SAVITRIBAI PHULE PUNE UNIVERSITY

T. Y. B. Sc. (Computer Science)

Laboratory Course I

System Programming & Operating System (CS347)

Semester II
(From Academic Year 2015-16)

Roll No. :______________

Name : ____

College :__________________________Division : _______

Academic Year :_______________

PROF.DR.VILAS KHARAT
Chairman Board of Study, Computer Science

PROF.DR.SHAILAJA C. SHIRVAIKAR
Co-Ordinator
Member Board of Study, Computer Science

PROF. NILESH B. MAHAJAN
Co-Ordinator

PREPARED BY :

Mr. Nilesh B. Mahajan (Bytco College, NashikRoad)

Mrs. Maduhri S. Ghanekar (Modern College,Pune)

Mr. Madhukar N. Shelar (KTHM College, Nashik)

About The Work Book Objectives –

- The scope of the course.
- Bringing uniformity in the way course is conducted across different Colleges.
- Continuous assessment of the students.
- Providing ready references for students while working in the lab.

How to use this book?

This book is mandatory for the completion of the laboratory course. It is a Measure of the performance of the student in the laboratory for the entire duration of the course.

Instructions to the students

- Students should carry this book during practical sessions of Computer Science.
- Printouts of the source code and output is not compulsory but optional
- Students should read the topics mentioned in reading section of this Book before coming for practical.
- Students should solve all exercises which are selected by Practical in-charge as a part of journal activity.
- Students will be assessed for each exercise on a scale of 5

1) Not done	0
2) Incomplete	1
3) Late complete	2

4) Needs improvement	3
5) Complete	4
6) Well-done	5

Instructions to the practical in-charge

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
- Choose appropriate problems to be solved by student.
- After a student completes a specific set, the instructor has to verify the outputs and sign in the provided space after the activity.
- Ensure that the students use good programming practices.
- You should evaluate each assignment carried out by a student on a scale of 5 as specified above ticking appropriate box.
- The value should also be entered on assignment completion page of respected lab course.

System Programming and Operating System (Semester-II)

Sr.No.	Assignment Name	Marks (out of 5)	Sign
1	Extended Shell (Toy Shell)		

2	CPU Scheduling		
3	Banker's Algorithm		
4	Demand Paging		
5	File Allocation Methods		
	Total Out of 25		

Assignment 1 : Extended Shell (Toy Shell)

What is Shell?

Shell is an interface between user and operating system. It is the **command interpreter**, which accept the command name (program name) from user and executes that command/program. The command interpreter in Unix/Linux is called as **Shell**. In Unix core operating system is also called as **kernel**.

Shell mostly accepts the commands given by user from keyboard. Shell gets started automatically when Operating system is successfully started.

There are various types of shell available on Linux viz. **BASH** (Bourn Again Shell), **CSH** (C Shell), **KSH** (Kourn Shell), **TCSH** it is enhanced C Shell.

When shell is started successfully it generally display some prompt to indicate the user that it is ready to accept and execute the command.

Shell executes the commands either **synchronously** or **asynchronously**. When shell accepts the command then it locates the program file for that command, start its execution, wait for the program associated with the command to complete its execution and then display the prompt again to accept further command. This is called as Synchronous execution of shell. In asynchronous execution shell accept the command from user, start the execution of program associated to the given command but does not wait for that program to finish its execution, display prompt to accept next command.

Objective of this practical assignment is to understand how the shell accepts and interpret the commands given by user and how it executes them and try to simulate our shell in similar manner.

How Shell Execute the command?

Following are the steps in order how shell execute the given command.

- Accept the command from user.
- Tokenize the different parts of command.
- First part given on the given command line is always a command name.
- **Creates (Forks)** a child process for executing the program associated with given command.
- Once child process is created successfully then it **loads (exec)** the binary (executable) image of given program in child process area.
- Once the child process is loaded with given program it will start its execution while shell is **waiting (wait)** for it(child) to complete the execution. Shell will wait until child finish its execution.
- Once child finish the execution then Shell wakeup, display the command prompt again and accept the command and continue.

Ex.

```
$ cat  f1.dat  f2.dat  f3.dat
```

This command line has four parts 1st is "cat", 2nd f1.dat, 3rd f2.dat and 4th f3.dat. First part is the command name which is to be executed.

Commands given to shell are of two types

1) System command

2) Internal commands which are internally coded and implemented by shell.

Additional commands (Extended Commands) to be interpreted and implemented in shell are :

1) **count**

It count and display the number of lines, words and characters in a given file.

2) **typeline**

It will display the all or number of lines in given file.

3) **list**

It will list the files in current directory with some details of files

4) **search**

It will allow searching the file for the occurrence of given string/pattern

Implementation of Toy Shell

To implement the shell knowledge of following **System calls** is essential

fork :

This system call can be used by any process to create the new process called as child process. The process that creates new process is called as **Parent Process** while newly created process is called as **Child Process**. By default child process is exact copy of parent process. Syntax of this system call is

int fork()

This system call is available in library file <**unistd.h>** . This system call returns following values.

1) returns the value 0 to the newly created child process

2) returns Process ID of child process to parent process.

3) returns -1 if fork fail to create child process or on error.

Both parent and child processes continue executing with the instruction that follows the call to fork. Fork system call is often followed by exec call.

exec :

Fork creates the new process which is exact copy of parent process. Call the exec will allow a child process to execute another program. When a process call exec function, that process is completely replaced by the new program and new program starts its execution from main. That is exec replaces the current process by new program and starts its execution.

There are several variations of exec function defined in library file **<unistd.h>**, as

int execl(char *filepath, char *arg0, char *arg1, ...,(char *)0);

int execv(char *filepath, char *argv[ ]);

int execlp(char *filename, char *arg0, char *arg1, ...,(char *)0);

int execvp(char *filename, char *argv[ ]);

First parameter can be complete path or a program name of the program to be executed.

arg0, arg1, …. are the list of parameters which will be passed as argument to that program. Instead of passing individual arguments we can pass array of arguments call as argument vector (argv[ ])

waitpid :

By using this system call it is possible for parent process to synchronize its execution with child process. Kernel will block the execution of a Process that calls waitpid system call if a some child of that process is in execution. It returns immediately when child has terminated by return termination status of a child. Syntax :

int waitpid(int pid, int *status, int options);

This function is defined in library fil **<sys/wait.h>**. In our toy shell when shell fork a child process and exec the given program in child process area then it call waitpid so that child process will execute and shell will wait for it to finish its execution. Once given command/program finish its execution it will send signal to parent so that it will finish it wait and continue the execution.

Program Logic

1) Main function will execute and display the command prompt as $

2) Accept the command at $ prompt from user.

3) Separate the different parts of command line.

4) Check that first part is one of the extended commands or not.

5) If the command is extended command then call corresponding functions which is implementing that command

6) Otherwise fork a new process and then load (exec) a program in that newly created process and execute it. Make the shell to wait until command finish its execution.

7) Display the prompt and continue until given command is “q” to Quit.

Slot 1

- Answer the following questions after carefully reading the description and program structure.

- What is Shell and which different shells are available on Linux/Unix platform?

- What is synchronous and asynchronous execution of shell?

- What is Process Id?

- What is the use of fork system call and what are the values retuned by it?

- What are the arguments to the execvp system call?

- What is the use of waitpid system call?

ii. Implement the toy shell program that accepts the command at $ prompt displayed by your shell (myshell$). Tokenize the command line and execute the given command by creating the child process. Also implement the additional command count as

myshell$ count c filename

It will display the number of characters in given file

myshell$ count w filename

It will display the number of words in given file

myshell$ count l filename

It will display the number of lines in given file

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Slot 2

Extend the shell to implement the commands as search, typeline, list

myshell$ search f filename pattern

It will search the first occurance of pattern in the given file

myshell$ search a filename pattern

It will search all the occurance of pattern in the given file

myshell$ search c filename pattern

It will count the number of occurance of pattern in the given file

myshell$ typeline n filename

It will display first n lines of the file.

myshell$ typeline - n filename

It will display last n lines of the file.

myshell$ typeline a filename

It will display all the lines of the file.

myshell$ list n dirname

It will display filenames in a given directory.

myshell$ list n dirname

It will count the number of entries in a given directory.

myshell$ list i dirname

It will display filenames and their inode number for the files in a given directory.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Assignment 2 : CPU Scheduling

CPU Scheduling is one of the important is one of the important task performed by the operating system. In multiprogramming operating systems many processes are loaded into memory for execution and these processes are sharing the CPU and other resources of computer system.

Scheduler is a program that decides which program will execute next at the CPU or which program will be loaded into memory for execution.

CPU scheduler is a program module of an operating system that selects the process to execute next at CPU out of the processes that are in memory. This scheduler is also called as Short term scheduler or CPU Scheduler

There are 3 types of schedulers as

1) Short Term Scheduler

2) Long Term Scheduler

3) Medium Term Scheduler

We have to implement various Short Term Scheduling algorithm as a part of this practical assignment.

Various CPU Scheduling algorithms are :

1) First Come First Serve (FCFS)

2) Shortest Job First (SJF)

3) Priority Scheduling

4) Round Robin Scheduling (RR)

These scheduling algorithms are further classified into 2 types as **Preemptive** and **Non-Preemptive**. FCFS scheduling is always Non-Preemptive while Round Robin is always Preemptive, while Shortest Job First and Priority Scheduling can be preemptive or non-preemptive.

The performance of various scheduling algorithms is compared on the basis of following criteria called as **scheduling criteria's** as

1) CPU Utilization	**2) Throughput**	**3) Turnaround Time**
4) Waiting Time	**5) Response Time**	

Data Structure

To simulate the working of various CPU scheduling algorithms following data structure is required.

1) Ready Queue : It represents the queue of processes which ready to execute but waiting for CPU to become available. Scheduling algorithm will select appropriate process from the ready queue and dispatch it for execution. For FCFS and for Round Robin this queue is strictly operated as **First In First Out queue**. But for Shortest Job First and Priority Scheduling it will operate as **Priority Queue**.

2) Process Control Block : It will maintain various details about the each process as processID, CPU-Burst, Arrival time, waiting time, completion time, execution time, turnaround time, etc. A structure can be used to define all these fields for processes and we can use array of structures of size n. (n is number of processes)

1) First Come First Serve Scheduling (FCFS) :

In this algorithm the order in which process enters the ready queue, in same order they will execute on the CPU. This algorithm is simple to implement but it may be worst several times.

2) Shortest Job First (SJF) :

In this algorithm the jobs will execute at the CPU according their next CPU-Burst time. The job in a ready queue which has shortest next CPU burst will execute next at the CPU. This algorithm can be preemptive or non-preemptive.

3) Priority Scheduling (PS) :

In this algorithm the job will execute according to their priority order. The job which has highest priority will execute first and the job which has least priority will execute last at the CPU. Priority scheduling can be preemptive or non-preemptive.

4) Round Robin Scheduling (RR) : This algorithm is mostly used in time-sharing operating systems like Unix/Linux. This algorithm gives the fair change of execution to each process in system in rotation. In this algorithm each process is allowed to execute for some fixed time quantum. If process has the CPU-burst more than time quantum then it will execute for the given time quantum. But if it has CPU-burst less than the time quantum then it will execute for its CPU-burst time and then immediately release the CPU so that it can be assigned to other process. The advantage of this algorithm is that the average waiting time is less. It this algorithm ready queue is strictly operated as First In First Out queue. This algorithm is intrinsically preemptive scheduling algorithm.

Slot 1

- Answer the following questions after carefully reading the concept of CPU Scheduling.

- What is CPU Scheduling?

  ____

  ____

  ____

- Define the term ready queue and priority queue. Which scheduling algorithms operate the queue as priority queue?

  ____

  ____

  ____

- Define the terms turnaround time and waiting time.

  ____

  ____

____

- What is CPU-burst and I/O-burst?

  ____

  ____

  ____

- Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

- Write the program to simulate Non-preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Slot 2

- Write the program to simulate Preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

- Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Slot 3

- Write the program to simulate Preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

- Write the program to simulate Round Robin (RR) scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Also give the time quantum as input. Assume the fixed IO waiting time (2 units). The next

CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Assignment Evaluation

____

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Assignment 3 : Banker's Algorithm

Software Description - This algorithm is related to handling deadlocks. There are three methods to handle deadlocks.

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection

Banker's algorithm is a technique used for **Deadlock Avoidance**. A deadlock avoidance algorithm dynamically examines the resource allocation state of system to the processes to ensure that a **circular-wait condition never exists.** The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the processes. A system is **safe** if the system can allocate resources to each process in some order and still avoid a deadlock. i.e. a system is in a safe state only if there exists a **safe sequence**. A sequence of processes <P_1, P_2, …., P_n> is a safe sequence for the current allocation state if, for each P_i , the resources that P_i can still request can be satisfied by currently available resources plus the resources held by all the P_j, with $j < i$.

Resource allocation graph is a technique used for deadlock avoidance. But the resource allocation graph algorithm is not applicable to the system with multiple instances of each resource type. Banker's algorithm is applicable to such a system.

Banker’s algorithm : When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will keep the system in a safe state. If it will, then resources are allocated, otherwise the process must wait until some other process releases enough resources. This includes 2 algorithms

1) **resource-request algorithm**

2) **Safety Algorithm.**

Safety Algorithm :

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:

Work = Available

Finish [*i*] = *0* for *i* = 1,2,3, …, *n.*

2. Find and *i* such that both:

(a) *Finish* [*i*] = 0

(b) $Need_i$ *Work*

If no such *i* exists, go to step 4.

3. $Work=Work+Allocation_i$
   Finish[*i*]=1
   go to step 2.

4. If *Finish* [*i*] == 1 for all *i*, then the system is in a safe state.

Resource-Request Algorithm for Process P_i :

Request = request vector for process P_i. If $Request_i$ [*j*] = *k* then process P_i wants *k* instances of resource type $R_{j.}$

1. If $Request_i$ $Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.

2. If $Request_i$ $Available$, go to step 3. Otherwise P_i must wait, since resources are not available.

3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$Available = Available - Request_i;$

$Allocation_i = Allocation_i + Request_i;$

$Need_i = Need_i – Request_i;$

- *If safe the resources are allocated to Pi.*
- *If unsafe Pi must wait, and the old resource-allocation state is restored*

The menu for the program should look like

1. Accept Allocation
2. Accept Max
3. Calculate Need
4. Accept Available
5. Display Matrices
6. Accept Request and Apply Banker’s Algorithm
7. Exit

Choose option by specifying corresponding integer

Data Structure Design:

Let n be the number of processes in the system and m be the number of resource types.

Component	**Description**	**'c' code**
Available	A vector of length m indicates the number of available resources of each type	int Avail[10]
Max	An n x m matrix which defines the maximum demand of each process	int Max[10][10]
Allocation	An n x m matrix that defines the number of resources of each type currently allocated to each process	int Alloc[10[10]
Need	An n x m matrix indicates the remaining resource need of each process.	int Need[10][10]
Request	A vector of length m indicates the request made by a process Pi	int Request[10]
Finish	Vector of length n to check whether the process finished or not	int Finish[10] ={0}
Work	Vector of length m which is initialized to Available	int Work[10]
Safe	Vector of length n initialized to 0	Int Safe[10]= {0};

Control structure – The design is modular. The main module performs necessary Variable declarations, displays menu and accepts option from user. The structure diagram is as follows.

Procedural Design – The following is a table of the guidelines for accepting inputs, calculated need and updating availables, etc as per Banker's Algorithm and implementation hints.

Procedure	Description	Programming hints

Bankers main	First accept number of processes and number of resources.
	In a loop
	Print options
	Read an option
	Branch depending on option
		Printf("enter no. of processes & no. of resources ");
		scanf("%d%d", &n,&m);
		While (option != 9)
		{ printoptions();
		   scanf("%d",&option);
		switch(option)
		{case 1: ……
		}

Accept_matrix (int A[][10])	A generalized procedure which will accept all required matrices, like Allocation, Max depending upon which matrix is passed as parameter	Void accept_matrix(int A[][10])
		{ int I,j;
		for (i=0; i<n;i++)
		      for (j=0; j<m;j++)
		            scanf("%d" ,&A[i][j]) ;
		}

Accept_available	Accept single dimentional array/vector Available	Void Accept_available()
		{ int I;
		  for (i=0; i<m; i++)
		            scanf("%d" ,&Avail[i]) ;
		}

Display_matrix	Display Allocation, Max, Need	Void Display_matrix()
		{….
		Printf (\n\tAllocation\t\tMax\t\tNeed\n”);
		for (i=0; i<n;i++)
		 { for (j=0; j<m;j++)
		            printf(“%d”,Allocl[i][j]) ;
		        printf(“\t);
		        for (j=0; j<m;j++)
		            printf(“%d”,Maxil[i][j]) ;
		        printf(“\t);
		        for (j=0; j<m;j++)
		            printf(“%d”,Need[i][j]) ;
		        printf(“\t);
		}
		Printf(“\n Available\n”);
		for (j=0; j<m;j++)
		            printf(“%d”,Avail[i][j]) ;
		        printf(“\t);
		}

Find_Need	Calculate Need matrix	Void Find_Need()
		{ ….
		for (i=0; i<n;i++)
		     for (j=0; j<m;j++)
		          Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*];
		}

Accept_Request	Accept the request for a process Pi	Void Accept_Request()
		{ int i;
		Printf(“\nenter process no:”);
		Scanf(“%d”,&proc);
		  for (i=0; i<m; i++)
		            scanf(“%d” ,&Requestl[i]) ;
		}

Bankers_algo	Invoke resource_request_algo and from it safety_algo	Bankers_algo()
		{
		resource_request_algo ();
		 }

Safety-algo	Find if a safe sequence exists	Safety_algo()
		{ int over =0, i, j, k,l=0, flag;

		//initialize work = available
		for (i=0; i<m; i++)
		    work[i] = Avail[i];
		
		while (!over)
		{ // check for any not finished process
		    for (i=0; i<n; i++)
		    {
		        if (Finish[i] ==0)
		        { flag =0;
		            pno=compare_need(i);
		            if(pno>-1)
		                break;
		        }
		    }
		    if (i ==n)
		    { printf("System is unsafe");
		            exit(1);
		    }
		
		    if (i < n && pno>=0)
		    {
		
		        for (k=0; k<m; k++)
		            work[k] += Alloc[pno][k];
		
		        Finish[pno] = 1;
		        Safe[l++] = pno; //add process in
		safe sequence
		
		        if (l >= n) // if all processes over
		            { // print safe sequence
		            printf("\n Safe sequence is :");
		            for(l=0;l<n; l++)
		                printf("P%d\t",Safe[l]);
		            over = 1;
		            }}
		    }
		}

Int compare_need (int p)	Checks if need of process p is <= work or not. If condition	Int compare_need(int p)
		{
		  int i,j, flag=0;

	true returns process no else returns -1	for (j=0;j< nor; j++)
		{
		if (Need[p][j] > work[j])
		{ flag =1;
		break; }
		
		}
		if(flag==0)
		return p;
		return -1;
		}

Resource_request_algo	Check if the given request can be immediately granted or not	Resource_request_algo()
		{ …
		// check if Requesti <= $Need_i$
		for (i=0; i<m; i++)
		{
		if Request[i] > Need[proc][i]
		{ printf("error.. process exceeds its Max demand"); exit(1); }
		}
		
		// check if Reuest [i] <= Available
		for (i=0; i<m; i++)
		{ if Request[i] > Avail[i]
		{ printf("Process must wait , resources not available");
		exit(1); }
		}
		
		// pretend to have allocated requested recources
		for (i=0; i<m; i++)
		{
		Avail[i] = Avail[i] – Request[i];
		Alloc[proc][i] = Alloc[proc][i] + Request[i];
		Need[proc][i] = Need[proc][i] – Request[i]$_i$;
		}
		
		//run Safety algorithm to check whether safe sequence exists or not
		Safety_algo();

		}

Slot 1

- Answer the following questions after carefully reading the description and program structure.

- Give the names of Deadlock handling methods. In which method Banker’s algorithm technique is used?
  ____

  ____

  ____

- Give the names of the algorithms that are used in Bankers algorithm.
  ____

  ____

  ____

- What does Deadlock Avoidance method ensure?
  ____

  ____

  ____

- Partially implement the Menu driven Banker’s algorithm for accepting Allocation, Max from user

- Add the following functionalities in your program
  - Accept Available
  - Display Allocation, Max
  - Find Need and Display it
  - Display Available

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Slot 2

- Modify above program so as to include the following:
  - Accept Request for a process
  - Resource_request algorithm
  - Safety algorithm

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Assignment 4 : Demand Paging

Demand paging the one of the most commonly used method of implanting the **Virtual Memory Management scheme**. Virtual memory management scheme has the advantages as :

- System can execute the process of larger size than the size of available physical memory.
- Reduced disk IO operations.
- More numbers of processes can be loaded into memory.

- Performance of multi-programming and multi-tasking can be improved.

In this memory management scheme instead of loading entire process into memory, only required portion of each process is loaded in memory for execution. Different parts of physical memory of each process are brought into memory as and when required hence the name is demand paging.
Each process is logically divided into number of pages. Each page is of same size.
Physical memory (RAM) is also divided into number of equal size frames. Generally size of frame and a page is same. When process is to be loaded into memory its pages are loaded into available free frames.
Memory management scheme maintains the list of free frames.
It also maintains the page table for each process. Page table keep track of various frames allocated to each process in physical memory. That is page table is used to map the logical pages of a process to frames in physical memory.
Memory Management Unit (MMU) of computer system is responsible to convert the logical address in process to physical memory in frame.
In demand paging memory management scheme no page of any process in brought into memory until it not really required. Whenever page is required (demanded) then only it is brought into memory.

Page Falult :
When process requests some page during execution and that page in not available in physical memory then it is called as Page Fault.
Whenever page fault occurs Operating system check really it is a page fault or it is a invalid memory reference. If page fault has occurred then system try to load the corresponding page in available free frame.

Page Replacement :
When page fault occurs, system try to load the corresponding page into one of the available free frame. If no free frame is available them system try to replace one of the existing page in frame with new page. This is called as Page Replacement. Frame selected for page replacement is as victim frame. Page replacement algorithm of demand paging memory management scheme will decide which frame will be selected as victim frame. There are various page replacement algorithms as:

- **First In First Out Page Replacement (FIFO)**
- **Optimal Page Replacement(OPT)**

- **Least Recently Used Page Replacement (LRU)**
- **Most Recently Used Page Replacement(MRU)**
- **Most Frequently Used Page Replacement(MFU)**
- **Least Frequently Used Page Replacement(LFU)**
- **Second Change Page Replacement.**

Input to Page Replacement Algorithm :

- **Reference String :** It is the list of numbers which represent the various page numbers demanded by the process.
- **Number of Frames :** It represent the number of frames in physical memory.

Ex.
Reference String : 3, 6, 5, 7, 3, 5, 2, 5,7 ,3,2,4,2, 8, 3, 6

It means first process request the page number 3 then page number 6 then 5 and so on.

Data Structure :

- Array of memory frames which is used to maintain which page is loaded in which frame. With each frame we may associate the counter or a time value whenever page is loaded in that frame or replaced.
- Array of reference String.
- Page Fault Count.

Output :

The output for each page replacement algorithm should display how each next page is loaded in which frame.
Finally it should display the total number of page faults.

FIFO :

- In this page replacement algorithm the order in which pages are loaded in memory, in same order they are replaced.
- We can associate a simple counter value with each frame when a page is loaded in memory.
- Whenever page is loaded in free frame a next counter value is also set to it.
- When page is to be replaced, select that page which has least counter value.
- It is most simple page replacement algorithm.

OPT :

- This algorithm looks into the future page demand of a process.
- Whenever page is to be replaced it will replace that page from the physical which will not be required longer time.
- To implement this algorithm whenever page is to be replaced, we compare the pages in physical memory with their future occurrence in reference string. The page in physical memory which will not be required for longest period of time will be selected as victim.

LRU :

- This algorithm replaces that page from physical memory which is used least recently.
- To implement this algorithm we associate a next counter value or timer value with each frame/page in physical memory wherever it is loaded or referenced from physical memory. When page replacement is to be performed, it will replace that frame in physical memory which has smallest counter value.

MRU :

- This algorithm replaces that page from physical memory which is used most recently.
- To implement this algorithm we associate a next counter value or timer value with each frame/page in physical memory wherever it is loaded or referenced from physical memory. When page replacement is to be performed, it will replace that frame in physical memory which has greatest counter value.

MRU :

- This algorithm replaces that page from physical memory which is used most recently.
- To implement this algorithm we associate a next counter value or timer value with each frame/page in physical memory wherever it is loaded or referenced from physical memory. When page replacement is to be performed, it will replace that frame in physical memory which has greatest counter value.

MFU :

- This algorithm replaces that page from physical memory which is used most frequently.
- This algorithm is bit complex to implement.
- To implement this algorithm we have to maintain the history of each page that how many times it is used (frequency count) so far whenever it is loaded in physical memory or referenced from physical memory. When page replacement is to be performed, it will replace that frame in physical memory of which frequency count is greatest.

- If frequency count is same for two or more pages then it will apply FCFS.

LFU :

- This algorithm replaces that page from physical memory which is used most frequently.
- This algorithm is bit complex to implement.
- To implement this algorithm we have to maintain the history of each page that how many times it is used (frequency count) so far whenever it is loaded in physical memory or referenced from physical memory. When page replacement is to be performed, it will replace that frame in physical memory of which frequency count is smallest.
- If frequency count is same for two or more pages then it will apply FCFS.

Second Chance Page Replacement :

- This algorithm is also called as Clock replacement policy.
- In this algorithm frames from physical memory are consider for the replacement in round robin manner. A page that has been accessed in two consecutive considerations will not be replaced.
- This algorithm is a extension of FIFO page replacement.
- To implement this algorithm we have to associate second chance bit to each frame in physical memory. Initially when a page is loaded in memory its second chance bit is set to 0. Each time a memory frame is referenced set the second chance bit to 1.When page replacement it to be performed access the memory frames in round robin manner. If second chance bit is 1 then reset it to 0. If second chance bit is zero then replace the page with that frame.
- This algorithm gives far better performance than FCFS page replacement.

Slot 1

- Answer the following questions after carefully reading the concept of Virtual memory, demand paging and page replacement algorithm examples.
    - Define the terms Virtual Memory and Demand Paging.

      ____

      ____

      ____

____

- What is page fault and page replacement?
  ____

  ____

  ____

- Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

  Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

  - Implement FIFO
  - Implement LRU

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Slot 2 :

- Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

  Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

  - Implement OPT
  - Implement MFU

- Implement LFU

Slot 3 :

- Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

  Reference String : 2,5,2,8,5,4,1,2,3,2,6,1,2,5,9,8

  - Implement MRU
  - Implement Second Chance Page Replacement.

- Attempt each of the above algorithm with different reference string.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/_____**

Assignment 5 : File Allocation Methods

Files are managed by the operating system on secondary storage devices. Hard disk drive is most commonly used storage device.
Physically hard disk is divided into number **sectors** also called as **blocks**. When a file is to be created the free blocks on the disk are allocated to the file.
Operating system maintains the status of each block. **Status of disk block** can be

- **Free**

- **Allocated**
- **Reserved**
- **Bad**

Purpose of this practical assignment is to understand various file allocation methods and their implementation.
When user accesses any file, the system has to locate all the disk blocks of that file on disk. Hence operating system has systematic way to maintain, allocate the blocks for each file and releasing the blocks of file when it is deleted.
File Allocation Method is a method that specifies how the blocks allocated to file are maintained so that file can be accessed properly. Most commonly used file allocation methods are

- **Contiguous File Allocation**
- **Linked File Allocation**
- **Indexed File Allocation**

Contiguous File Allocation :

- This file allocation method is also called as Sequential file allocation.
- In this method blocks allocated the file are contiguous. That is if file is of n blocks then all the n blocks are adjacent to one another.
- Advantage of this method is that the file can access sequentially as well as randomly.
- Directory maintains the file name along with starting block number and length (that is number of blocks allocated to file).
- Drawback of this method is that there can be external fragmentation problem and file cannot grow efficiently.

Linked File Allocation :

- In this method n blocks file are maintained as chain (linked list) of n blocks. First block of file contains the address or link to second block of file. Second block of file contains the address or link to third of file as so on. Last block of file points to null. That any free block on the storage device can be allocated to the file.
- Advantage of this method is that the file can grow or shrink dynamically and there is no external fragmentation.
- Directory maintains the file name along with starting block number and last block number of a file.

- Drawback of this method is that the file can be access sequentially only. Random file accessing is not possible and wastage of memory for pointer in each block of a file.

Index File Allocation :

- This method overcomes the drawbacks of Sequential File Allocation method as well as Linked Allocation Method.
- In this method a special Index Block is maintained for each file. This block contains the list of block numbers allocated to the file. When file is to be accessed the index block will give the list of blocks allocated to that file.
- Hence in this method file can be accessed sequentially as well as randomly.
- Each available free block on disk can be allocated to any file on demand and that number is added to the index block of that file.
- Directory maintains the file name along with its index block number and number of entries in it.
- Drawback of this method is the overhead of maintaining index block.

Data Structures :

- **Bit Vector :**

  It is used to maintain the free space on disk. It is a vector of size n (where n is number of blocks on disk) with values 0 or 1. These values mark the status of block as whether it is free or allocated. For Ex. It bit vector is :
  00111010001101…….
  It means that first 2 blocks are allocated, block 3,4,5 are free, 6th is allocated, 7th free, 8th to 10th are not free, 11th and 12th are free and so on.

- **Directory :**

  Each directory entry maintains the filename along with certain details relevant to file allocation method. Directory list can be maintained statically or dynamically.
  For Contiguous allocation each directory entry contains filename, starting block number and length. Likewise for each file allocation method appropriate details are maintained in directory entry.

Implementation :

- A bit vector(array) of size n which is initialized randomly to 0 or 1.
- A menu with the options
  - **Show Bit Vector**
  - **Create New File**
  - **Show Directory**

    - **Delete File**
    - **Exit**
- When Create New File option is selected the program should ask the file name along with number of blocks to allocate to the file. According the free blocks should be searched by using the bit vector and allocated to the file. A new directory entry shall be added in directory list along with appropriate details.
- When Delete File option is selected the program should ask the file name. Release the blocks allocated to the file. Accordingly update the bit vector and remove the entry from directory.

Slot 1

- Answer the following questions after carefully reading the concept File Allocation Methods
    - What is File allocation method?

      ____

      ____

      ____

    - Define the term External Fragmentation in the context of file allocation methods.

      ____

      ____

      ____

    - What is index block?

      ____

      ____

- Write a program to simulate Sequential(Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**

Slot 2

- Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option.

- Write a program to simulate Indexed file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion ____/____/______**