Name: Ayush Jain
SAP ID: 60004200132
Div: B
Computer Engineering

Operating Systems

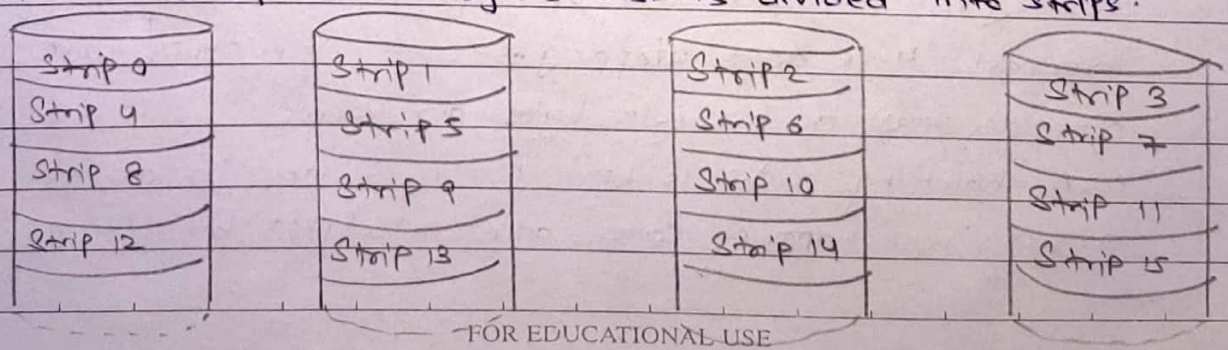
Assignment 1

1) Describe RAID disk management. Provide detailed description of all the levels of RAID.

→ 1) Raid is a set of physical disk drivers viewed by the operating system as a signal logical drive.
2) RAID is the acronym of Redundant Array of Independent DISK or Redundant array of Inexpensive Disks.
3) Here, the data are distributed across the physical drives of an array in a scheme known as stripping.
4) Redundant disk capacity is used to store parity information which guarantees data recoverability in case of disk failure.
5) RAID consists of seven levels:

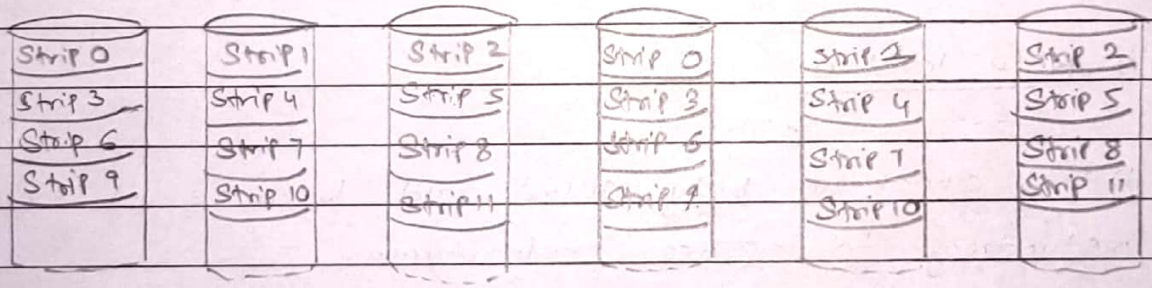
a) **RAID level 0**

i) This cannot be truly called raid because it does not include redundancy to improve performance.
ii) User and system data are distributed across all of the disk in the array.
iii) Here, there is no duplication of data. Hence a block once lost, cannot be recovered. This is used for parallel processing which increases speed. The logical disk is divided into strips.



RAID level 1

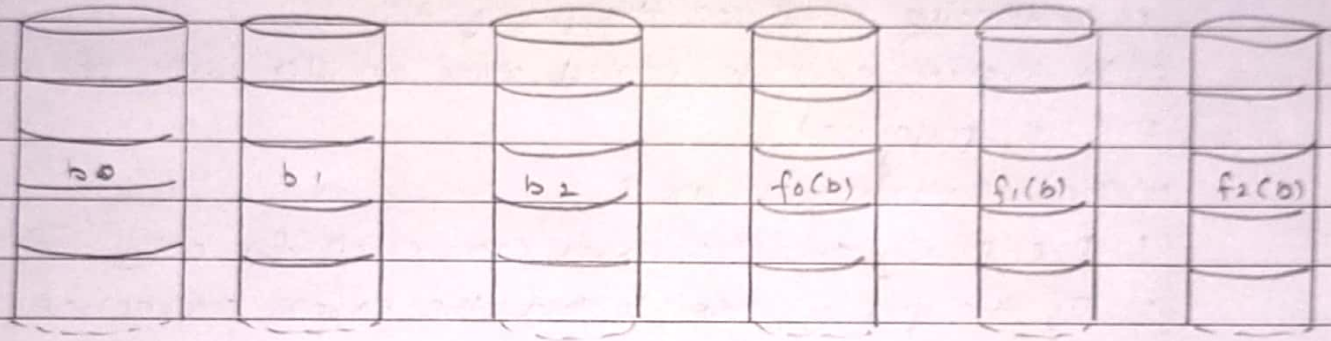
(1) Also known as mirrored disks as redundancy is achieved by the simple expedient of duplicating all the data.
(2) Here if one drive fails the data may be still accessed from the second drive. Here there is no 'write penalty' i.e. a write request requires that both corresponding strips be updated but this can be done in parallel.
(3) The principal disadvantage is cost as it requires twice the disk space of the logical device it supports.
(4) RAID 1 configuration is limited to drives that stores system software and data and other highly critical files. RAID 1 provides real time backup of all data so that is the event of a disk failure, all of the critical data is still immediately available.



RAID level 2

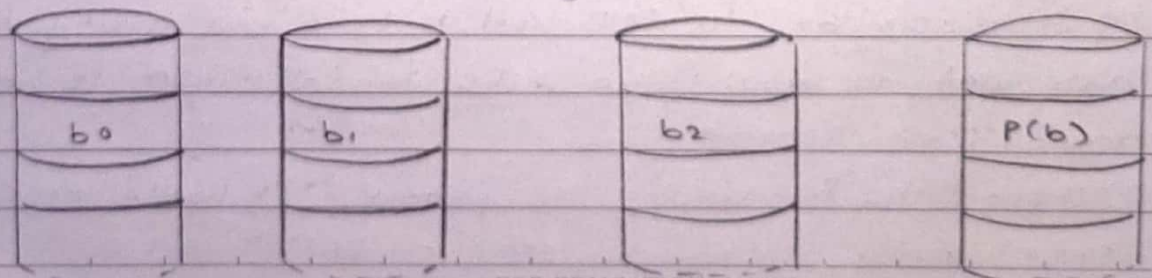
(1) It makes use of parallel access technique. In a parallel access array, all member disk participate in execution of every I/O request. Here data stripping is used. The strips are very small often as small as a single byte or word.
(2) A hamming code is used, which is able to correct single bit error. These hamming code are calculated bit-wise and stored

in the corresponding bit positions or multiple parity disk.
(3) RAID level 2 requires less disk than RAID level 1. If there is a single bit error, the controller can recognize and correct the error instantly so that read access time is not slow.
(4) RAID level 2 will only be effective choice in an environment in which multiple disk error occurs.



RAID level 3

(1) It is similar to RAID level 2. Here, the difference is that it requires only a single redundant disk, no matter the size of disk array.
(2) It computes a single parity bit for the set of individual bits in the same position on all the data disks.
(3) In the event of drive failure, the parity drive is accessed and data is reconstructed from the remaining drives. Once the filled drive is replace the missing data can be restored and operations are resumed. It can achieved very high data transfer rates.



RAID level 4

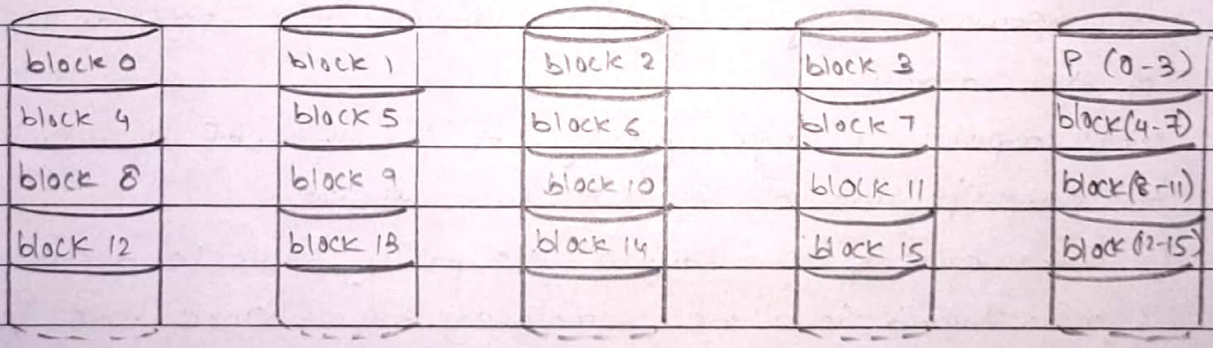
(1) Uses independent access technique. In independent access array each member disk operates seperately. Here the data is striped in blocks.

(2) A parity strip is calculated and this parity is stored in the corresponding strip on the parity disk.

(3) It involves a write penalty when an I/O write request of small size is performed.

(4) The disadvantages are :

1. The parity can give error correction for only one block.
2. If the parity disk is lost the error correction cannot be done.
3. Since the parity is given to one disk only, this disk is repeatedly used. This results in an overload and it may slow down.

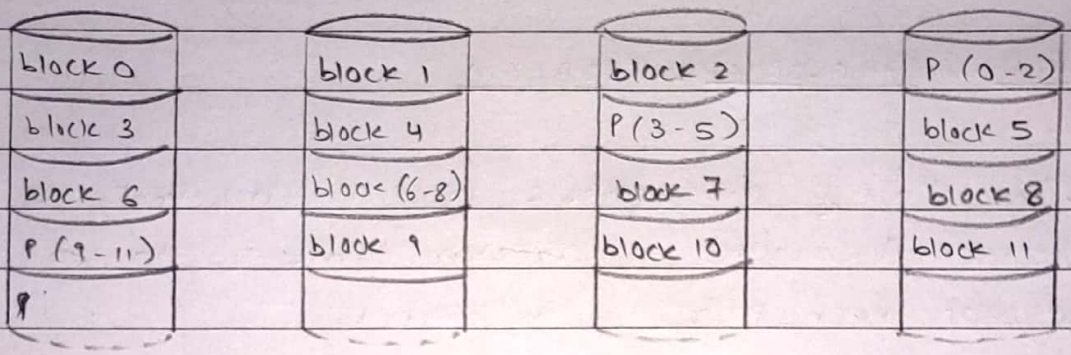


RAID level 5

(1) It is similar to RAID level 4. Here the parity strips are distributed all across the disks. The allocation is done by round-robin scheme.

(2) Those helps in avoiding the potential I/O bottle neck of the single parity disk, as seen in RAID level 4.

(3) It has the characteristic that the loss of any one disk does not result in data loss.

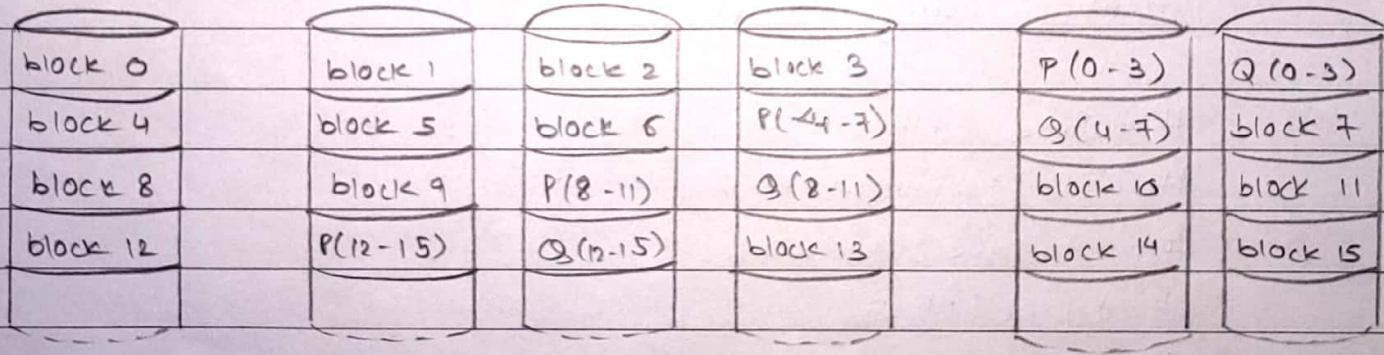


RAID level 6

(1) Here, two different parity calculations are carried out and stored in seperate blocks on different disks. Thus, if user data requires N disks, RAID level 6 will contain N+2 disks.

(2) The main advantage is that it provides extremely high data availability. It incurs a write penalty because each write affects two parity blocks.

(3) Presence of two parity strips helps in calculating for two errors.



Q. 2) Explain Dining Philosophers problem and its solution using semaphores.

→ DINING PHILOSOPHER'S PROBLEM

1) The dining philosopher's prolem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers.
2) To eat, a philosophers needs both their right and left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosophers is available.
3) In case, if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their chopstick and starts thinking again.
4) The dining philosophers problem is the classical prolem of synchronization and demostrates a large class of concurrency control problems.

Solution using Semaphores

```
# include <Pthread.h>
# include<semaphore.h>
# include <stdio.h>
# define N 5
# define thinking 2
# define hungry 1
# define eating 0
# define left (phnum + 4) %.N
# define right (phnum + 1) %. N
```

```
int state[N];
int phil[N] = {0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test (int phnum)
{
  if (state[phnum] == hungry && state[left] != eating
      && state[right] != eating)
  {
    state[phnum] = eating;
    sleep(2);
  Printf("Philosopher %d takes fork %d and %d \n", phnum+1, left+1,
                                                   phnum+1);
  Printf("Philosopher %d is Eating \n", phnum+1);
  sem_post(S[phnum]);
  }
}

 void take_fork (int phnum)
{
  sem_wait(&mutex);
  state[phnum] = hungry;
  Printf("Philosopher %d is hungry \n", phnum+1);
  test(phnum);
  sem_post(&mutex);
  sem_wait(&S[phnum]);
  sleep(1);
}
```

```
void put_fork (int phnum)
{
    sem_wait (&mutex);
    State[phnum] = thinking;
    Printf(" Philosopher %d putting fork %d and %d down \n" ;
                    phnum+1, left+1, phnum+1);
    Printf (" philosopher %d is thinking \n ", phnum+1);
    test (left);
    test (right);
    sem_post (& mutex);
}

void * philosopher (void * num)
{
    while (1)
    {
        int* i = num;
        sleep (1);
        take_fork (*i);
        sleep (0);
        Put_fork (*i);
    }
}

int main ()
{
    int i;
    Pthread_t . thread_id [N];
    sem_init (&mutex, 0, 1);
```

```
for(i=0; i<N; i++)
    sem_init(&s[i],0,0);
for(i=0; i<N; i++)
 {
    pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);

   printf("Philosopher %d is thinking \n"; i+1));
}
 for (i=0; i<N; i++)
    pthread_join (thread_id[i], NULL);

}
```