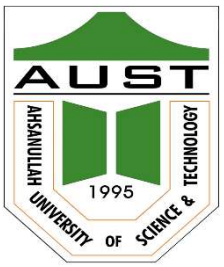


Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

LABORATORY MANUAL

Course No. : CSE3214
Course Title: Operating System Lab

For the students of 3rd Year, 2nd Semester of
B.Sc. in Computer Science and Engineering program

TABLE OF CONTENTS

COURSE OBJECTIVES

The main objective of this Lab is to familiarize students with the architecture of Generic Operating Systems. At the end of the lab works, students should be able to write down programs in C/C++ or in Java to simulate various components of the Operating system such as process scheduler, Inter-process communication using semaphores, threads, deadlock detection, deadlock prevention mechanism etc. They should be also writing simulation codes which deal with the file systems and disk arm scheduling algorithms. The students shall use basic to advanced commands in Unix to simulate Operating System's security and authentication mechanism.

PREFFERED TOOL(S)

C / C++ / Java / Linux Operating System

TEXT/REFERENCE BOOK(S)

- Tanenbaum A.S., Modern Operating Systems, Prentice Hall, 4th Edition.
- Silberschatz A., Galvin P.B., Operating System Concepts, John Wiley & Sons, 7th Edition.

ADMINISTRATIVE POLICY OF THE LABORATORY

- ✓ Students must perform class assessment tasks individually without help of others.
- ✓ Viva for each program will be taken and considered as a performance.
- ✓ Plagiarism is strictly forbidden and will be dealt with punishment.

Session # 1

Introduction to Operating System

Objective: Basic OS management Commands in Unix/Linux
IDE: RedHat Linux/ Ubuntu
You are supposed to get familiar with Linux environment either using RedHat Linux or Ubuntu.

Tasks

1. Installing Linux
2. How to use Command Line Interface (CLI)
3. Use the basic OS commands of various Categories

File Commands

1. ls Directory listing

2. ls -al Formatted listing with hidden files

3. ls -lt Sorting the Formatted listing by time modification

4. cd dir Change directory to dir

5. cd Change to home directory

6. pwd Show current working directory

7. mkdir dir Creating a directory dir

8. cat >file Places the standard input into the file

9. more file Output the contents of the file

10. head file Output the first 10 lines of the file

11. tail file Output the last 10 lines of the file

12. tail -f file Output the contents of file as it grows,starting with

the last 10 lines

13. touch file Create or update file

14. rm file Deleting the file

15. rm -r dir Deleting the directory

16. rm -f file Force to remove the file

17. rm -rf dir Force to remove the directory dir

18. cp file1 file2 Copy the contents of file1 to file2

19. cp -r dir1 dir2 Copy dir1 to dir2;create dir2 if not present

20. mv file1 file2 Rename or move file1 to file2,if file2 is an existing directory

21. ln -s file link Create symbolic link link to file

Process management

1. ps To display the currently working processes

2. top Display all running process

Unix/Linux Command Reference

3. kill pid Kill the process with given pid

4. killall proc Kill all the process named proc

5. pkill pattern Will kill all processes matching the pattern

6. bg List stopped or background jobs,resume a stopped

job in the background

7. fg Brings the most recent job to foreground

8. fg n Brings job n to the foreground

System Info

1. date Show the current date and time

2. cal Show this month's calender

3. uptime Show current uptime

4. w Display who is on line

5. whoami Who you are logged in as

Unix/Linux Command Reference

6. finger user Display information about user

7. uname -a Show kernel information

8. cat /proc/cpuinfo Cpu information

9. cat proc/meminfo Memory information

10. man command Show the manual for command
11. df Show the disk usage
12. du Show directory space usage
13. free Show memory and swap usage
14. whereis app Show possible locations of app
15. which app Show which applications will be run by default

Session # 2

Operating System Security

Objective: Basic Commands in Unix/Linux to simulate Security and authentication Mechanism of the OS

IDE: RedHat Linux/ Ubuntu

Tasks: Use various Categories of OS and Security Commands

Compression

1. tar cf file.tar file Create tar named file.tar containing file
2. tar xf file.tar Extract the files from file.tar
3. tar czf file.tar.gz files Create a tar with Gzip compression
4. tar xzf file.tar.gz Extract a tar using Gzip
5. tar cjf file.tar.bz2 Create tar with Bzip2 compression
6. tar xjf file.tar.bz2 Extract a tar using Bzip2
7. gzip file Compresses file and renames it to file.gz
8. gzip -d file.gz Decompresses file.gz back to file

Network Related

1. ping host Ping host and output results
2. whois domain Get whois information for domains
3. dig domain Get DNS information for domain
4. dig -x host Reverse lookup host
5. wget file Download file
6. wget -c file Continue a stopped download

Unix/Linux Command Reference

Shortcuts

1. ctrl+c Halts the current command

2. ctrl+z Stops the current command, resume with fg in the foreground or bg in the background

3. ctrl+d Logout the current session, similar to exit

4. ctrl+w Erases one word in the current line

5. ctrl+u Erases the whole line

6. ctrl+r Type to bring up a recent command

7. !! Repeats the last command

8. Exit Logout the current session.

>>>>> Get familiar with the following commands:

1. chmod - modify file access rights
2. su - temporarily become the superuser
3. sudo - temporarily become the superuser
4. chown - change file ownership
5. chgrp - change a file's group ownership

chmod

The chmod command is used to change the permissions of a file or directory. To use it, you specify the desired permission settings and the file or files that you wish to modify. There are two ways to specify the permissions. In this lesson we will focus on one of these, called the octal notation method. It is easy to think of the permission settings as a series of bits (which is how the computer thinks about them). Here's how it works:

rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4

If you represent each of the three sets of permissions (owner, group, and other) as a single digit, you have a pretty convenient way of expressing the possible permissions settings. For example, if we wanted to set some_file to have read and write permission for the owner, but wanted to keep the file private from others, we would:

$ chmod 600 some_file

Here is a table of numbers that covers all the common settings. The ones beginning with "7" are used with programs (since they enable execution) and the rest are for other kinds of files.

Value	Meaning

777 No restrictions on permissions. Anybody may do anything. Generally not a desirable setting.
755 The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users.

700 The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others.
666 All users may read and write the file.
644 The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change.
600 The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private.
Directory Permissions

The chmod command can also be used to control the access permissions for directories. Again, we can use the octal notation to set permissions, but the meaning of the r, w, and x attributes is different:

- r - Allows the contents of the directory to be listed if the x attribute is also set.
- w - Allows files within the directory to be created, deleted, or renamed if the x attribute is also set.
- x - Allows a directory to be entered (i.e. cd dir).

Here are some useful settings for directories:
Value Meaning
777 (rwxrwxrwx) No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting.
755 (rwxr-xr-x) The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users.
700 (rwx------) The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others.

Becoming The Superuser For A Short While

It is often necessary to become the superuser to perform important system administration tasks, but as you have been warned, you should not stay logged in as the superuser. In most distributions, there is a program that can give you temporary access to the superuser's privileges. This program is called su (short for substitute user) and can be used in those cases when you need to be the superuser for a small number of tasks. To become the superuser, simply type the su command. You will be prompted for the superuser's password:

```
$ su
Password:
$
```

After executing the su command, you have a new shell session as the superuser. To exit the superuser session, type exit and you will return to your previous session.
In some distributions, most notably Ubuntu, an alternate method is used. Rather than using su, these systems employ the sudo command instead. With sudo, one or more users are granted superuser privileges on an as needed basis. To execute a command as the superuser, the desired command is simply preceeded with the sudo command. After the command is entered, the user is prompted for the user's password rather than the superuser's:

```
$ sudo some_command
Password:
$
```

Changing File Ownership

You can change the owner of a file by using the chown command. Here's an example: Suppose I wanted to change the owner ofsome_file from "me" to "you". I could:

```
$ su
Password:
$ chown you some_file
$ exit
$
```

Notice that in order to change the owner of a file, you must be the superuser. To do this, our example employed the su command, then we executed chown, and finally we typed exit to return to our previous session. The command chown works the same way on directories as it does on files.

Changing Group Ownership

The group ownership of a file or directory may be changed with chgrp. This command is used like this:

```
$ chgrp new_group some_file
```

In the example above, we changed the group ownership of some_file from its previous group to "new_group". You must be the owner of the file or directory to perform a chgrp.

Session # 3
Process Scheduler

Objective: Writing codes in Java or in C/C++ to simulate various Process Scheduling Algorithms

IDE: C/C++, Java

Background:

3a. First Come First Serve Process Scheduling

For FCFS scheduling algorithm, you shall get the number of processes/jobs in the system and their CPU burst times. This is a non-preemptive scheduling where a process getting the CPU will finish its works and the leave the CPU after it exhausts its required CPU time. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. The waiting time and turnaround time of each of the processes can be calculated as well as the average system turnaround and waiting time.

Example:

Process	Arrival Time	CPU Time
P1	4	5
P2	0	7
P3	2	9

Gantt chart:

|--------P2---------|--------P3---------|--------P1---------|

0 7 16 21

	Waiting Time	Turnaround Time
P1	16-4=12	12+5=17
P2	0	0+7=7
P3	7-2=5	5+9=14

Average Waiting Time: (12+0+5)/3 = 5.66

Average Turnaround Time: (17+7+14) / 3 = 12.66

Algorithm

Input the processes along with their cputime and arrivaltime
Find waiting time (wt) for all processes.
As the first process comes in and do not need to wait; waiting time for process 0 will be 0 i.e. waitingtime [0] = 0.
Find waiting time for all other processes i.e. for
process i ;
waitingtime[i] = cputime[i-1] + waitingtime [i-1] – arrivaltime[i].
Find turnaround time[i] = waiting_time [i] + cputime[i]
for all processes.
Find average waiting time =
total_waiting_time / no_of_processes.
Similarly, find average turnaround time =
total_turn_around_time / no_of_processes.

Sample input:

Enter the number of process: 3

Enter the CPU times

5 7 9

Enter the arrival times

4 0 2

Sample Output:

Process 1: Waiting Time: 12 Turnaround Time: 17

Process 2: Waiting Time: 0 Turnaround Time: 7

Process 3: Waiting Time: 5 Turnaround Time: 14

Average Waiting time: 5.66

Average Turnaround time: 12.66

3.b Shortest Job First (Non Pre-emptive version)

For the **Shortest Job First scheduling** algorithm, you shall get the number of processes/jobs in the system and their CPU burst times. This is a non-preemptive scheduling where a process getting the CPU will finish its works and the leave the CPU after it exhausts its required CPU time. The scheduling is performed on the basis of the shortest CPU time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. The waiting time and turnaround time of each of the processes can be calculated as well as the average system turnaround and waiting time.

Example

Process	Arrival Time	CPU Time
P1	4	5
P2	0	7
P3	2	9

Gantt chart:

|--------P2---------|--------P1---------|--------P3---------|

0 7 12 21

	Waiting Time	Turnaround Time
P1	7- 4= 3	3+5=8
P2	0	0+7=7
P3	12-2=10	10+9=19

Average Waiting Time: (3+0+10)/3 = 4.33

Average Turnaround Time: (8+7+19) / 3 = 11.33

Algorithm Hint

Input the processes along with their cputime and arrivaltime

Select a process *i* having the lowest arrival time.
Keep a timer variable that is incremented as follows
Timer = timer + cputime [i]
For all the processes having their arrival time <= Timer
Select a process *i* having the lowest arrival time.

Sample input:

Enter the number of process: 3

Enter the CPU times

5 7 9

Enter the arrival times

4 0 2

Sample Output:

Process 1 : Waiting Time : 3 Turnaround Time : 8

Process 2 : Waiting Time : 0 Turnaround Time : 7

Process 3 : Waiting Time : 10 Turnaround Time : 19

Average Waiting time : 4.66

Average Turnaround time : 11.66

Note:

- You may use Linux OS for compiling and running your code. You may use any programming language (C / gcc / C++) for implementing the program.

- Commands to compile and run C/C++ programs in Linux environement.

  gcc -o output online.c (for C)
  g++ -o output online.cpp (for C++)
  Executing the compiled code: ./output

- You also can use code blocks or any other IDE for compile and run your code

Session # 4
Process Scheduler

Objective: Writing codes in Java or in C/C++ to simulate various Process Scheduling Algorithms

IDE: C/C++ , Java

Background :

4a. Round Robin Scheduling Algorithm

A queue of processes are maintained for all the available processes in the system according to their arrival time. A process P in the system is selected as per the FCFS rule and given opportunity in the CPU for a pre-defined time quantum Q. If the remaining CPU time *r* of the process is less than the time quantum Q, a context switch takes place after *r* amount of time and the process P leaves the queue. Otherwise, the process P runs for Q time quantum and a context switch takes place after *Q* amount of time. The process P goes to the end of queue for more time quantum until it finishes its required amount of time.

Example

Process	Arrival Time	CPU Time
P1	4	5
P2	0	7
P3	6	9
P4	10	9

Time Quantum Q = 3

Gantt chart:

|----P2----|----P2----|----P1----|----P3----|----P2----|----P1----|----P4----|----P3----|----P4----|----P3----|----P4----|

0 3 6 9 12 13 15 18 21 24 27 30

	Waiting Time	Turnaround Time
P1	6-4 + 13-9 =6	6+5=11
P2	0 + 12-6 =6	6+7=13
P3	9-6 + 18-12 + 24-21 = 12	12+9=21

P4 15-10 + 21-18 + 27–24 =11 11+9=20

Average Waiting Time: (6+6+12+11) /4 = 8.75

Average Turnaround Time: (11+13+21+20) / 4 = 16.25

Sample input:

Enter the number of process: 4

Enter the CPU times

5 7 9 9

Enter the arrival times

4 0 6 10

Sample Output:

Process 1 : Waiting Time : 6 Turnaround Time : 11

Process 2 : Waiting Time : 6 Turnaround Time : 13

Process 3 : Waiting Time : 12 Turnaround Time : 21

Process 4 : Waiting Time : 11 Turnaround Time : 20

Average Waiting time : 8.75

Average Turnaround time : 16.25

4b. Priority Scheduling

Priority scheduling is one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on first come first served basis. If used as a pre-emptive version, a running process having lower priority will be sent back to the ready queue if a process with higher priority arrives in the queue.

Process	Arrival Time	CPU Time	Priority
P1	4	5	0
P2	0	7	2
P3	2	9	1

Gantt chart: (Non pre-emptive)

|--------P2---------|--------P1---------|--------P3---------|

0 7 12 21

	Waiting Time	Turnaround Time
P1	7-4= 3	3+5=8
P2	0	0+7=7
P3	12-2=10	10+9=19

Average Waiting Time: (3+0+10)/3 = 4.33

Average Turnaround Time: (8+7+19) / 3 = 11.33

Gantt chart: (Pre-emptive Version)

|--------P2---------|--------P3---------|--------P1---------|--------P3---------|--------P2---------|

0 2 4 9 16 21

	Waiting Time	Turnaround Time
P1	4-4= 0	0+5=5
P2	0 + 16-2 = 14	14+7=21
P3	2-2 + 9-4 = 5	5+9=14

Average Waiting Time: (0+14+5)/3 = 6.33

Average Turnaround Time: (5+21+14) / 3 = 13.33

Algorithm Hint (Non Pre-emptive)

Input the processes along with their cputime , arrivaltime and process priority (lower value means higher priority)

Select a process *i* having the lowest priority value.
Keep a timer variable that is incremented as follows
Timer = timer + cputime [i]
For all the processes having their arrival time <= Timer
Select a process *i* having the lowest priority value.

Sample input:

Enter the number of process: 3

Enter the CPU times

5 7 9

Enter the arrival times

4 0 2

Enter the priority values

0 2 1

Sample Output:

Process 1 : Waiting Time : 0 Turnaround Time :5

Process 2 : Waiting Time : 14 Turnaround Time : 21

Process 3 : Waiting Time : 5 Turnaround Time : 14

Average Waiting time : 4.33

Average Turnaround time : 11.33

***** Brainstorm for deriving a solution for the pre-emptive version**

Note:

- You may use Linux OS for compiling and running your code. You may use any programming language (C / gcc / C++) for implementing the program.

- Commands to compile and run C/C++ programs in Linux

  gcc -o output online.c (for C)
  g++ -o output online.cpp (for C++)
  Executing the compiled code: ./output

- You also can use code blocks or any other IDE for compile and run your code

Session # 5

Introduction to threads and how to use them in Java and Simulating the Producer-Consumer Problem

Objective: Introduction to threads and how to use them in Java and Implementing a classical Inter-process communication problems: **Producer-Consumer Problem**

IDE: C/C++ , Java

Sample code for thread implementation in Java:

```
package Mythread;
class FirstThread implements Runnable
{
 @Override
 public void run()
 {
  for ( int i=1; i<=10; i++)
  {
     System.out.println( "Messag from First Thread : " +i);
     try
     {
       Thread.sleep (1000);
     }
     catch (InterruptedException interruptedException)
     {
       System.out.println( "First Thread is interrupted when it is sleeping" +interruptedException);
     }
  }
 }
}
 class SecondThread implements Runnable
{
   @Override
  public void run()
  {
     for ( int i=1; i<=10; i++)
    {
      System.out.println( "Messag from Second Thread : " +i);
       try
      {
      Thread.sleep(1000);
      }
      catch (InterruptedException interruptedException)
      {
```

```
        System.out.println( "Second Thread is interrupted when it is sleeping"
+interruptedException);
       }
     }
   }
}
public class Mythread
{
    public static void main(String args[])
    {
      FirstThread   firstThread = new FirstThread();
      SecondThread   secondThread = new SecondThread();
      Thread thread1 = new Thread(firstThread);
      thread1.start();
      Thread thread2 = new Thread(secondThread);
      thread2.start();
    }
```

Task:

Producer-consumer problem also known as bounded-buffer problem. The problem describes two process, the producer and the consumer which share a common fixed-size buffer used as a queue.

- A producer's job is to generate data, putting it into the buffer and start again.
- The purpose of the consumer is to consume the available data from the buffer and empty the buffer.

The standard solution to this problem uses three semaphores: *empty* and *full*, which count the number of empty and full slots in the buffer, and *mutex*, which is a binary (or mutual exclusive) semaphore that protects the actual insertion or removal of items in the buffer. The producer and consumer are running as separate threads and they will move items to and from a buffer that is synchronized with the *empty and full* data structures.

Assume the following parameters for the program

- *n-sized* buffer, can hold n-number of items in total
- Semaphore *mutex* is initialized to the value 1
- Semaphore *full* is initialized to the value 0
- Semaphore *empty* initialized to the value *n*

The Pseudocode

ProducerProcess ( )

```
do {    ...
      /* produce an item in next_produced */
       wait(empty);
      wait(mutex);
        ...
      /* add next produced to the buffer */
        ...
      signal(mutex);
      signal(full);
   } while (true);
```

ConsumerProcess ()

```
do {
      wait(full);
      wait(mutex);
        ...
      /* remove an item from buffer to next_consumed */
        ...
      signal(mutex);
      signal(empty);
        ...
      /* consume the item in next consumed */
        ...
   } while (true);
```

Input:

Producer 1: Data to insert in the buffer
Data: 10
Producer 3: Data to insert in the buffer
Data: 110
..
..

Output :

Producer 1: Data 10 is inserted in the buffer
Empty space : n-1
Full space : 1
Producer 3: Data 110 is inserted in the buffer
Empty space : n-2

```
Full space : 2
Consumer 0: Consumed Data 10 from the buffer
Full space : 1
Consumer 3: Consumed Data 110 from the buffer
Full space : 0
```

Session # 6
Simulating the Reader-Writer Problem

Objective: Implementing a classical Inter-process communication problems : Reader-Writer Problem

IDE: C/C++ , Java

Sample code for thread implementation in Java:

```
package Mythread;
class FirstThread implements Runnable
{
 @Override
 public void run()
 {
  for ( int i=1; i<=10; i++)
  {
     System.out.println( "Messag from First Thread : " +i);
     try
     {
       Thread.sleep (1000);
     }
     catch (InterruptedException interruptedException)
     {
       System.out.println( "First Thread is interrupted when it is sleeping" +interruptedException);
     }
  }
 }
}
 class SecondThread implements Runnable
{
   @Override
  public void run()
  {
     for ( int i=1; i<=10; i++)
    {
      System.out.println( "Messag from Second Thread : " +i);
       try
      {
      Thread.sleep(1000);
      }
      catch (InterruptedException interruptedException)
      {
       System.out.println( "Second Thread is interrupted when it is sleeping"
+interruptedException);
```

```
      }
    }
  }
}
public class Mythread
{
   public static void main(String args[])
   {
     FirstThread   firstThread = new FirstThread();
     SecondThread   secondThread = new SecondThread();
     Thread thread1 = new Thread(firstThread);
     thread1.start();
     Thread thread2 = new Thread(secondThread);
     thread2.start();
   }
         }
```

Task:

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non-zero number of readers accessing the resource at that time. From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

You need to implement the code in Java programming language.

Pseudocode

In the standard solution, we use one mutex *m* and a semaphore *w*. An integer variable *read_count* is used to maintain the number of readers currently accessing the resource. The variable *read_count* is initialized to 0. A value of 1 is given initially to *m* and *w.*

Instead of having the process to acquire lock on the shared resource, we use the mutex *m* to make the process to acquire and release lock whenever it is updating the *read_count* variable.

The code for the writer process looks like this:

```
while(TRUE)
{
   wait(w);

  /* perform the write operation */

  signal(w);
}
```

```
while(TRUE)

{

   //acquire lock

   wait(m);

   read_count++;

   if(read_count == 1)

      wait(w);

   //release lock

   signal(m);

    /* perform the reading operation */

   // acquire lock

   wait(m);

   read_count--;

   if(read_count == 0)

      signal(w);

   // release lock

   signal(m);
```

- As seen above in the code for the writer, the writer just waits on the w semaphore until it gets a chance to write to the resource.
- After performing the write operation, it increments w so that the next writer can access the resource.
- On the other hand, in the code for the reader, the lock is acquired whenever the read_count is updated by a process.
- When a reader wants to access the resource, first it increments the read_count value, then accesses the resource and then decrements the read_count value.
- The semaphore w is used by the first reader which enters the critical section and the last reader which exits the critical section.
- The reason for this is, when the first readers enters the critical section, the writer is blocked from the resource. Only new readers can access the resource now.
- Similarly, when the last reader exits the critical section, it signals the writer using the w semaphore because there are zero readers now and a writer can have the chance to access the resource.

Input / Output :

```
Data in Buffer : 1 5 6 7 8 9  15
Reader 1: Reading 7
Reader 2: Reading 1
```

```
Reader 3: Reading 15
Reader 3: Leaves
Reader 2: Leaves
Reader 1: Leaves

Writer 0 : Writes in  the system 23, 45
Data in Buffer : 5 6 8 9 23 45
Writer 1 : Waiting
Reader 1: Waiting
Reader 2: Waiting
Writer 0 : Leaves the system
Writer 1 : Writes in  the system  3, 4
Data in Buffer : 5 6 8 9 23 45 3 4
Writer 1 : Leaves the system
Reader 1: Reading 5
..
..
```

Session # 7
Deadlock management

Objective: Deadlock detection using resource allocation graph for a single quantity of each resources using a cycle detection algorithm.

IDE: C/C++ , Java

Background

Deadlock:

From the application point of view, a deadlock is a situation in which two computer programs are sharing the same resource and effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

Resource Allocation Graph (RAG)

In the **Resource Allocation Graph ,** Vertices (Rectangle : Resource or Circle: process) represent process and resources. The directed edge from a process P to a resource R represents that the process P is willing to get resource R. The directed edge from a resource R to a process P represents that the process P is holding the resource R in its custody.

Deadlock Detection using Resource Allocation Graph (RAG)

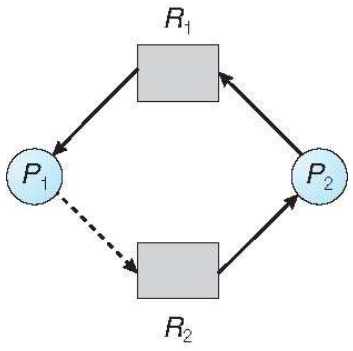
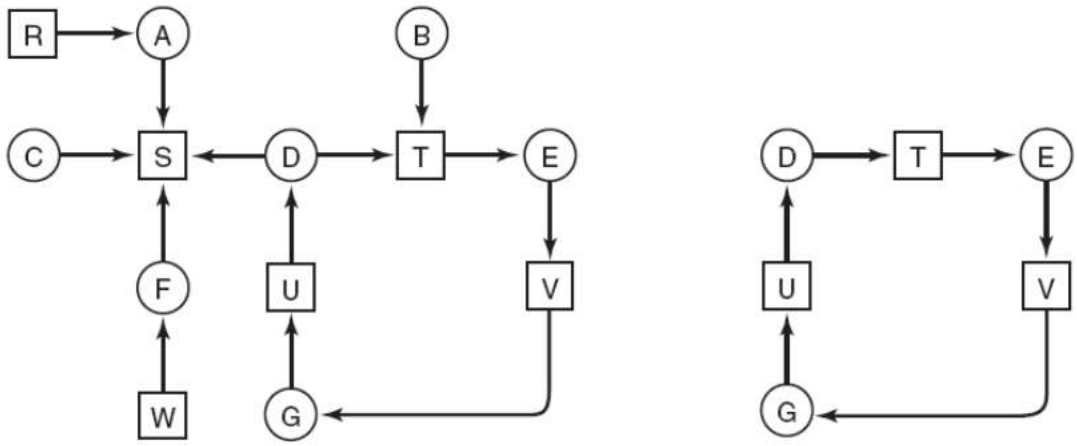


Fig1: A simple RAG

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.

Deadlock detection using Cycle detection method



Algorithm

1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.
4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

Input: A Directed graph (nodes can be either resource or process)

Output: Number of cycles / number of deadlocks existing

Input:

Number of nodes : 13
Node names : R A C S D T B E F U V W G
Edges: R to A

A to S

C to S

…

…

Output

1 - Deadlock Detected Among nodes: D T E V G U

Additional Task:

Recovery from a Deadlock

When you are taking input of the node names, you need to ask whether the node is a resource or a process.

You can recover deadlock in the following way,

1. Pre-empt a resource from a process in the existing cycle and allocate it to other process to break the deadlock
2. Kill a process and give its resource to the process looking for it.

Brainstorm to derive a solution code for this task

Session # 8
Simulate Banker's Algorithm for Deadlock Avoidance

Objective: Simulate Banker's Algorithm used for Deadlock Avoidance and find whether the system is in safe state or not.

IDE: C/C++ , Java

Background :

In Banker's algorithm, when a new process enters a system, it declares the maximum number of instances of each resource type it needed. This number may exceed the total number of resources currently available in the system. When a process requests a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will, the resources are allocated; otherwise the process must wait until some other process release the resources. If all the processes in the system can be satisfied in any order by giving them their requested number of resources, the system is said to be in safe state and the order of serving is called the safe sequence.

Working Procedure

Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:

Available

It is an array of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type R(j).

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process P(i) can request atmost k instances of resource type R(j).

3. Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).

4. Need

It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task. Here , Need[i][j] = Max[i][j] - Allocation [i][j]

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initially,
Work = Available
Finish[i] =false for i = 0, 1, ... , n - 1.
This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both Finish[i] ==false and Needi <= Work
If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:
Work = Work + Allocation;
Finish[i] = true;
Go to step 2.
When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If Finish[i] == true for all i, then the system is in a safe state.

Example: Resource X Y Z

X Y Z

Resources= {20, 19, 16}

	Allocation			**Max**			**Need**		
	X	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
P1	1	1	1	7	5	6	6	4	5
P2	2	3	3	5	8	5	3	5	2
P3	3	0	5	9	2	8	6	2	3
P4	2	2	0	8	8	6	6	6	6
P5	3	1	0	14	4	5	11	3	5
P6	1	2	1	4	6	3	3	4	2
P7	1	1	1	2	9	2	1	8	1

<P2 , P1, P6, P7, P3, P4, P5>is the safe sequence

Task: Given a set of data for the Banker's algorithm, detect whether the system is in safe state or not

Program Input:

Enter the no. of processes: 4

Enter the no. of resources:3

Process 1
Maximum value for resource 1:3
Maximum value for resource 2:2
Maximum value for resource 3:2
Allocated from resource 1:1
Allocated from resource 2:0
Allocated from resource 3:0

Process 2
Maximum value for resource 1:6
Maximum value for resource 2:1
Maximum value for resource 3:3
Allocated from resource 1:5
Allocated from resource 2:1
Allocated from resource 3:1

Process 3
Maximum value for resource 1:3
Maximum value for resource 2:1
Maximum value for resource 3:4
Allocated from resource 1:2
Allocated from resource 2:1
Allocated from resource 3:1

Process 4
Maximum value for resource 1:4
Maximum value for resource 2:2
Maximum value for resource 3:2
Allocated from resource 1:0
Allocated from resource 2:0
Allocated from resource 3:2

Enter total value of resource 1:9
Enter total value of resource 2:3
Enter total value of resource 3:6

Program Output:

The System is currently in safe state and
< P2 P1 P3 P4 > is the safe sequence.

NB: You can receive the input from Standard Input or **preferably from a file**.

Session # 9

Simulating Memory Management of an OS

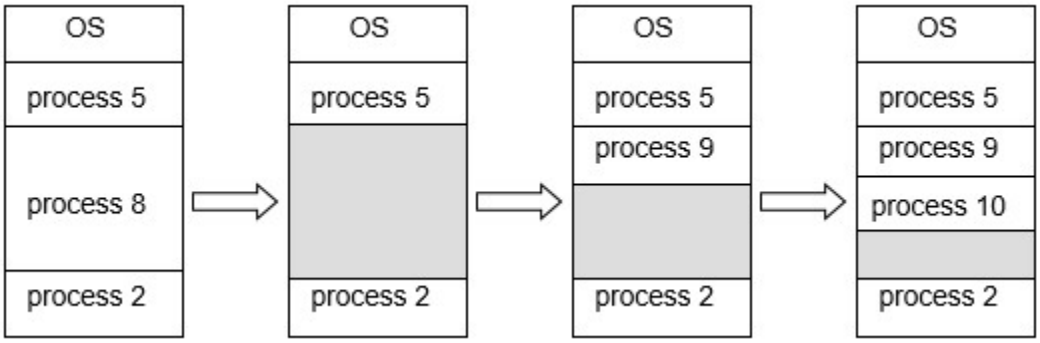
Objective: Implementing the general concepts of the following Memory allocation techniques and fragmentations. such as : First-fit , Best Fit, Worst-fit (In Fixed/ variable length partitioning)

IDE: C/C++ , Java

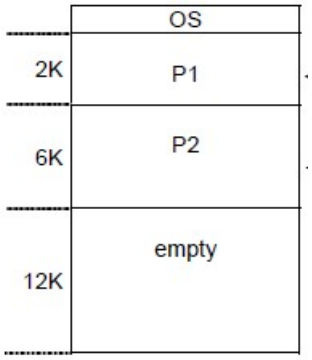
Background

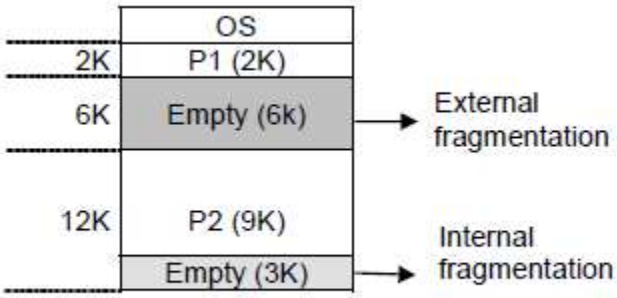
Memory allocation techniques:

Hole – block of available memory; holes of various size are scattered throughout memory
When a process arrives, it is allocated memory from a hole large enough to accommodate it
Operating system maintains information about:
a) allocated partitions b) free partitions (hole)



Fixed Partitioning: In this method, memory is divided into partitions whose sizes are fixed. OS is placed into the lowest bytes of memory. The number of fixed partition gives the degree of multiprogramming.





If a partition is being used by a process requiring some memory smaller than the partition size, then it is called an internal fragmentation. If a whole partition is currently not being used, then it is called an external fragmentation.

External Fragmentation in variable partitioning: Total memory space exists to satisfy a request, but it is not contiguous.

Example:

How to satisfy a request of size n from a list of free holes

First-fit: Allocate the first hole that is big enough to fit the request.

Best-fit: Allocate the smallest hole that is big enough; must search entire list, (unordered list by size)

Worst-fit: Allocate the largest hole; must also search entire list.

Memory Block	Size
Block 1	50
Block 2	200
Block 3	70
Block 4	115
Block 5	15

Memory Requests

100, 10, 35, 15, 23, 6, 25, 55, 88, 40

Step by step memory allocation situation in First Fit

100	10	35	15	23	6	25	55	88	40
50	40	5	5	5	5	5	5	5	CANT
100	100	100	85	62	56	31	31	31	BE
70	70	70	70	70	70	70	15	15	Allocated
115	115	115	115	115	115	115	115	27	Ext. Fragmentation
15	15	15	15	15	15	15	15	15	**93**

Step by step memory allocation situation in Best Fit

100	10	35	15	23	6	25	55	88	40
50	50	15	0	0	0	0	0	0	0
200	200	200	200	200	200	200	145	57	17
70	70	70	70	57	57	32	32	32	32
15	5	5	5	5	5	5	5	5	5
15	15	15	15	15	9	9	9	9	9
No External Fragmentation									

Step by step memory allocation situation in Worst Fit

100	10	35	15	23	6	25	55	88	40
50	50	50	50	50	50	50	50	**CANT**	
100	100	100	85	62	62	62	62	**BE**	
70	70	70	70	70	63	63	8	**ALLOCATED**	
115	105	70	70	70	70	45	45		
15	15	15	15	15	15	15	15		
External Fragmentation : 235									

Program

Input:
Memory Holes : 50 200 70 115 15
Memory Requests: 100, 10, 35, 15, 23, 6, 25, 55, 88, 40
Output :
First Fit
Memory Allocation Step by step (See the example above)
External Fragmentation: 93

Session # 10

Simulating Page Replacement- techniques of an OS

Objective: Implementing the general concepts of the following Memory Page Replacement techniques such as:
FIFO, OPTIMAL, LRU

IDE: C/C++ , Java

Background

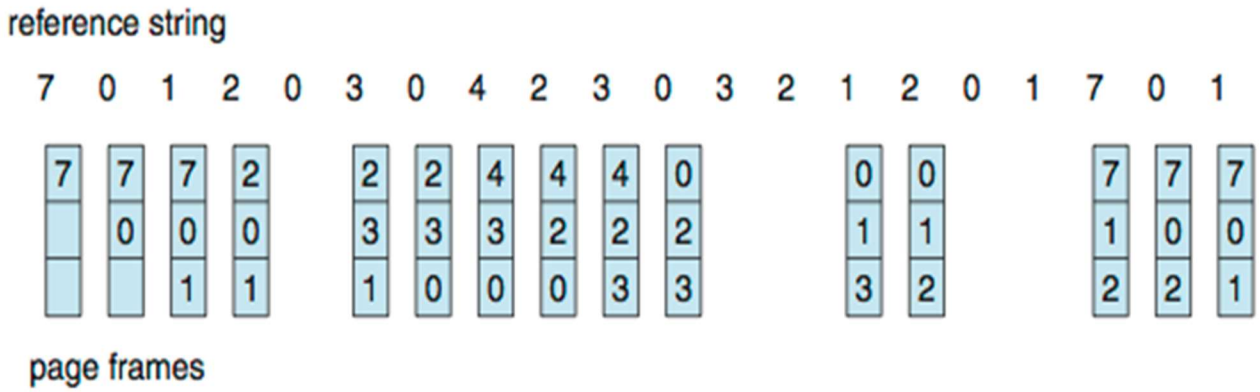
10.a FIFO Page Replacement Algorithm

If there is no space left in memory, select the page came earlier in the memory compared to the available pages in the memory and replace that page with the new one.

Example

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

3 frames (3 pages can be in memory at a time per process)



Total : 15 page faults

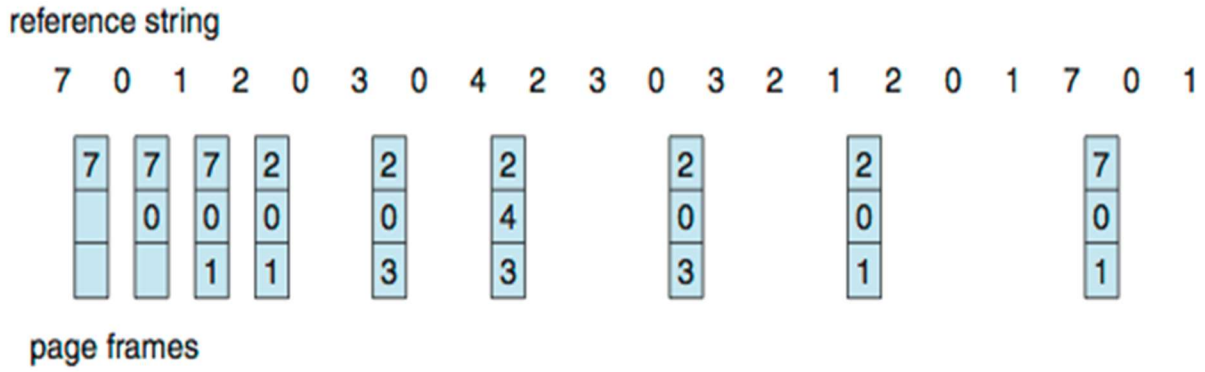
10.b Optimal Page Replacement Algorithm

If there is no space left in memory, select the page that will either not be used or used later compared to the available pages in the memory and replace that page with the new one.

Example

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

3 frames (3 pages can be in memory at a time per process)



Total : 9 page faults

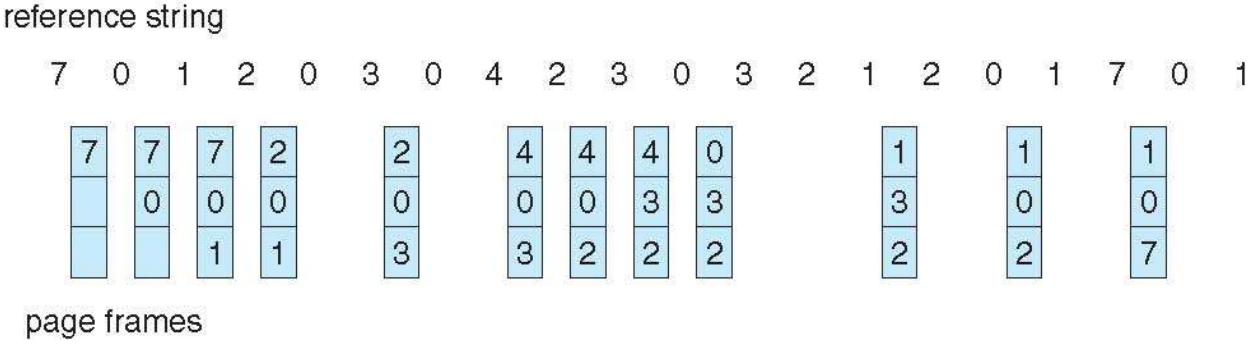
10.c Least Recently Used Algorithm

If there is no space left in memory, select the page that has not been used recently compared to the available pages in the memory and replace that page with the new one. A page which is not used recently will be replaced.

Example

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

3 frames (3 pages can be in memory at a time per process)



Total : 12 page faults

ALGORITHM

Start

Read the number of frames from the terminal

Read the number of pages from the terminal

Read the page number requests

Initialize the values in frames to -1

Allocate the pages in to frames as per the algorithm.

Display the number of page faults in the terminal

Stop

Input:

Number of pages: 8

Number of Page References: 22

Reference String: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Number of Memory Page Frame: 3

Output:

Number of page fault using FIFO Page replacement Algorithm: 15

Page Fault Rate: 68%

Number of page fault using Optimal Page replacement Algorithm: 9

Page Fault Rate: 41%

Number of page fault using Least Recently Used Page replacement Algorithm: 12

Page Fault Rate: 55%

Session # 11

Simulating Page Replacement- techniques of an OS

Objective: Implementing the general concepts of the following Memory Page Replacement techniques such as: LRU using stack, Enhanced 2nd Chance

IDE: C/C++ , Java

Background

10.a Least Recently Used page replacement algorithm using stack

If there is no space left in memory, select the page that has not been used recently compared to the available pages in the memory and replace that page with the new one. A page which is not used recently will be replaced.

Example

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

3 frames (3 pages can be in memory at a time per process)

Using Stack:

Algorithm:

Start
Read the number of frames from the terminal
Read the number of pages from the terminal
Read the page number requests
For each request, Place the requested page in the memory.
The most recent page goes on top of the stack and other pages are shifted down by one place. If there is no space in the stack, replace the page at the bottom.
Put the new page at the top of the stack and shift the rest of the pages by one place towards downward.
Stop

Step by step Stack situation

7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0

7 0 1 2 2 3 0 4 2 2 2 2 0 3 3 1 2 0 1 7

Input:

Number of pages: 8

Reference String : 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Number of Memory Page Frame: 3

Output:

```
7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1
  7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0
    7 0 1 2 2 3 0 4 2 2 2 2 0 3 3 1 2 0 1 7
```

11.b: **Enhanced second chance algorithm**

Uses Reference bit and Modify bit to select a page for replacement.

- (0, 0) - Neither recently used nor modified - best page to replace.
- (0, 1) - Not recently used but modified - not as good because we need to swap out a page, but still better than used pages.
- (1, 0) - Recently used but unmodified.
- (1, 1) - Recently used and modified - the worst page to replace.

When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class.

Task: You need to use Linked List based implementation for this task. Store reference, modify value for each page in the linked implementation.

Session # 12

Disk-Arm Scheduling

Objective: Writing program codes to simulate the following Disk-Arm scheduling algorithms FCFS, SCAN, CSCAN, LOOK, C-LOOK.

IDE: C/C++ , Java

Disk Arm Scheduling

The operating system tries to use hardware efficiently for disk drives access time, disk bandwidth

Access time has two major components:

- *Seek time* is time to move the heads to the cylinder containing the desired sector
- *Rotational latency* is additional time waiting to rotate the desired sector to the disk head.

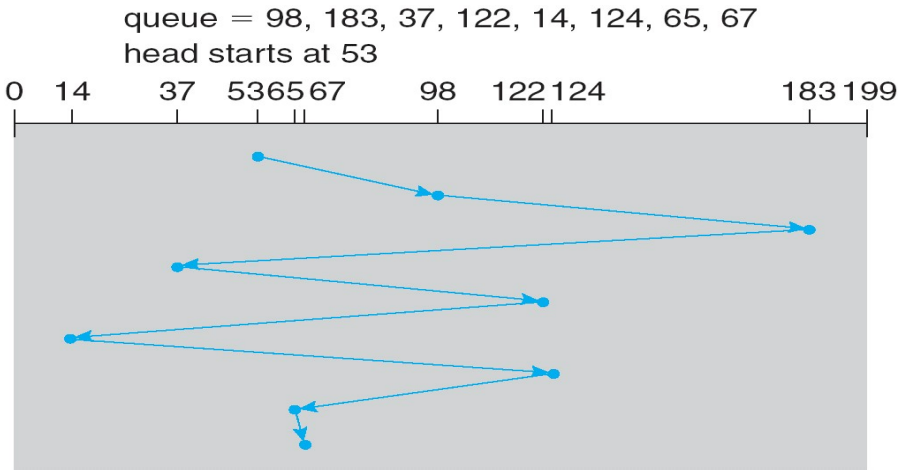
The main objective of disk arm scheduling is to minimize the seek time

FCFS (First Come First Serve) Scheduling

The cylinder requests are served in the same order they appear in the request pool.

We illustrate them with a request queue (0-199 cylinder).

Cylinder Requests: 98, 183, 37, 122, 14, 124, 65, 67 ; Head pointer starts at 53

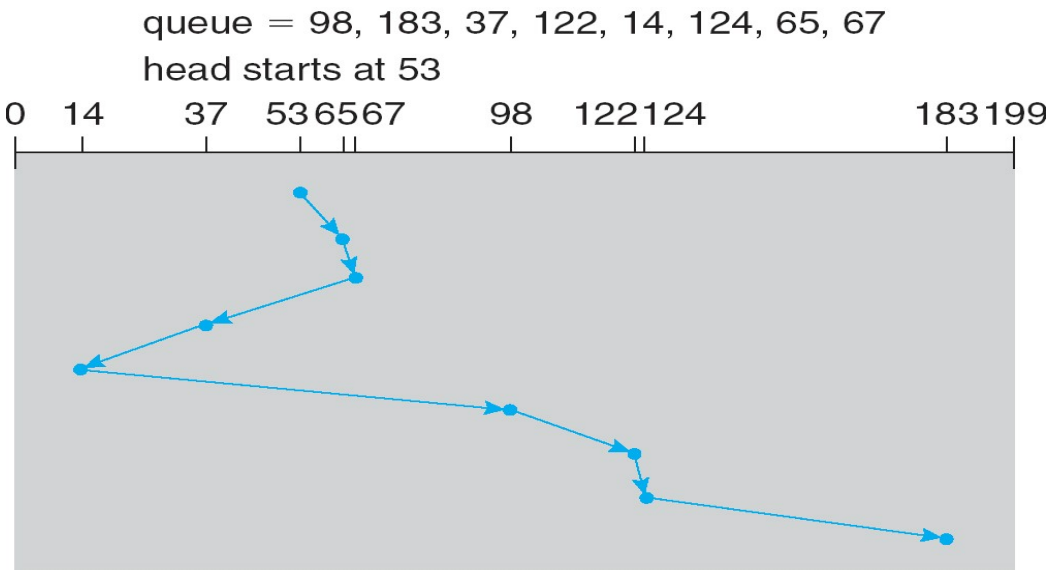


Total Cylinder movement: 640

SSTF (Shortest Seek time First) Scheduling

The next cylinder request is served which is nearby from the current serving location in the request pool.
We illustrate them with a request queue (0-199 cylinder).

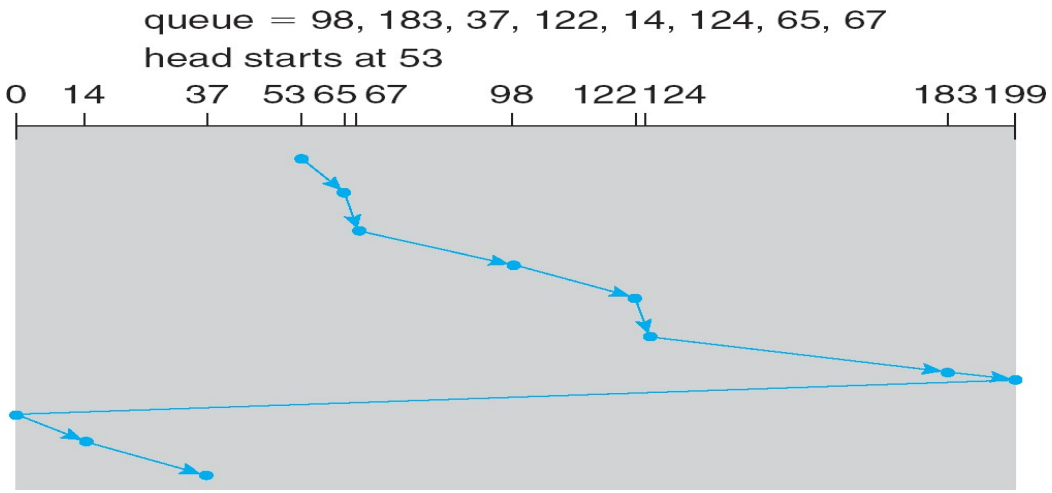
Cylinder Requests: 98, 183, 37, 122, 14, 124, 65, 67 ; Head pointer starts at 53



C-SCAN Algorithm

In this algorithm, the service is provided only in one direction and the head moves from the first cylinder to the last cylinder and serves the request only in one direction.

Example:



Input: (FOR FCFS)

Number of heads: 200

Cylinder requests: 7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

Current head position: 53

Output:

Cylinder Serving Order: 53 -> 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67

Total Cylinder movement: 640

Session # 13

File Management

Objective: Simulate file Allocation strategies: **Contiguous /Non Contiguous file space allocation**

IDE: C/C++/Java

Contiguous allocation (Sequential)

It requires each file to occupy a set of contiguous blocks on the hard disk where disk address define a linear ordering on the disk.

Algorithm

Create a one dimension array of size n (n is the number of blocks in the disk)
Read a filename from the user that is about to be created and allocated blocks
Read the file-size (S - number of blocks)
Now, Create a unique number for the file.
Read your One dimension array to look for a free space of S number of contiguous blocks.
If available, allocate the file those blocks and mark in your array
Continue to read q number of files in the same way
If, space cannot be allocated, inform the user that, the file could not be created
Now, read a file name from the user and
show its occupied blocks or 'NA' if the file does not exists.

Input/Output:

Enter total number of blocks : 200

Enter Filename: A
Enter File-Size: 4 blocks
File A created

Enter Filename: B
Enter File-Size: 400 blocks
File B cannot be created (not enough free blocks)

Enter Filename: C
Enter File-Size: 40 blocks
File C created

..
..
Search Filename : A
File Found in the blocks : 5, 6, 7, 8

Search Filename : D
File not Found.

Non Contiguous allocation

A file may be allocated any free blocks which may not be contiguous but free.

Algorithm

Create a one dimension array of size n (n is the number of blocks in the disk)
Read a filename from the user that is about to be created and allocated blocks
Read the file-size (S - number of blocks)
Now, Create a unique number for the file.
Read your one dimension array to look for a free space of S number of contiguous or non-contiguous blocks.
If available, allocate the file those blocks and mark in your array
Continue to read q number of files in the same way
If, space cannot be allocated, inform the user that, the file could not be created
Now, read a file name from the user and show its occupied blocks or 'NA' if the file does not exists.

Input/Output:

Enter total number of blocks : 200

Enter Filename: A
Enter File-Size: 4 blocks
File A created

Enter Filename: B
Enter File-Size: 400 blocks
File B cannot be created (not enough free blocks)

Enter Filename: C
Enter File-Size: 40 blocks

File C created

..
..
Search Filename : A
File Found in the blocks : 1, 9, 10, 20

Search Filename : D
File not Found.

NB: You may create any additional data structure to preserve any necessary information.

MID TERM EXAMINATION

There will be a 40-minutes written mid-term examination. Different types of questions will be included such as MCQ, mathematics, writing code fragments etc.

FINAL TERM EXAMINATION

There will be a written examination. Different types of questions will be included such as MCQ, mathematics, write a program etc.