PES UNIVERSITY

(Established under Karnataka Act No.16 of 2013)

100-ft Ring Road, BSK III Stage, Bangalore – 560 085

Department of Computer Science & Engg

Session: Jan-May 2021

UE19CS254: Operating Systems

UNIT 4 Notes

Read the sections mentioned

Chapter 12 Mass Storage Structures	
Topics	**Page No**
12.1 Mass-Storage Structure, Mass-Storage overview	539-542
12.4 Disk Scheduling	544-550
12.6-12.7 Swap-Space Management, RAID structure	554-566
Chapter 10 File System	
10.1-10.2 File Concept, Access Methods	455-467
10.3 Directory and Disk Structure	467-478
10.4–10.6 File-System Mounting, File Sharing, Protection	478-490
Chapter 11 Implementing File Systems	
11.1-11.3 File-System Structure, File-System Implementation, Directory Implementation	495-505
11.4 Allocation Methods	505-513
11.8, 16.7 Case Study: Linux/Windows File Systems	523-529, 723-729

Text Book

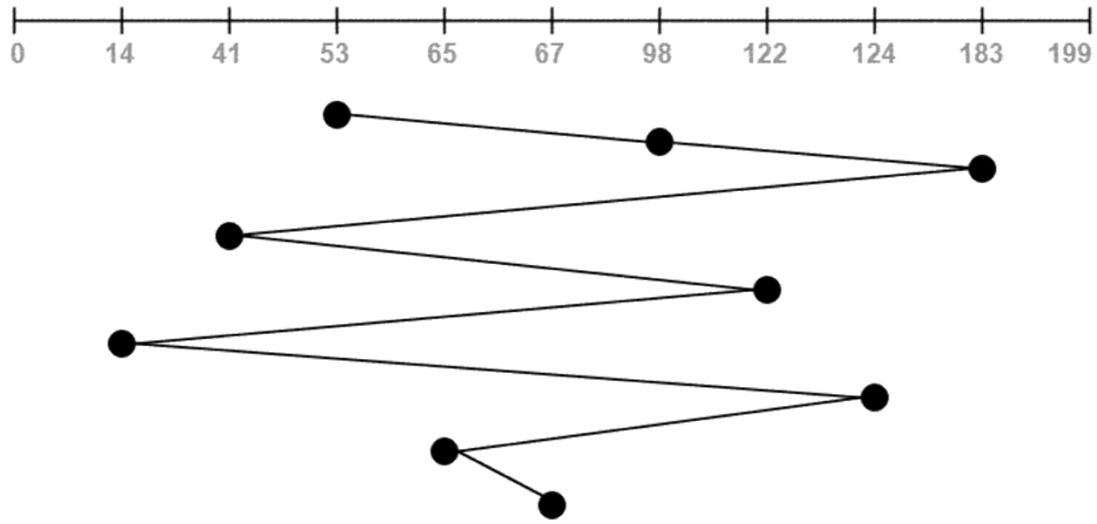
[1]. "Operating System Concepts", Abraham Silberschatz, Peter Baer Galvin, Greg Gagne 9th Edition, John Wiley & Sons, 2016, Indian Print.

Storage Management

Disk Scheduling : Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. Calculate the total head movement (in number of cylinders) incurred while servicing these requests based on FIFO,

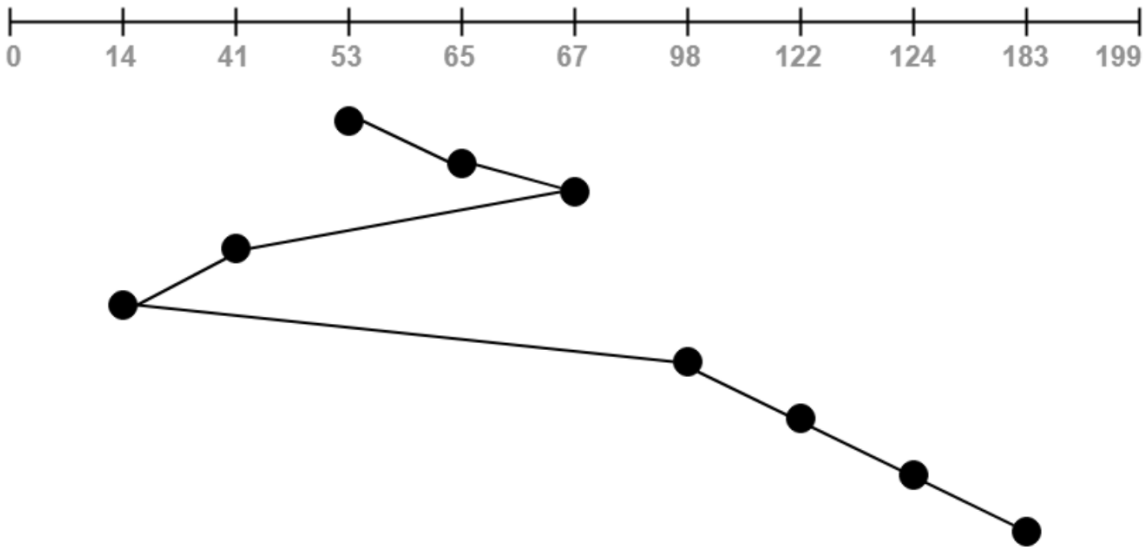
Solution:

1. FIFO:



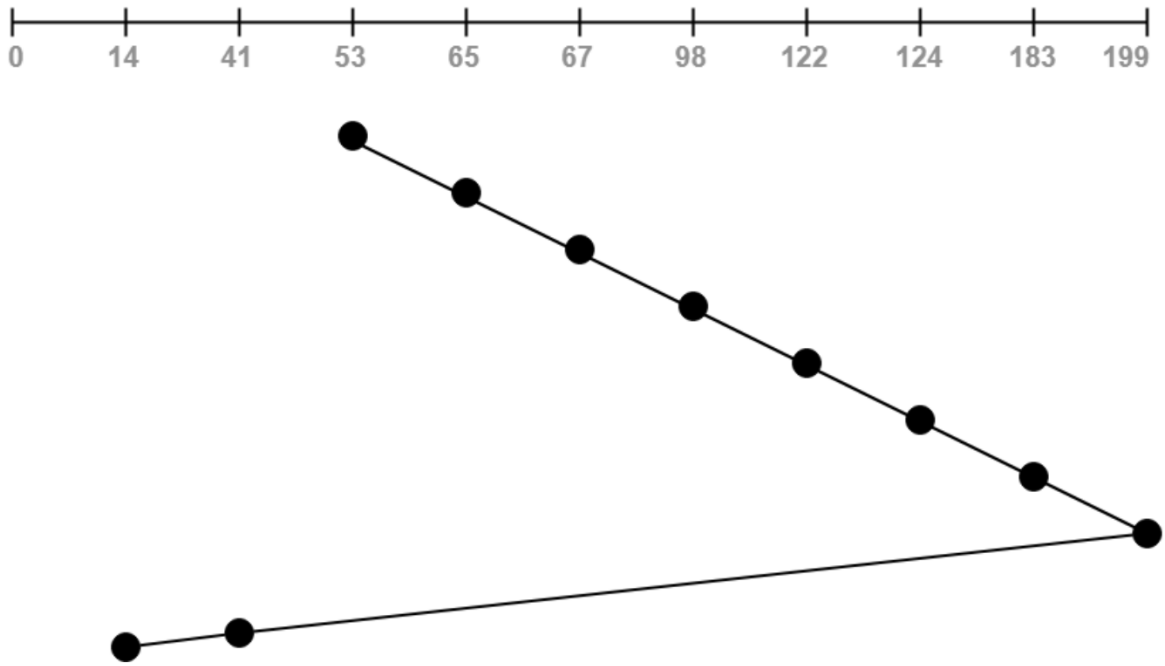
Total head movements = (98 – 53) + (183 – 98) + (183 – 41) + (122 – 41) + (122 – 14) + (124 – 14) + (124 – 65) + (67 – 65) = 632

2. SSTF:



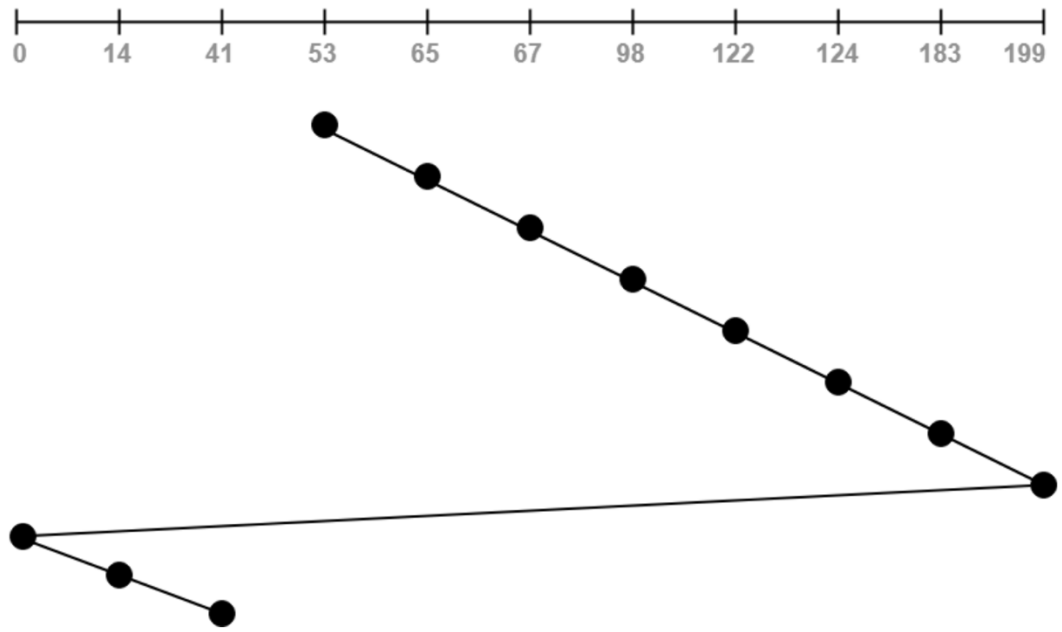
Total head movements = (65 – 53) + (67 – 65) + (67 – 41) + (41 – 14) + (98 – 14) + (122 – 98) + (124 – 122) + (183 – 124) = 236

3. SCAN:



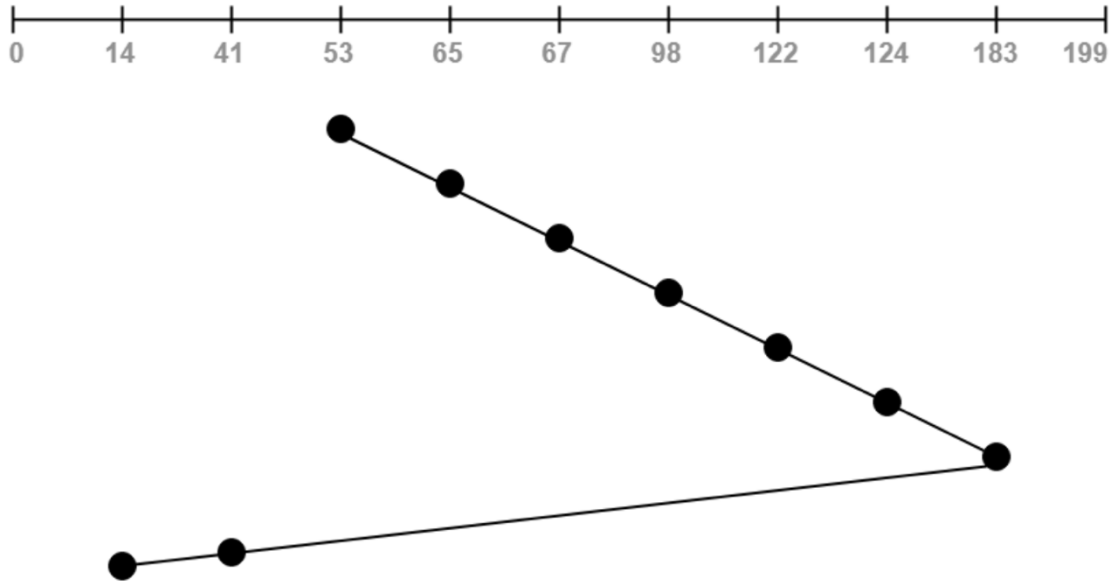
Total head movements = (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 41) + (41 – 14) = 331

4. C-SCAN:



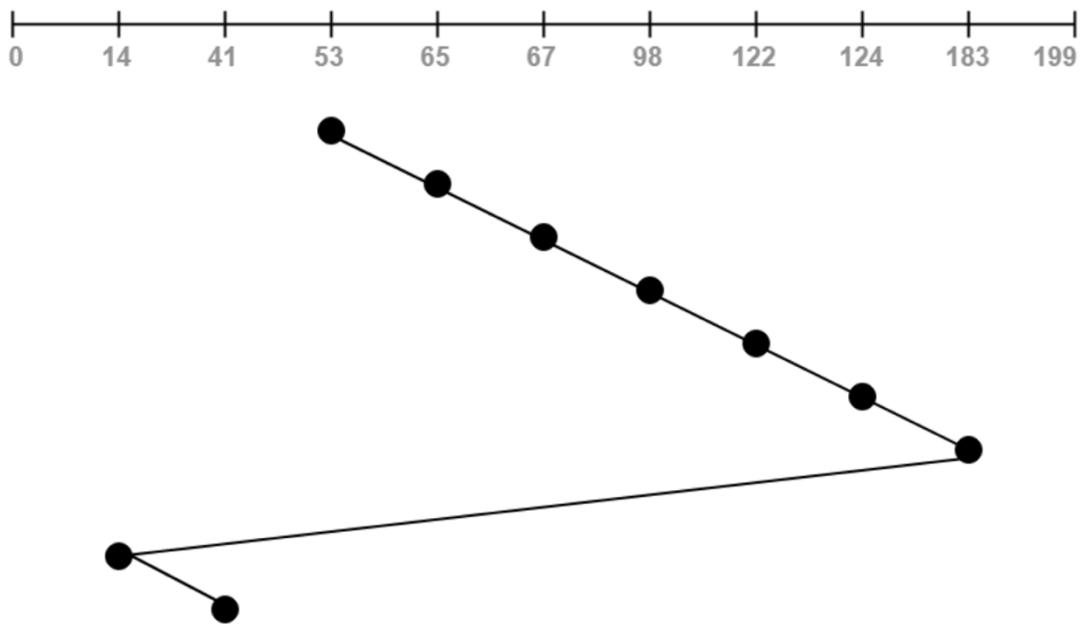
Total head movements = (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 0) + (14 – 0) + (41 – 14) = 386

5. LOOK:



Total head movements = (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 41) + (41 – 14) = 299

6. C-LOOK:



Total head movements = (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 14) + (41 – 14) = 326

Unix System Calls for File operations

1. Open

System call to open a file. Open returns a file descriptor, an integer specifying the position of this open file in the table of open files for the current process.

```
USAGE:
      #include <sys/types.h>
      #include <sys/stat.h>
      #include <fcntl.h>

      int open (const char *path, int oflag);
```

Here, path points to a path name naming a file. open opens a file descriptor for the named file and sets the file status flags according to the value of oflag. Some possible values of oflag are:
`O_RDONLY` - Open for reading only.
`O_WRONLY` - Open for writing only.
`O_RDWR` - Open for reading and writing.

2. Read

System call to read data from a file opened for reading.

```
USAGE:
      #include <unistd.h>
```

```
ssize_t read(int fildes, void *buf, size_t nbyte);
```

Here, read attempts to read nbyte bytes from the file associated with fildes (file descriptor) into the buffer pointed to by buf. If nbyte is zero, read returns zero and has no other results.

3. Write

System call to write data to a file opened for writing

```
USAGE:
     #include <unistd.h>
     ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Here, write attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with fildes. If nbyte is zero and the file is a regular file, write returns zero and has no other results. fildes is a file descriptor.

4. Close

System call to close a file.

```
USAGE
     #include <unistd.h>
     int close(int fildes);
```

Here, close closes the file descriptor indicated by fildes.

In C, one can use the fopen, fread, fwrite, fclose library functions.

```
fp = fopen("myfile.dat", "rb");
```

This opens a file called myfile.dat for reading binary data. fopen is a 'C' library function that calls open, the true system call on UNIX. On any OS, the implementation of the fopen library must eventually call the system call for that OS. The relevant system call is almost always known as open.

Program : To create a file with name ‘x2’ with permissions rw-rw-r—. If the file already exists, do not recreate the file instead display an error message and exit.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
    char name[20];
    int fd;
```

```
    printf("enter a filename : ");
    scanf("%s",name);
    fd=open(name,O_WRONLY | O_CREAT |O_EXCL, S_IRUSR | S_IWUSR|
    S_IRGRP | S_IWGRP | S_IROTH);
    if(fd < 0)
    {
       perror("open ");
       exit(1);
    }
    if(close(fd) < 0)
    {
       perror("close");
       exit(2);
    }
}
```

Program : To copy the contents of an existing file into another file. The names of the two files should be read as an input from the command line.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFSIZE 512

int main (int argc, char** argv)
{
     int from, to, nr, nw, n;
     char buf[BUFSIZE];

     if ((from=open(argv[1], O_RDONLY)) < 0)
      {
           perror(Error opening source file);
           exit(1);
      }

      if ((to=creat(argv[2], 0666)) < 0)
      {
           perror("Error creating destination file");
           exit(2);
      }

     while((nr=read(from, buf, sizeof( buf))) != 0)
      {
        if (nr < 0)
```

```
        {
           perror("Error reading source file");
          exit(3);
        }
        nw=0;
        do
        {
          if ((n=write(to, &buf[nw], nr-nw)) < 0)
          {
            perror("Error writing destination file");
            exit(4);
          }
          nw += n;
        } while (nw < nr);
      }
      close(from);
      close(to);
}
```

Program : To display the contents of a directory, specifying the type for each of its files. The name for the directory should be an input parameter.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
void listDir(char *dirName)
{
   DIR* dir;
   struct dirent *dirEntry;
   struct stat inode;
   char name[1000];
   dir = opendir(dirName);
   if (dir == 0) {
      perror ("Error in opening directory");
      exit(1);
   }
   while ((dirEntry=readdir(dir)) != 0) {
      printf(name,"%s/%s",dirName,dirEntry->d_name);
      lstat (name, &inode);

      if (S_ISDIR(inode.st_mode))
          printf("dir ");
      else if (S_ISREG(inode.st_mode))
            printf ("file ");
           else
              if (S_ISLNK(inode.st_mode))
```

```
                printf ("link ");
            else;
          printf(" %s\n", dirEntry->d_name);
        }
    }

    int main(int argc, char **argv)
    {
        if (argc != 2) {
          printf ("Use: %s directory_name \n", argv[0]);
          exit(0);
        }
        printf(„The contents of the directory are: ”);
        listDir(argv[1]);
    }
```

Program : To use the lseek system call. Open an existing file in read-write mode and
a. Seek before the beginning of the file and check the output.
b. Seek beyond the end of the file, check the offset, close the file, exit and check the file size.
c. Seek beyond the end of the file, write a few bytes, close the file, and check
the file size.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
int main()
{
    char name[20];
    int fd;
    char *msg = "this is a long line";
    printf("enter a filename: ");
    scanf("%s",name);
    fd = open(name,O_RDWR);
    if(fd < 0)
    {
        perror("open");
        exit(1);
    }

    if(lseek(fd,-1001,SEEK_SET) < 0)
    {
        perror("lseek");
```

```
        }

        if(lseek(fd,+10001,SEEK_SET) < 0)
        {
             perror("lseek");
        }

        if(lseek(fd,+10001,SEEK_SET) < 0)
        {
             perror("lseek");
        }

        if(write(fd,msg,strlen(msg))<0)
        {
             perror("write");
        }

        if(close(fd) < 0)
        {
             perror("close");
             exit(2);
        }
}
```

Note:
• Cannot seek before the beginning of file, lseek fails and returns -1.
• Can seek beyond end of file, lseek returns the sum of number of bytes in file and the number of bytes seek-ed beyond end of file. But file size remains unchanged if no write is performed after seeking beyond end of file.
• File size is changed when write is performed after seeking beyond end of file.

References:

1. W. Richard Stevens, Stephen A. Rago, Advanced Programming in the UNIX® Environment: Second Edition, Addison Wesley Professional, 2005