    wait (empty);
    wait (mutex);

    : PUT ITEM IN BUFFER

    signal (mutex);
    signal (full);

}while(1);
```

Consumer

```
do {
    wait (full);
    wait (mutex);

    : REMOVE item from Buffer

    signal (mutex);
    signal (empty);

    : CONSUME ITEM

}while(1);
```

mutex, empty, full are semaphores. Here mutex is initialized to 1, empty is initialized to n (maximum size of buffer) and full is initialized to 0.

mutex → ensures mutual exclusion

Empty and full semaphores count the number of empty and full spaces in the buffer.

Scanned with CamScanner

Producer :

After an item is produced, wait operation is carried out on empty. This indicates that empty space in buffer has decreased by 1. The wait operation is carried out on mutex so that consumer process couldnot interfere.

After an item is put in the buffer, signal operation is carried on mutex and full. The former indicates that consumer process can now act and the latter shows that the buffer is full by 1.

Consumer :

The wait operation is carried out on full. This indicates that items in the buffer have decreased by 1. Then wait operation is carried out on mutex so that produce cannot interfere.

Then the item is removed from buffer. After that, signal operation is carried out on the mutex and empty. The former indicates that consumer process can now act and the latter shows that the empty space in buffer has increased by 1.

Scanned with CamScanner

Banker's Algorithm → Deadlock Avoidance:

Process	Allocation			MAX			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			
	7	2	5						

(1) Need matrix Need [i, j] = max [i, j] − Allocation [i, j]

Need:

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1.
P_4	4	3	1.

(2) Safe Sequence

With available A = 3, B = 3, C = 2 need of either P_1 or P_3 can be fulfilled.

let allocate resources to P_1. When P_1 is done executing, it will release the resources.

Scanned with CamScanner

2	0	0	2 1 0
+1	+2	+2	5 3 2
3	2	2	

	A	B	C
Available [Available + Allocation] =	5	3	2

1	Available		
	A	B	C
	3	3	2
After P_1	5	3	2
After P_3	7	4	3
After P_0	7	5	3
After P_2	10	5	5
After P_4	10	5	7

Scanned with CamScanner

(Q)

Processes	Allocation A B C D	Max A B C D	Available A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	2 9 10 12
P_4	0 0 1 4	0 6 5 6	3 14 12 12
	2 9 10 12		

(1)

Need :

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

(2)

	Available A	B	C	D
	1	5	2	0
After P_0	1	5	3	2
After P_3	1	11	5	4
After P_1	2	11	6	4
After P_2	3	14	12	8
After P_4	3	14	12	12

Scanned with CamScanner

Classical Synchronization Problems:

(1) Dining - Philosophers Problem :

Statement:

Consider 5 philosophers who spend their lives thinking and eating. The philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher. In center of table, there is a bowl of rice, and the table is laid with five chopsticks. When a philosopher get hungry, she tries to pick up her left and right chopstick, one at a time, in either order. If successful in acquiring two chopsticks, she eats for while, then put down the chopsticks and continues to think. The key question: write a program for each philosopher that does what it supposed to do and never gets stuck.

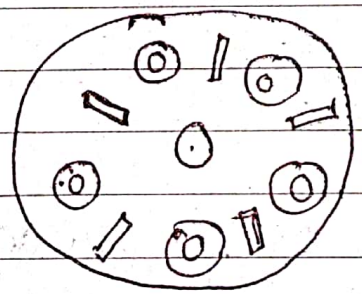


Fig. Dining Table of Philosophers

Solution :

One simple solution is to represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore, she releases her chopstick the signal() operation on the

appropriate semaphores. Thus, shared data are:

```
semaphore chopstick [5] ;
```

The structure of philosophers is show below;

```
do {
    wait (chopstick [i]) ;
    wait ( chopstick [(i+1) % 5]) ;

    - - -

    // eat

    signal (chopstick [i]) ;
    signal (chopstick [ (i+1) % 5]) ;

    // think

} while (true);
```

But if this solution is simply applied deadlock will occur.

Scanned with CamScanner

To avoid deadlock we can apply following solution:

→ Allow at most four philosophers to be sitting simultaneously at the table.

→ Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this she must pick them up in critical region].

→ Use an asymmetric solution, that is, an odd philosopher picks up first her left chopstick and her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Scanned with CamScanner

(2) Readers Writers Problems:

The dining philosopher's problem is useful for modeling processes that are competing for exclusive access to limited number of resources, such as I/O devices.

Another famous problem is Readers and writers problem which models access to a database.

statement:

Imagine, for example, an airline reservation system, with many competing processes wishing to read and write it. It is acceptable to have multiple processes reading the database at the same time but if one process is updating (writing) the database, no other process may have access to the database, not even a reader.

The question is how to program the readers and writers.

One solution is shown below:

```
typdef int semaphore;

semaphore mutex = 1;
semaphore db = 1;

int rc = 0; /* no. of processes reading or wanting to */
```

Scanned with CamScanner

```
void reader() {
    while (TRUE) {  /* repeat forever */
    down (&mutex) ; /* get exclusive access to 'rc' */
    rc = rc+1 ;
    if (rc ==1)  down (&db) ; /* if this first reader */
    up ( &mutex)
    read-data-base() ;
    down (&mutex) ;
      rc = rc-1;
      if (rc ==0) up (&db)  /* if this is last reader */
    up( &mutex)
      use-data-read() ; // Remainder section
    }
}
```

Scanned with CamScanner

```
Void writer () {
    while (TRUE) {
        think_up_data();  // Non critical */
        down(&db) ; /* get exclusive access */
        write_data_base(); /* update data */
        down (&db); /* release exclusive access */
    }
}
```

In this solution, first reader to get the access to the database does a on down on semaphore db. Subsequent readers merely have to increment the rc. As readers leave, they decrement the counter and the last one out does an up on the db, allowing a blocked writer, if there is one, to get in.

Suppose that while a reader is using the database, another reader comes along. Since having two readers at the same time it is not a problem, the second reader is admitted. A third and subsequent readers can also be admitted if they come along.

Now suppose that a writer comes along. The writer cannot be admitted to the database, since writers must have exclusive access, so the writer is suspended. later additional readers show up. As long as at least one reader is still active, subsequent readers are admitted. As a consequence of this strategy, as long as there is a steady supply of readers, they will get in as soon as they arrive. The ~~wait~~ writer will be kept suspended until no reader is present. If a new reader arrives, say, every 2 seconds, and each readers take 5 secs to do its work, the writer will never get in.

Scanned with CamScanner

→ To prevent this situation, the program could be written significantly differently: when a reader arrives and a writer is waiting, the reader is suspended behind the writer inside of being admitted immediately. In this way, a writer has to ~~write~~ wait for readers that were active when it arrived to finish but does not have to wait for readers that come along after it. The disadvantage of this solution is that it achieves less concurrency and thus lower performance.

(3) Sleeping Barber Problem :

Another classical IPC problem takes place in the barber shop. The barber shop has one ~~sho~~ barber, one barber chair and n chairs for waiting customers, if any, to sit on. If there are no customers present, the barber sits down in the barber chair and falls asleep. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customer's hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full). The problem is similar to various queuing situations like multiperson helpdesk with a computerised call waiting system for holding a limited number of incoming calls.

The solution uses three semaphores: customers → which counts waiting customers (excluding customer in the barber chair, who is not waiting), barbers → number of barbers (0 or 1) who are idle, waiting for customers and a mutex which is used for mutual exclusion.

We also need a variable, waiting, which counts the waiting customers.

Scanned with CamScanner

The reason for having waiting is that there is no way to read the current value of semaphore. In this solution, a customer ~~has~~ entering the shop has to count the number of waiting customers. ~~The reason for having~~ If it less than the number of chairs, he stays; otherwise, he leaves.

When the barber shows up for the work in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. The barber then goes to sleep. He sleeps until the first customer shows up.

When the customer arrives, he executes customer, starting by acquiring mutex to enter a critical section. If another customer enters shortly thereafter, the second one will not be able to do anything until the first one has released mutex. The customer then checks to see if number of waiting customers is less than number of chairs. If not, he releases mutex and leaves without a hair cut.

If there is an available chair, customer increments the integer variable, waiting. Then, he does an up on semaphore customers, thus waking up the barber. At this point, the customer and the barber are both awake. When the customers release mutex, the barber grabs it, does some housekeeping and begins haircut.

When the haircut is over, the customer exits the procedure and leaves the shop. Unlike previous examples, there is no loop for the customer because each one gets only one haircut. The barber loops, however, to try to get the next customer. If one is present, haircut is given; if not, the barber goes to sleep.

Scanned with CamScanner

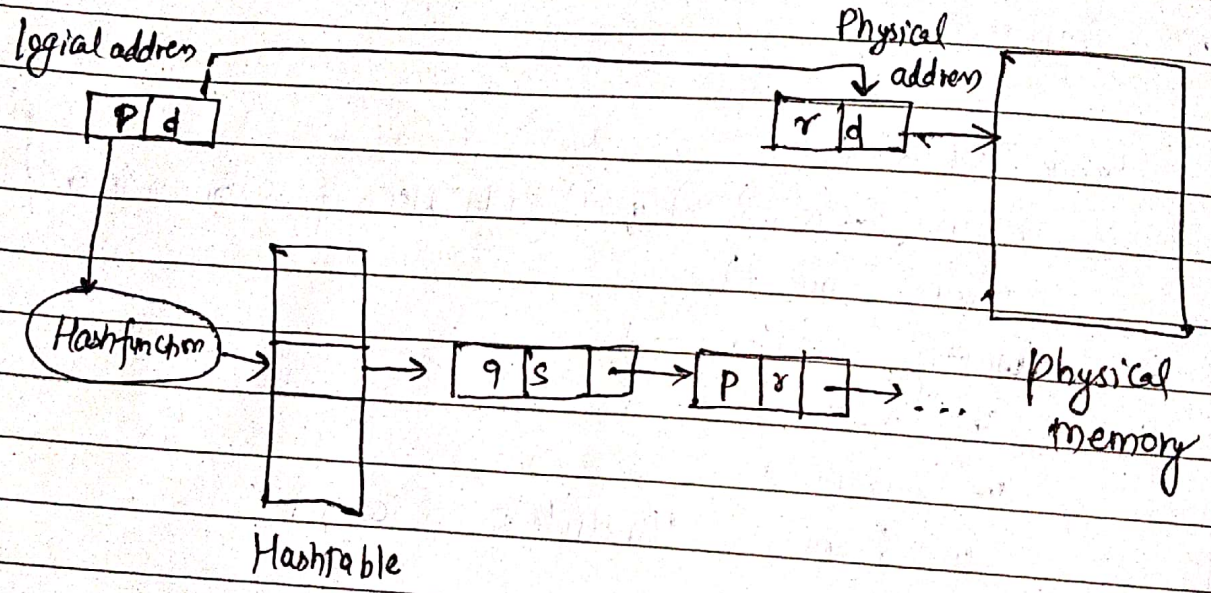
```
#define CHAIRS 5
typedef int semaphore;
semaphore customers = 0;
semaphore barbers = 0 ; /* no. of barbers waiting for customers */
semaphore mutex = 1 ;
int waiting = 0 ; /* customers are waiting */

void barber {

    while (TRUE) {
        down(&customers) ; /* go to sleep if customers is 0 */
        down(&mutex) ;
        waiting = waiting - 1 ;
        up(&barbers);
        up(&mutex)
        cut_hair() ;   //outside critical section
    }
}


void customer {
    down(&mutex) ;
    if (waiting < CHAIRS) {
            waiting = waiting + 1 ;
            up(&customers) ;
            up(&mutex);
            down(&barbers);
            get_haircut() ;
    } else {
            up(&mutex) ;
    }
```

Scanned with CamScanner

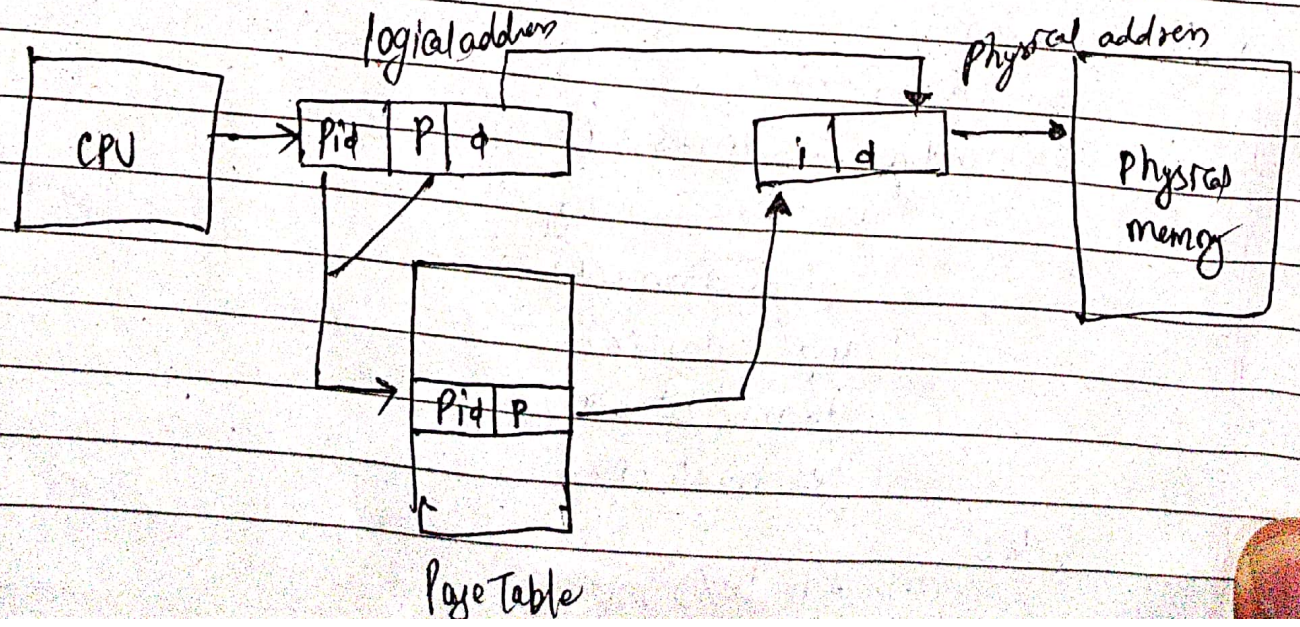
Page Table Structure:

(1) Hierarchial Paging
(2) Hashed Page Tables
(3) Inverted Page Tables

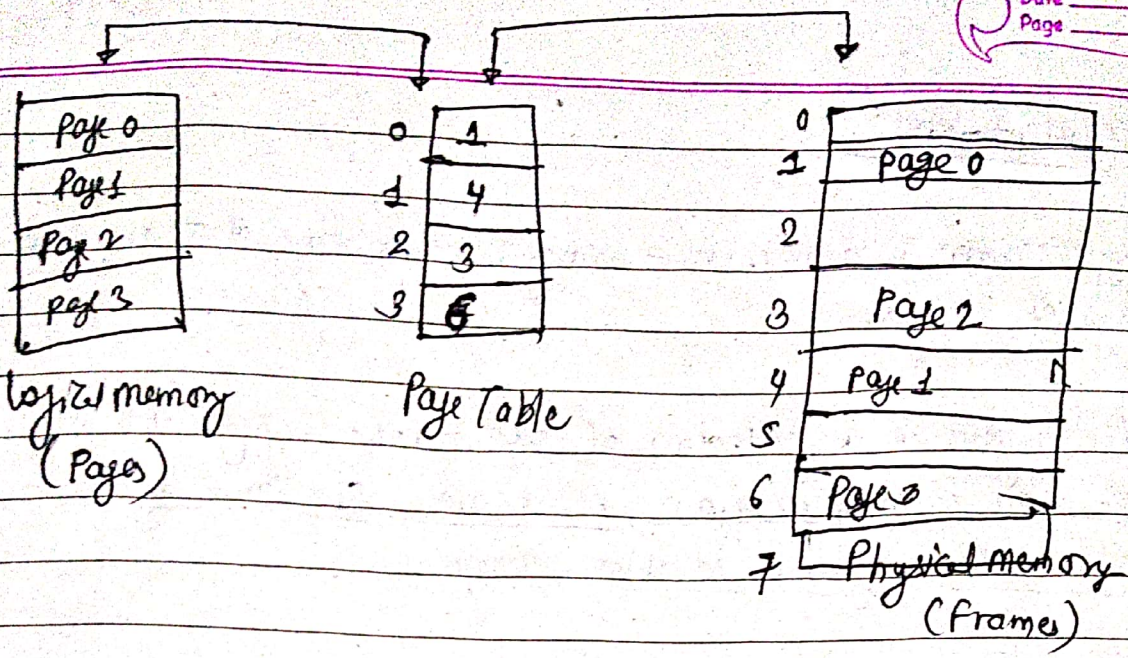
Hashed Page Table :



Inverted Page Table :



Scanned with CamScanner



TLB (Translation look Aside Buffer) :

A TLB is a memory cache that stores recent translations of Virtual memory to physical addresses for faster retrieval.

Each entry in TLB consists of two parts : a key and a value.

Scanned with CamScanner

Segmentation:

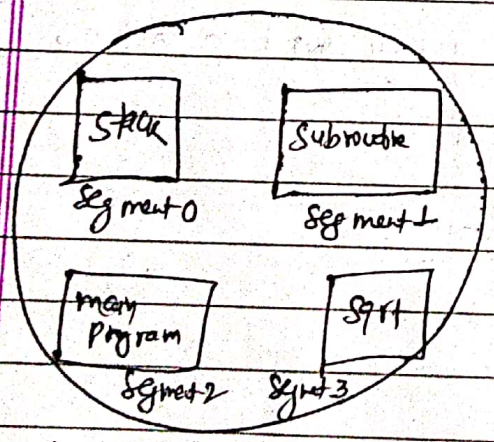
→ Segmentation is a memory-management scheme that supports user view of memory.

→ A program is a collection of segments. A segment is a logical unit such as main program, procedure, function, method, object, local variables, global variables, arrays.

→ logical address consists of a two tuple:

<segment-number, offset>

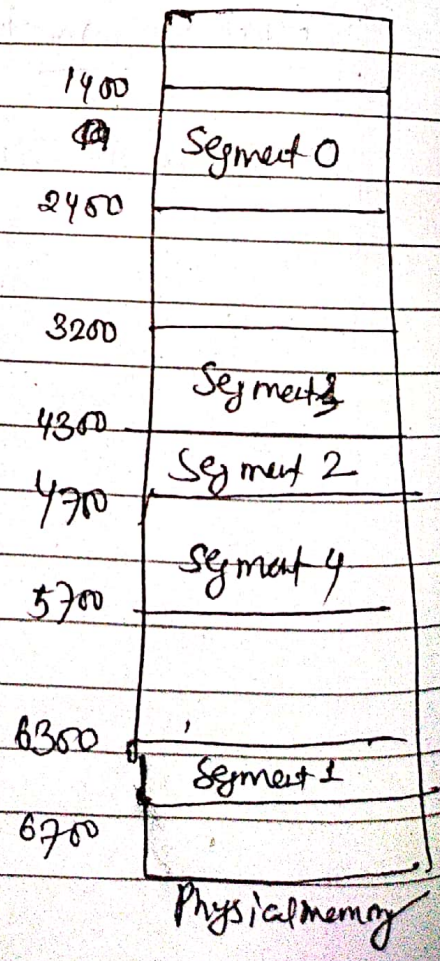
→ Segment table maps each segment to physical memory.



logical address space

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment table



Physical memory

Segment table consists of:

→ base : contains the starting physical address where segment lies inside memory

→ limit : length of segment.

Segmentation WITH Paging ??

Scanned with CamScanner

Place Replacement :

(A) FIFO :

7	0	1	2	0	3	0
7	7	7	2	2	2	2
	0	0	0	0	3	3
		1	1	1	1	0
✓	✓	✓	✓	✗	✓	✓

4	2	3	0	3	2	1
4	4	4	0	0	0	0
3	2	2	2	2	2	1
0	0	3	3	3	3	3
✓	✗	✓	✓	✗	✗	✓

2	0	1	7	0	1
0	0	0	7	7	7
1	1	1	1	0	0
2	2	2	2	2	1
✓	✗	✗	✓	✓	✓

No. of page faults = 14.

Scanned with CamScanner

Belady's (or FIFO) Anomaly:

- Certain page reference patterns actually cause more page faults

Optimal Replacement:

→ Replace those pages which will not referenced immediately in near future.

LFU: (least frequently used)

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 Frame size=3

2	3	2	1	5	2	4	5
2	2	2	2	2	2	2	2
	3	3	3	5	5	5	5
			1	1	1	4	4
✓	✓	X	✓	✓	X	✓	X

3	2	5	2
2	2	2	2
5	5	5	5
3	3	3	3
✓	X	X	X

No. of page faults = 6

Scanned with CamScanner

Date
Page

Optimal :

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

2	3	2	1	5	2	4	5
2	2	2	2	2	2	4	4
	3	3	3	3	3	3	3
			1	5	5	5	5
✓	✓	X	✓	✓	X	✓	X

3	2	5	2
4	4	4	4
3	2	2	2
5	5	5	5
X	✓	X	X

No. of page faults = 6

Second chance :

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

2	3	2	1	5	2	4	5
0 2	0 2	1 2	1 2	0 2	1 2	0 2	0 2
	0 3	0 3	0 3	0 5	0 5	0 5	1 5
			0 1	0 1	0 1	0 4	0 4
✓	✓	X	✓	✓	X	✓	X

3	2	5	2
0 3	0 3	0 3	0 3
1 5	0 5	1 5	1 5
0 4	0 2	0 2	1 2
✓	✓	X	X

No. of page faults = 7

Scanned with CamScanner

Date
Page

Chapter-4 I/O management :

(1) Tertiary storage Devices

(2) ~~map~~ memory mapped I/O to device control registers

(3) Bad sector . Bad Block recovery mechanism.

(4) Swapspace and ~~s~~ swapspace management.

(5) Different I/O techniques.

(6) What are different ways to input/output? Explain Interrupt-Driven I/O with diagram.

(7) How data is managed and maintained using RAID technology ? Is there any chance of losing data using it? Describe each of available model in details.

~~Interrupt~~

Ways of I/O :

(1) Programmed I/O
(2) Interrupt Driven
(3) DMA (Direct memory Access)

(1) Programmed I/O :

→ simplest form of I/O where CPU do all of the work.

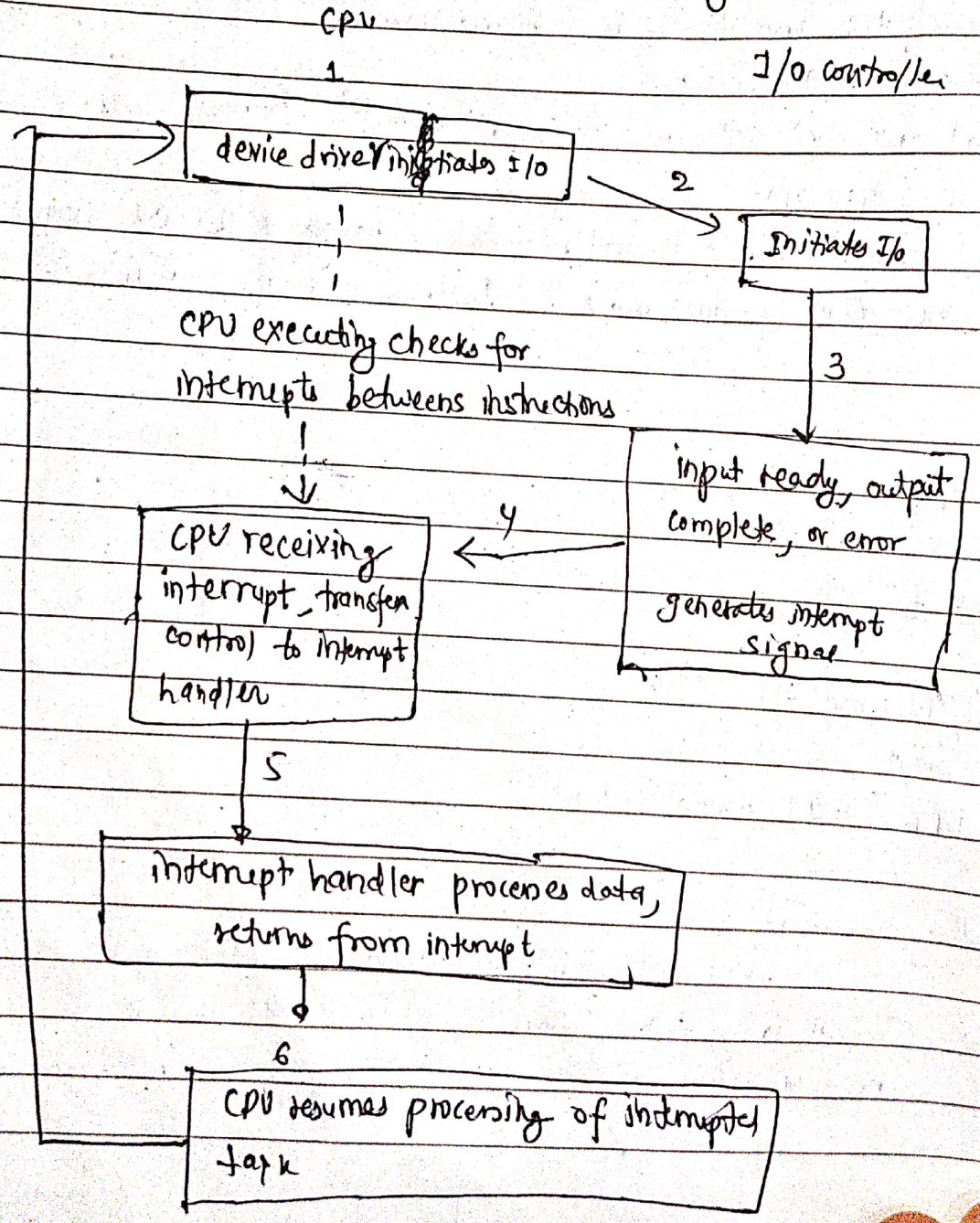
→ CPU continuously polls the device to see if the device is ready to accept or requesting data. This behavior is known as polling or busy waiting.

Scanned with CamScanner

(2) Interrupt Driven: I/O :

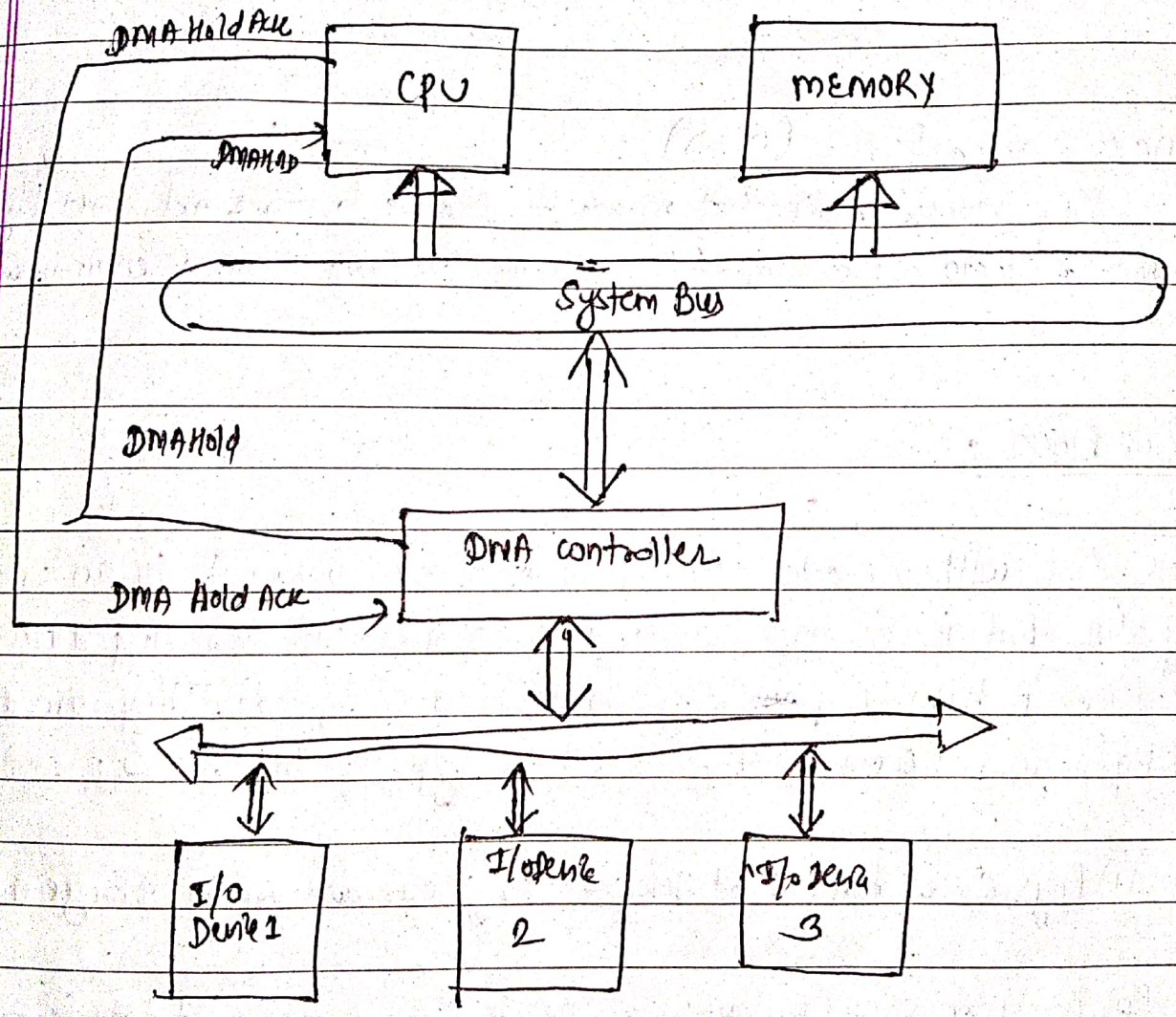
The CPU hardware has a wire called the interrupt request line that CPU senses after executing every instruction.

When CPU detects a request signal on the line, CPU saves a small amount of state such as current value of instruction pointer and jumps to the interrupt handler routine at fixed address in memory.



③ DMA :

Many computers avoid the burdening main CPU with Programmed I/O by offloading some of the work to special purpose processor called direct-memory-access (DMA) controller. To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of transfer and a count of number of bytes to be transferred.



Scanned with CamScanner

Kernel I/O Subsystem :

Kernel provide many services related to I/O. They are:

(1) I/O scheduling
(2) Buffering → store data in memory while transfering between devices
(3) Caching
(4) Spooling → hold output for a device
(5) Device reservation → provides exclusive access to a device.
(6) Error Handling

Mass Storage Device (MSD) :

Any storage device that makes it possible to store and port large amount of data across computers, servers and within an IT environment.

Bad Blocks :

The disk with defected sector is called as bad block. It is an area of disk that is no longer reliable for storing and retrieving data because it has been ~~physically~~ damaged or corrupted. They are also known as bad sectors. There are two types of Bad blocks:

(1) Physical or Hard Bad Block = physical damage to storage media

(2) logical or Soft Bad Block : OS is unable to read data from a sector.

Scanned with CamScanner

ECC → Error correcting Codes

Date
Page

Method-1 : "Handled manually".

If blocks go bad during normal operation, a special program must be run manually to search the bad blocks and to lock them away so that file system does not allocate them. Data that resided on the bad blocks usually are lost.

Method-2 : Sector Sparing / forwarding:

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of spare sectors. This scheme is known as Sector sparing / forwarding.

A typical bad sector transaction might be as follows:

→ The operating system tries to read logical block 80.

→ The controller calculates ECC and finds that the sector is bad.

→ It reports the finding to operating system.

→ The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.

→ After that, whenever the system requests logical block 80, the request is translated into the replacement sector's address by the controller.

Scanned with CamScanner

Date
Page

Method 3 – Sector slipping:

The sector slipping method preserves the sequential ordering of data.

For example: Suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, the sector slipping would map the sectors from 17 to 202, moving them all down one spot. That is sector 202 would be copied into spare, then sector 201 into 202 and then 200 into 201 so on, until sector 18 into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped into it.

Scanned with CamScanner

Swap Space Management :

~~Swap space is a virtual memory~~

Swap space management is one of the task of operating system in which disk space is used as an extension of main memory by the virtual memory. As we know that disk access is much slower than memory access, in swap-space management we are using disk space so it will significantly decreases system performance. Basically, in all our systems, we require the best throughout, so that goal of swap-space implementation is to provide the virtual memory the best throughput.

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as if it were of the part of main memory. Virtual memory is implemented using Demand Paging, which is the process of loading the page into memory on demand (whenever page faults occur).

Swapping is a memory management technique in which a process is removed from main memory (swap out) and stored it into secondary memory, and then bringing back into main memory (swap in).

Swap space is the area on the disk where the swapped-out processes are stored.

The need of swapspace can vary from a megabyte to gigabyte but it also depends upon the amount of physical memory, virtual memory it is backing and the way in which it is using virtual memory.

Scanned with CamScanner

It is safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort the process or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it doesnot harm other.

Swap space resides in

- Normal file System
- Separate Disk Partitioning

Scanned with CamScanner

Tertiary Storage Devices :

→ A tertiary storage system typically refers to a data storage system that uses drives that accept removable media, a storage rack for removable media, and a robot arm to transfer media between storage rack and the drives.

1. Floppy Disk
2. magnetooptic Disk
3. optical Disk
4. WORM (Write once Read Many Times) Disk
5. Tapes

RAID:

Disk drivers have continued to get smaller and cheaper, so it is now economically feasible to attach a large number of disks to a computer system.

Having a large number of disks in a system presents opportunities for improving the rate at which data can be read or written, if the disks are operated in parallel.

Having a large number of disks in a system presents opportunities for improving the rate at which the data can be read or written, if the disks

Scanned with CamScanner

are operated in parallel.

Furthermore, this setup offers the potential for improving the reliability of data storage, because redundant information can be stored on multiple disks. Thus, failure of one disk doesnot lead to loss of data.

A variety of disk organization techniques, collectively called redundant arrays of ~~inexpensive~~ independent disks (RAID), are commonly used to address the performance and reliability issues.

RAID levels:

(1) RAID level 0 – Stripping:

In a RAID 0 system, data are split up into the blocks that get written across all the drives in the array. By using multiple disks (at least) at the same time, this offers superior I/O performance.

Advantage:

→ great performance.
→ All storage capacity is used
→ Easy to implement

Disadvantage:

No fault tolerant. If one drive fails, data is lost as it not redundant.

Scanned with CamScanner

(2) RAID level 1 — Mirroring

Data are stored twice by writing them to both the data drive and a mirror drive.

Advantage:

- Excellent read and write speed
- → Fault tolerant
- → Simple

Disadvantage:

- → Effective storage capacity is only half used.

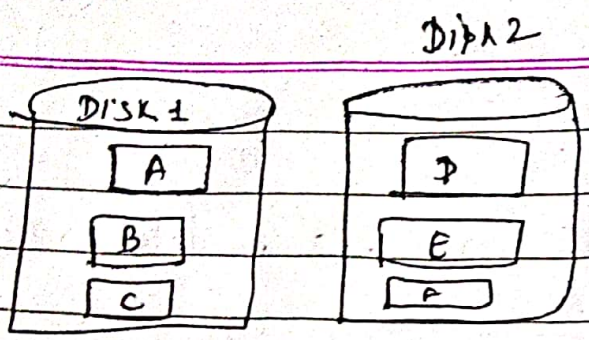
(3) RAID level 5:

- → most common secure RAID level.
- → Requires at least 3 drives but can work with upto 16.
- → Data blocks are stripped across the drive and on one drive a parity checksum of all block data is written.

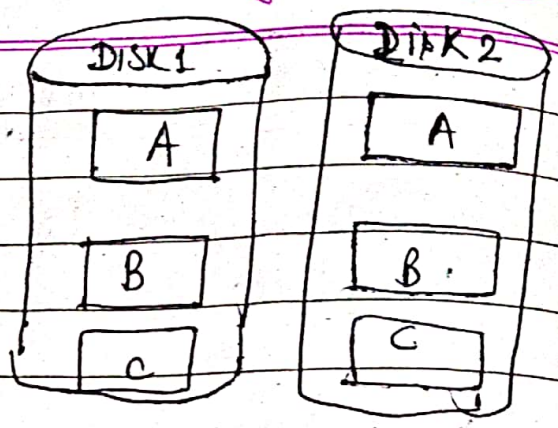
The parity data are not written to a fixed drive, they are spread across all drives. Using the parity data, computer can recalculate the data of one of the other data blocks, if those data are no longer available.

That means a RAID 5 array can withstand a single drive failure without losing or access to data.

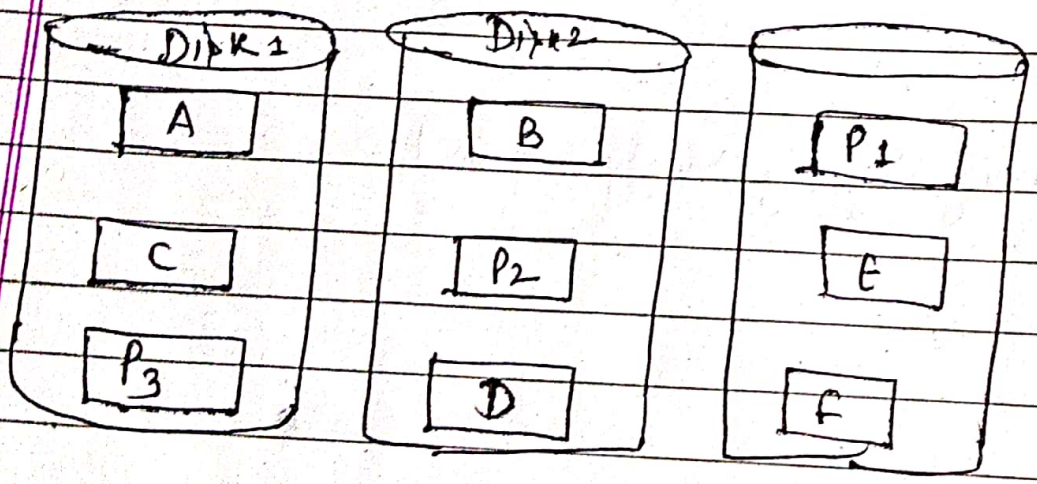
Scanned with CamScanner



Raid 0: Blocks striped. No mirror. No Parity



RAID 1: Blocks mirrored. No Stripes. No parity.



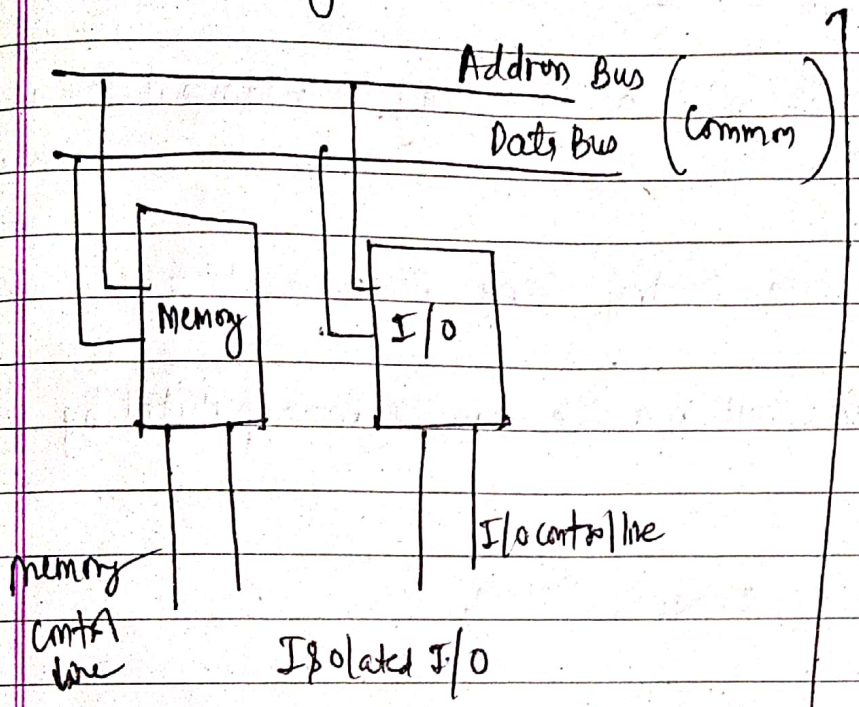
RAID 5 : Blocks Striped . Distributed Parity

Scanned with CamScanner

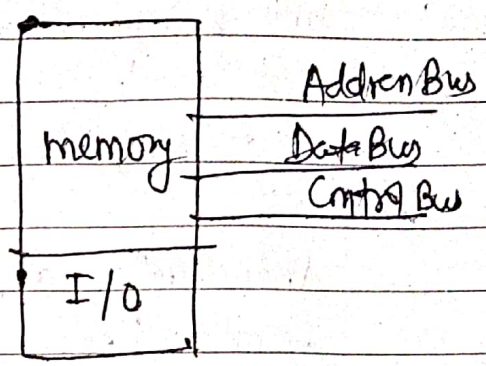
Q) What are the advantages and disadvantages of supporting memory mapped I/O to device control registers?

Isolated I/O = ~~where~~ have common bus (data and address) for I/O and memory but seperate read and write control lines for I/O & memory.

Memory mapped I/O → have common data, address bus & control lines for I/O and memory.



Isolated I/O



Memory mapped I/O.

→ Memory and I/O have separate address space.	→ Memory & I/O have same address space.
→ More efficient due to seperate buses.	→ less efficient.
→ larger in size due to more buses.	→ smaller in size.

Scanned with CamScanner

Date
Page

Disk Interleaving:

Interleaving is a technique used to improve slow system performance by putting data accessed sequentially into non-sequential blocks, typically sector.

Interleaving was used to arrange the sectors most efficiently.

Chapter-5 File System:

File: a named collection of data that may be manipulated as a unit by operations such as:

- open, close, create, Destroy, copy, Rename, List

Individual data items within a file may be manipulated by operations like:

- Read, write, Update, Insert, Delete

file characteristics include: Location, Accessibility, Type, volatility.

Scanned with CamScanner

File Descriptor | File Control Block:

→ control block containing information the system needs to manage a file. It is highly system-dependent structure.

A typical file descriptor might include :

- symbolic file name
- location of file in secondary storage
- file organization
- access control data
- type disposition
- creation date and time

File Access Methods :

① Sequential Access
② Direct Access

① Sequential Access :

The simplest access method. Information in file is processed in order, one record after the another. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

read next
write next
Reset
no read after last write
(rewrite)

Scanned with CamScanner

② **Direct Access** : Another method is direct access or relative access.

A file is made up of fixed-length logical records that allows program to read and write records rapidly in no particular order.

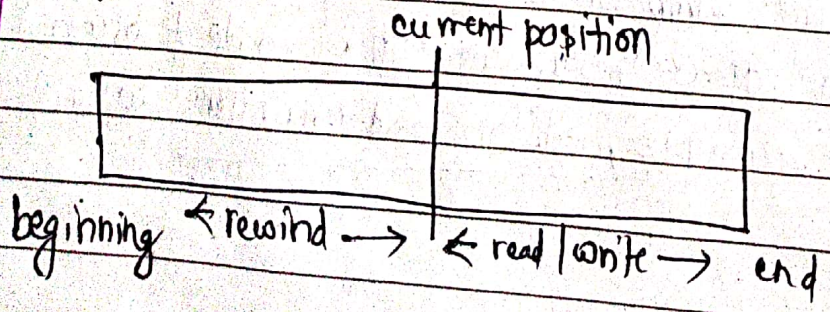
The direct access metho is based on a disk model of file, since disk allow random access to any file block.

For direct access, file is viewed as a numbered sequence of blocks | records. A direct access allows arbitary blocks to be read or written.

read n
write n
position to n
    read next
    write next
rewrite n

n = relative block number

Sequential :



Scanned with CamScanner

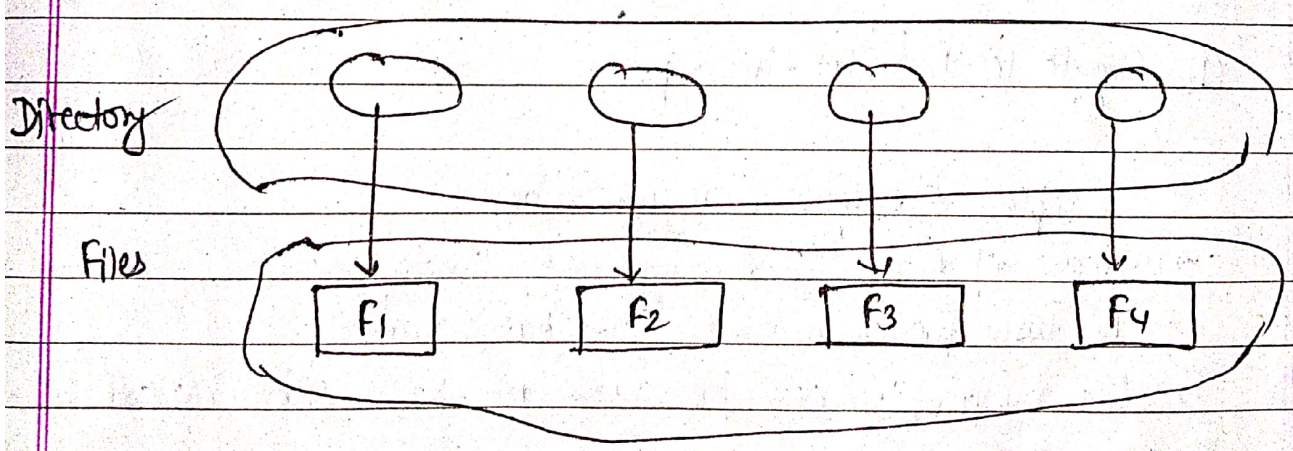
Directory Structure:

Directories:

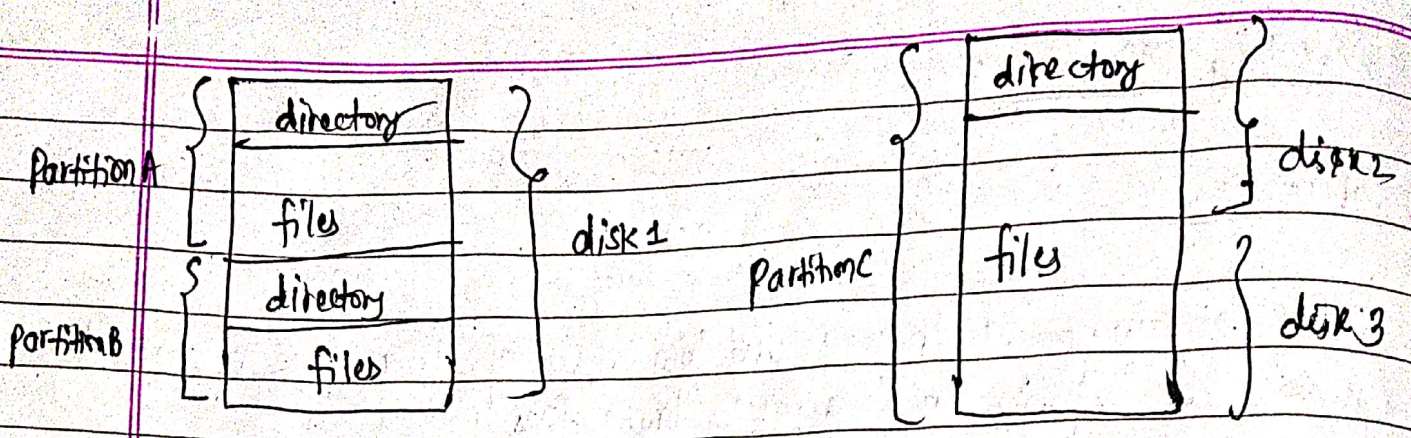
- File containing the names and locations of other files in the file system, to organize and quickly locate files.

- Directory entry stores information such as:
  File name.
  location
  Size
  Type
  Accessed
  modified and creation times

Both the directory structure and the files reside on the disk. Backups of these two structures are kept on tapes.



Scanned with CamScanner



Directory options :

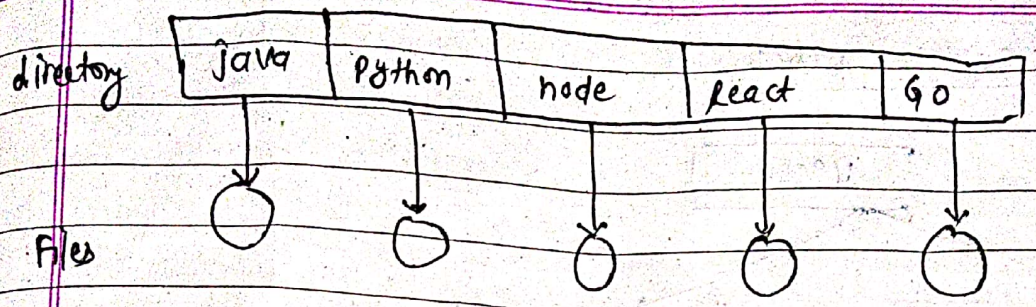
→ Create a file
→ Delete a file
→ List a directory = Contents of directory
→ Rename a file
→ Traverse file system.

Most common schemes for defining logical structure of a directory are:

① Single level | Flat directory:

→ Simplest file system organization
→ Stores all of its files using one directory
→ No two files can have the same name.
→ file system contents to locate each file, which can lead to poor performance.

Scanned with CamScanner



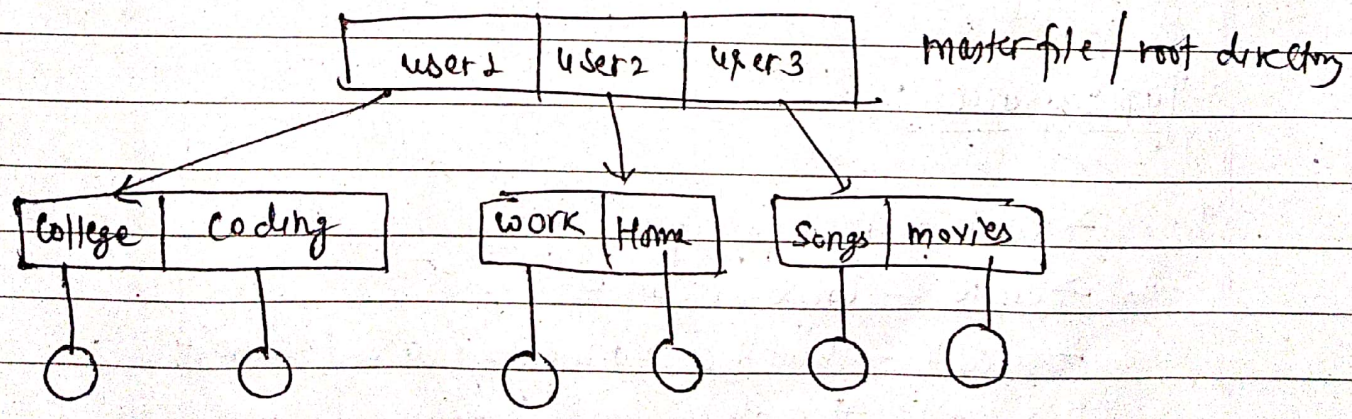
Hierarchial :

→ A root indicates where on the storage device the root directory begins.

→ The root directory points to various directories, each of which contains an entry for each of its files.

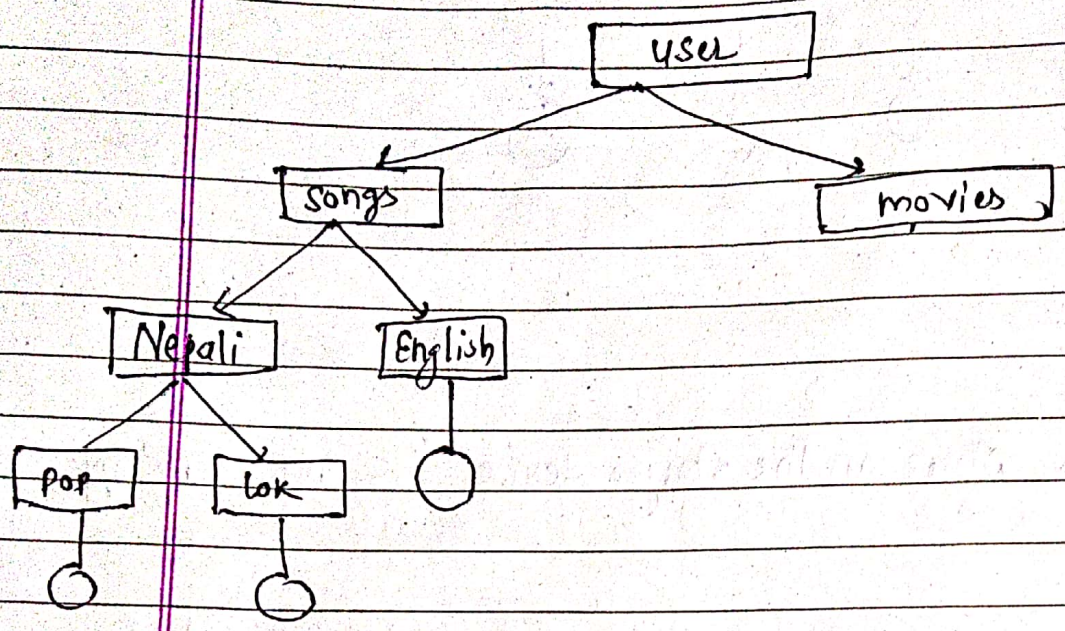
→ File names must be unique only within a given directory.

→ The name of file is usually ~~found~~ formed as the pathname from the root directory to the file.



Scanned with CamScanner

Two level directory structure:



Protection :

~~When informator~~

File owner | creator should be able to control:

1. what can be done
2. By whom

Types of access :

→ Read – read from file
→ write – write or rewrite
→ Execute – load file into memory and execute it
→ Append → write at end of file
→ Delete
→ list

Scanned with CamScanner

Access Lists and Groups:

Mode of access: read, write, execute

Three classes of users: RWX – RWX – RWX

owner group public

Access Control List (ACL):

ACL is a list of rules that specifies which users/groups are granted or denied access to a particular resource (files).

When a user requests to a particular file, the OS checks the ACL associated with that file. If the user is listed for request access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

Here, we see three processes, each belong to different domain: A, B and C and three files f_1, f_2, f_3. We assume each domain is owned by exactly one user. Often in the security literature, the users are called subject or principals, to contrast them with the things owned, the objects such as files.

Scanned with CamScanner

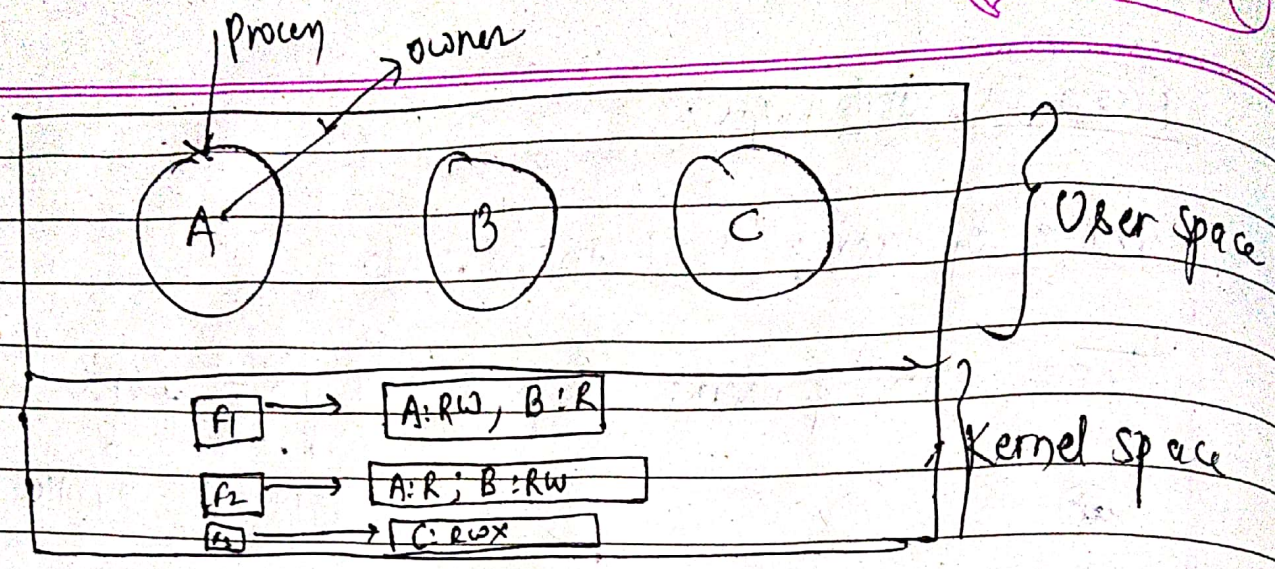


fig: Use of access control list to manage the files.

→ Describe the figure. f1 has RW permission for A, and so on.

File System Structure :

Methods of Allocation :

→ An allocation method refers to how disk blocks are allocated for files:

(1) Contiguous allocation
(2) linked allocation
(3) Indexed allocation

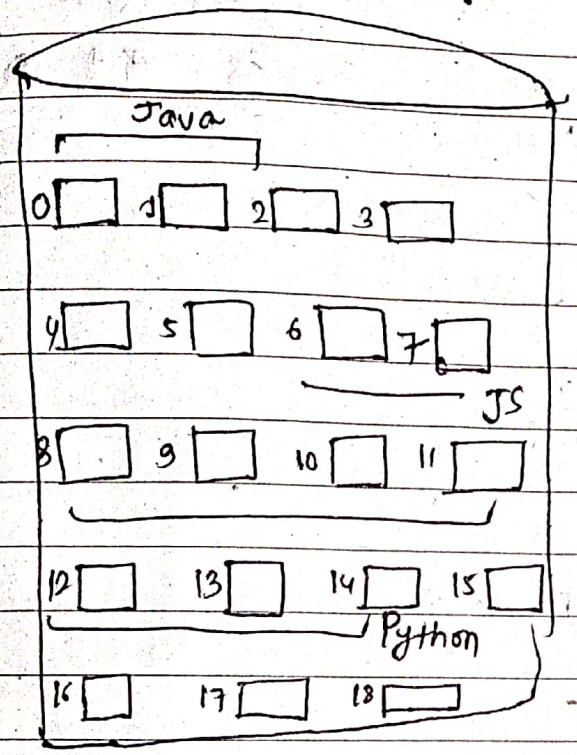
(1) Contiguous Allocation :

This method requires each file to occupy a set of contiguous blocks on the disk.

Scanned with CamScanner

Disk addresses define a linear ordering on the disk.

In the contiguous allocation, accessing the block b+1 after b doesnot require head movements. Head movement is needed ~~when~~ When moving from block of one cylinder to another cylinder. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.



directory

file	start	length
Java	0	2
JS	6	5
Python	12	3

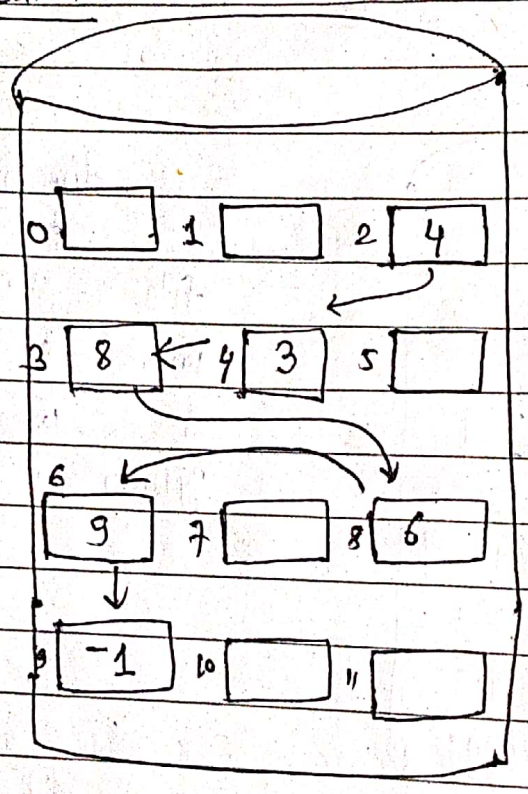
② Linked List Allocation :

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.

for example: a file of 5 blocks might start from block 9, continue to 16, then block 1, block 10 and finally block

25. Each block contains a pointer to the next block. These pointers are not made available to the user. It does not access randomly and provide free space management system so there is no waste of space.

② Linked Allocation:



directory

file	start	end
js	2	9

2 → 4 → 3 → 8 → 6 → 9

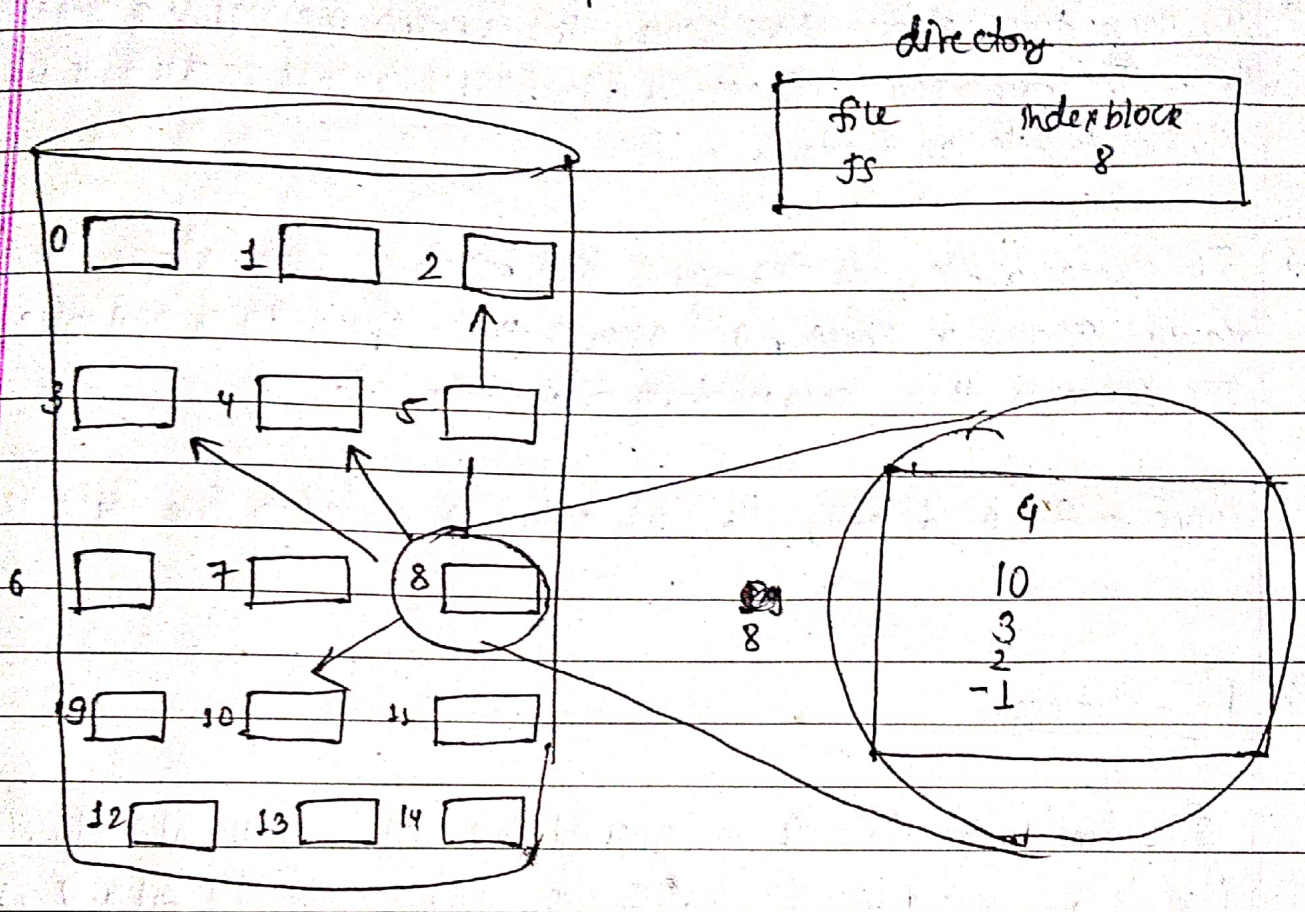
③ Index Allocation:

- Brings all pointers together into index block.
- Need index table
- Random access

Scanned with CamScanner

→ Dynamic access without external fragmentation, but have overhead of index block.

→ mapping from logical to physical in a file of maximum ~~words~~ size 256K words and block size of 512 words.



Scanned with CamScanner

Free Space Management :

→ Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files, if possible.

→ To keep track of free disk space, the system maintains a free-space list. The free space list records all disk blocks that are free - those not allocated to file or directory.

→ To create a file, the free-space list has to be searched for the required amount of space, and allocate that space to a new file. The space is then remove from free space list.

→ When a file is deleted, its disk space is added to free space list.

① Bit-Vector :

The free space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 0, if the block is allocated, the bit is 1.

For example : Consider a disk where blocks are 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free and rest are allocated. The free space bit would be : (Block number starts from 0)

1 1 0 0 00 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 1 . . .

~~0 0 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1~~

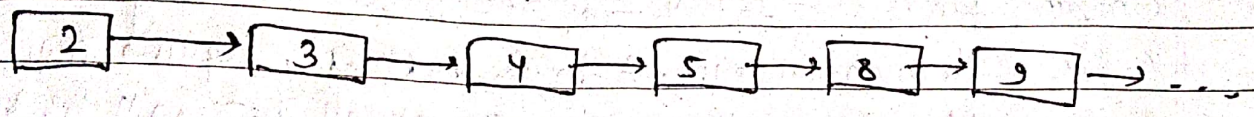
1 1

Blocks Blocks

The main advantage of this approach is that it is relatively simple and efficient to find n consecutive free blocks on disk.

② Linked list :

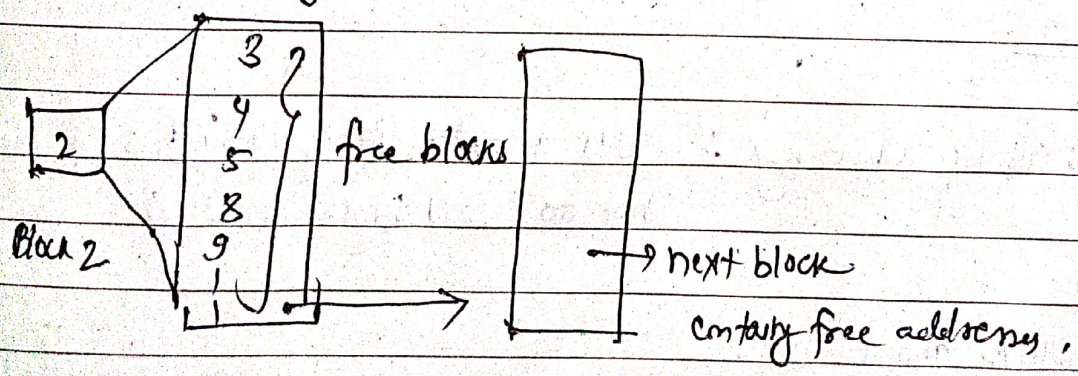
Another method is to link all the free disk blocks together, keeping a pointer to the first free block. This blocks contains a pointer to the next free disk block and so on.



This scheme is not efficient and to traverse the list, each block must be read, which requires substantial I/O time.

③ Grouping :

A modification of free list approach is to store the addresses of n free blocks in the first free block. The first (n-1) of these are actually free. The last one is the disk address of another block containing addresses of another n free blocks. The importance of this implementation is that addresses of a large number of free blocks can be found quickly. In above example



Date
Page :

④ Counting:

Another approach is to take advantage of the fact that ~~are~~, generally, several contiguous blocks may be allocated or free simultaneously, particularly when contiguous allocation is used.

Thus, rather than keeping a list of free disk addresses, the address of first free block is kept and the number of n free contiguous blocks that follows first free blocks. Each entry contains a disk address and a count. Although, each ~~entry~~ requires more space than would a simple disk address, the overall list will be shorter as long as count is greater than 1.

Directory Implementation:

① linkedlist

② Hash Table

Recovery:

① Consistency checking : data in ~~data~~ directory structure is consistent or not to data blocks on disk.

② Backup and Restore:

Scanned with CamScanner