LAB MANUAL

310248: Laboratory Practice I (SYSTEM PROGRAMMING& OPERATING SYSTEM)

CLASS: TE2019Pattern

Prepared by

Prof. Neelam Ashish Jadhav

Sr. No.	Title	Page No	Date of Conduction	Date of Submission
	GROUP-A			
1	Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) Should be input for Pass-II.			
2	Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass- I (MNT, MDT and intermediate code file without any macro definitions) should be in put for Pass-II.			
	GROUP-B			
3	Program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority and Round Robin (Non-Preemptive).			
4	Write a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.			

GROUP-A

LAB ASSIGNMENT NO.: 01

Title:

Design suitable data structures and implement pass-I and Pass-II of a two-pass assembler for pseudo-machine in Java. Implementation should consist of a few instructions from each category and few assembler directives.

Objectives :

- To understand Data structure of Pass-1 assembler
- To understand Pass-1 assembler concept
- To understand Advanced Assembler Directives

Problem Statement :

Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine in Java using object oriented feature.

Outcomes:

After completion of this assignment students will be able to:

- Implemented Pass – 1 assebmler
- Implemented Symbol table, Literal table & Pool table.
- Understood concept Advanced Assembler Directive.

Theory Concepts:

Introduction :-

There are two main classes of programming languages: *high level* (e.g., C, Pascal) and *low level*. *Assembly Language* is a low level programming language. Programmers code symbolic instructions, each of which generates machine instructions.

An *assembler* is a program that accepts as input an assembly language program (source) and produces its machine language equivalent (object code) along with the information for the loader.

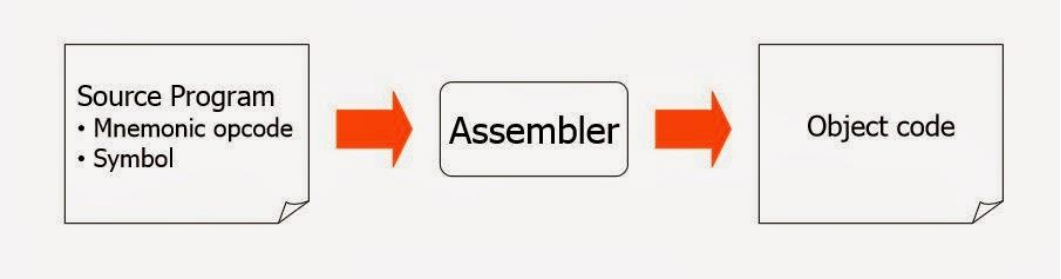


Figure 1.Executable program generation from an assembly source code

Advantages of coding in assembly language are:

- Provides more control over handling particular hardware components
- May generate smaller, more compact executable modules
- Often results in faster execution

Disadvantages:

- Not portable
- More complex
- Requires understanding of hardware details (interfaces)

Pass – 1 Assembler:

An assembler does the following:

1. Generate machine instructions
   - evaluate the mnemonics to produce their machine code
   - evaluate the symbols, literals, addresses to produce their equivalent machine addresses
   - convert the data constants into their machine representations
2. Process pseudo operations

Pass – 2 Assembler:

A two-pass assembler performs two sequential scans over the source code:

Pass 1: symbols and literals are defined
Pass 2: object program is generated

Parsing: moving in program lines to pull out op-codes and operands

Data Structures:

- **Location counter (LC):** points to the next location where the code will be placed
- **Op-code translation table**: contains symbolic instructions, their lengths and their op-codes (or subroutine to use for translation)
- **Symbol table (ST):** contains labels and their values
- **String storage buffer (SSB):** contains ASCII characters for the strings
- **Forward references table (FRT):** contains pointer to the string in SSB and offset where its value will be inserted in the object code

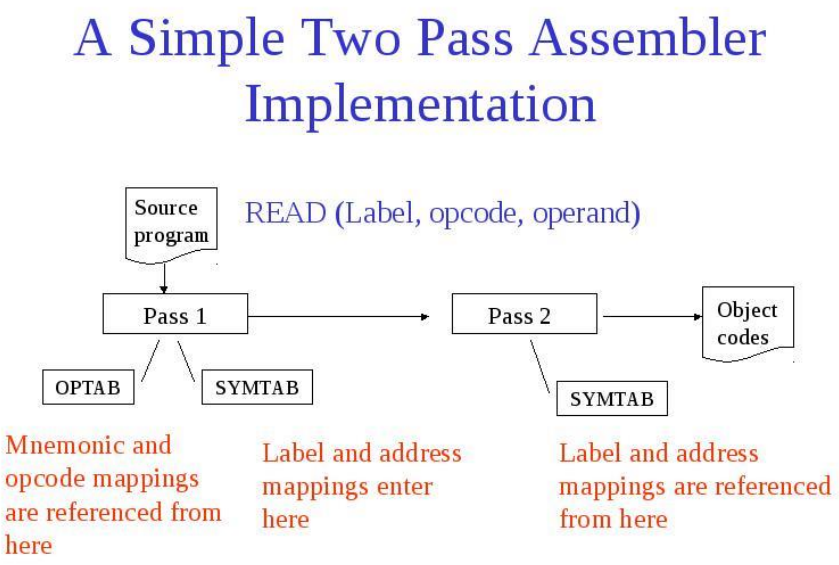


Figure 2. ***A simple two pass assembler.***

Elements of Assembly Language :

An assembly language programming provides three basic features which simplify programming when compared to machine language.

1. Mnemonic Operation Codes :

Mnemonic operation code / Mnemonic Opcodes for machine instruction eliminates the need to memorize numeric operation codes. It enables assembler to provide helpful error diagnostics. Such as indication of misspelt operation codes.

2. Symbolic Operands :

Symbolic names can be associated with data or instructions. These symbolic names can be used as operands in assembly statements. The assembeler performes memory bindinding to these names; the programmer need not know any details of the memory bindings performed by the assembler.

3. Data declarations :

Data can be declared in a variety of notations, including the decimal notation. This avoids manual conversion of constants into their internal machine representation, for example -5 into (11111010)2 or 10.5 into (41A80000)16

4. Statement format :

An assembly language statement has the following format :

[Label] <Opcode> <operand Spec> [, operand Spec> ..]

Where the notation [..] indicates that the enclosed specification is optional.

Label associated as a symbolic name with the memory word(s) generated for the statement

Mnemonic Operation Codes :

Instruction Opcode	Assembly Mnemonic	Remarks
00	STOP	Stop Execution
01	ADD	Op1 ← Op1+ Op2
02	SUB	Op1 ← Op1 – Op2
03	MULT	Op1 ← Op1* Op2
04	MOVER	CPU Reg ← Memory operand
05	MOVEM	Memory ← CPU Reg
06	COMP	Sets Condition Code
07	BC	Branch on Condition
08	DIV	Op1 ← Op1/ Op2
09	READ	Operand 2 ← input Value
10	PRINT	Output ← Operand2

Fig: Mnemonic Operation Codes

Instruction Format :

Instruction Format

Pass Structure of Assembler :

One complete scan of the source program is known as a pass of a Language Processor.Two types 1) Single Pass Assembler 2) Two Pass Assembler.

Single Pass Assembler :

First type to be developed Most Primitive Source code is processed only once.

The operand field of an instruction containing forward reference is left blank intiallyEg) MOVER BREG,ONE

Can be only partially synthesized since ONE is a forward reference

During the scan of the source program, all the symbols will be stored in a table called **SYMBOL TABLE.** Symbol table consists of two important fields, they are symbol name andaddress.

All the statements describing forward references will be stored in a table called Table ofIncompleted Instructions (TII)

TII (Table of Incomplete instructions)

Instruction Address	Symbol
101	ONE

By the time the END statement is processed the symbol table would contain the

address of allsymbols defined in the source program.

Two Pass Assembler :

Can handle forward reference problem easily.

First Phase : *(Analysis)*

- Symbols are entered in the table called Symbol table
- Mnemonics and the corresponding opcodes are stored in a table called Mnemonic table
- LC Processing

Second Phase : *(Synthesis)*

- Synthesis the target form using the address information found in Symbol table.
  - First pass constructs an Intermediated Representation (IR) of the source program for use by the second pass.

Data Structure used during Synthesis Phase :

1. Symbol table
2. Mnemonics table
3.

ADVANCED ASSEMBLER DIRECTIVES

1. ORIGIN
2. EQU
3. LTROG

ORIGIN :

Syntax : ORIGIN < address spec>

< address spec>can be an <operand spec> or constant

Indicates that Location counter should be set to the address given by < address spec>

This statement is useful when the target program does not consist of consecutive memory words.Eg) ORIGIN Loop + 2

EQU : *Syntax*

<symbol> EQU <address spec>

<address spec>operand spec (or) constant

Simply associates the name symbol with address specification

NoLocation counter processing is implied

Eg) Back EQU

Loop LTORG :

(Literal Origin)

Where should the assembler place literals ?

It should be placed such that the control never reaches it during the execution of a program.

By default, the assembler places the literals after the END statement.

LTROG statement permits a programmer to specify where literals should be placed.

Algorithms :

Flowchart : .

Conclusion :

Thus , I have implemented Pass-2 assembler by taking input as output of pass-1

LAB ASSIGNMENT NO.: 02

Title:

Design suitable data structures and implement pass-I and and Pass-II of a two-pass macro-processor using Java

Objectives :

- To understand Data structure of Pass-1 macroprocessor
- To understand Pass-1 macroprocessor concept
- To understand macro facility.

Problem Statement :

Design suitable data structures and implement pass-I of a two-pass macro-processor using OOP features in Java.

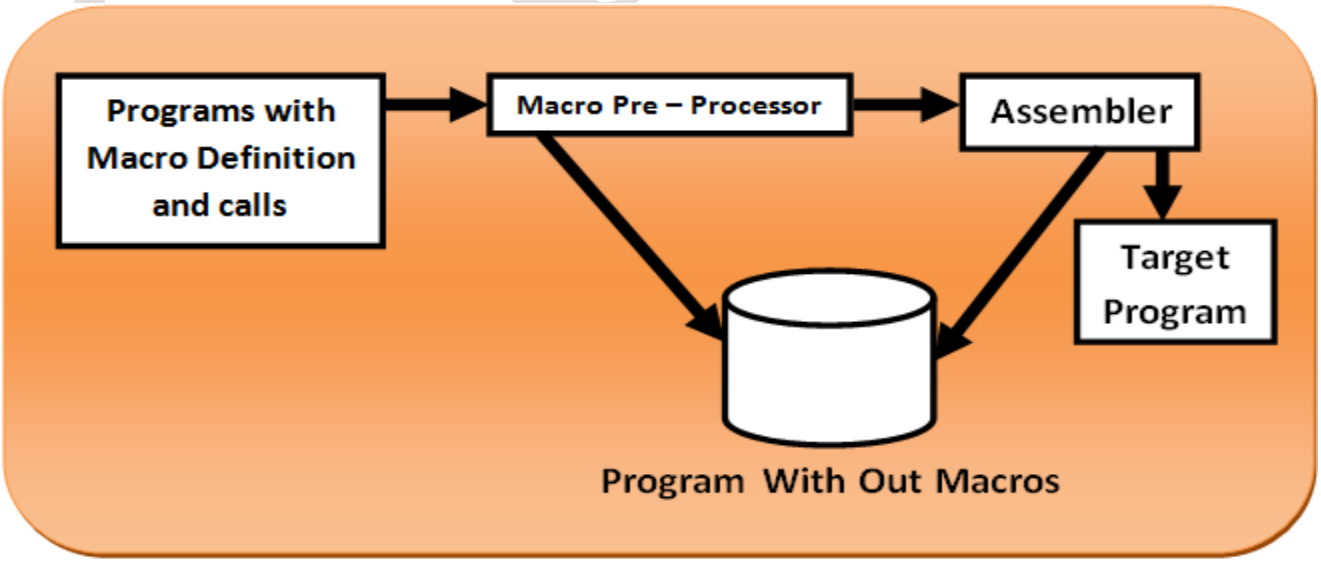
Outcomes:

After completion of this assignment students will be able to:

- Implemented Pass – 1 macroprocessor
- Implemented MNT, MDT table.
- Understood concept Pass-1 macroprocessor.

Theory Concepts:

Macroprocessor



Macro :

Macro allows a sequence of source language code to be defined once & then referred to by name each time it is referred. Each time this name occurs in a program , the sequence of codes is substituted at that point.

Macro has following parts:-

(1) Name of macro

(2) Parameters in macro

(3) Macro Definition

Parameters are optional.

Definition of Macro :-

Macro can be formatted in following order:-

Start of definition	MACRO
macro name	mymacro
macro body	ADD AREG, X ADD BREG, X
End of macro definition	MEND

'MACRO' pseudo-op is the first line of definition & identifies the following line as macroinstruction name.

Following the name line is sequence of instructions being abbreviated the instructions

comprising the 'MACRO' instruction.

The definition is terminated by a line with MEND pseudo-op.

Example of Macro:-

(1) Macro without parameters

```
MACRO
        mymacro
                ADD
                AREG,X
                ADD
                BREG,X
MEND
```

(2) Macro with parameters

```
MACRO
        addmacro  &A
                ADD
                AREG,&A
                ADD
                BREG,&A
MEND
```

The macro parameters (Formal Parameters) are initialized with '&'.used as it is in operation..Formal Parameters are those which are in definition of macro. Whereas while calling macros we use Actual Parameters.

How To Call a Macro?

A macro is called by writing macro name with actual parameters in an AssemblyProgram.

Macro call leads to Macro Expansion.

Syntax: <macro-name> [<list of parameters>]

Example:- for above definitions of macro…

(1) mymcro

(2) addmacro X

Macro Expansion:-

Each Call to macro is replaced by its body.

During Replacement, actual parameter is used in place of formal parameter.

- During Macro expansion, each statement forming the body of macro is picked up one bye one sequentially.
- Each Statement inside the macro may have:

  (1) An ordinary string, which is copied as it is during expansion.

  (2) The name of a formal parameter which is proceeded by character '&'.

- During macro expansion an ordinary string is retained without any modification. Formal Parameters(Strings starting with &) is replaced by the actual parameter value.

Example

- **Macro definition**
  - MACRO
  - MCALC &par1=, &par2=, &par3=MULT, &par4=
  - &par4
  - MOVER REG1, &par1
  - ADD REG1, &par2
  - MOVEM REG1, &par1
  - MEND
- **Macro call**
  - MCALC A, B, par4=loop
- **1. Expanded code**
  - +loop MOVER REG1, A
  - MULT REG1, B
  - MOVEM REG1, A

Macro with Keyword Parameters :-

These are the methods to call macros with formal parameters. These formal parameters are oftwo types

(1) Positional Parameters :

Initiated with '&'.

Ex:- mymacro &X

(2) Keyword Parameters :

Initiated with '&' . but has some default value.

During a call to macro , a keyword parameter is specified by its name.Ex:- mymacro &X=A

Nested Macro Calls :-

Nested Macro Calls are just like nested function calls in our normal calls. Only the transfer ofcontrol from one macro to other Is done.

Consider this example :-

```
MACRO
  Innermacro

      ADD AREG,X
MEND
MACRO
  outermacro
      innermacro
  ADD AREG,Y
MEND

outermacro
```

In this example, firstly the MACRO outermacro gets executed & then innermacro.

So Output will be Adding X & Y values in AREG register.

Algorithm:

Scan all MACRO definition one by one.

(a) Enter its name in macro name table (MNT).

(b) Store the entire macro definition in the macro definition table (MDT).

(c) Add the information in the MNT indicates where definition of macro can be found in MDT.

(d) Prepare argument list array (ALA).

Data Structures of Two Pass Macros:

1] Macro Name Table Pointer (MNTP) :

2] Macro Definition Table Pointer (MDTP) :

3] Macro Name Table :

- macro number(i.e pointer referenced from MNTP)
- Name of macros
- MDTP (i.e points to start position to MDT)

4] Macro Definition Table :

- Location Counter(where MDTP points to start position of macro)
- Opcode
- Rest (i.e it will contain the other part than opcodes used in macro).

5] Argument List Array :

- Index given to parameter
- Name of parameter

Algorithms :

Flowchart : .

Conclusion :

Thus , I have implemented Pass-2 macroprocessor by taking output of pass one as input to pass-2 (i.e.MDT and MNT table

GROUP-B

LAB ASSIGNMENT NO.: 03

Title:

Write a Java program (using OOP features) to implement following scheduling algorithms: FCFS,SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

Objectives :

- To understand OS & SCHEDULLING Concepts
- To implement Scheduling FCFS, SJF, RR & Priority algorithms
- To study about Scheduling and scheduler

Problem Statement :

Write a Java program (using OOP features) to implement following scheduling algorithms: FCFS,SJF, Priority and Round Robin .

Outcomes:

After completion of this assignment students will be able to:

- Knowledge Scheduling policies
- Compare different scheduling algorithms

Theory Concepts:

CPU Scheduling:

• CPU scheduling refers to a set of policies and mechanisms built into the operating systems that govern the order in which the work to be done by a computer system is completed.

• Scheduler is an OS module that selects the next job to be admitted into the system and next process to run.

Scheduling

Scheduling is defined as the process that governs the order in which the work is to be done. Scheduling is done in the areas where more no. of jobs or works are to be performed. Then it requires some plan i.e. scheduling that means how the jobs are to be performed i.e. order. CPU scheduling is best example of scheduling.

Necessity of scheduling

• Scheduling is required when no. of jobs are to be performed by CPU.

• Scheduling provides mechanism to give order to each work to be done.

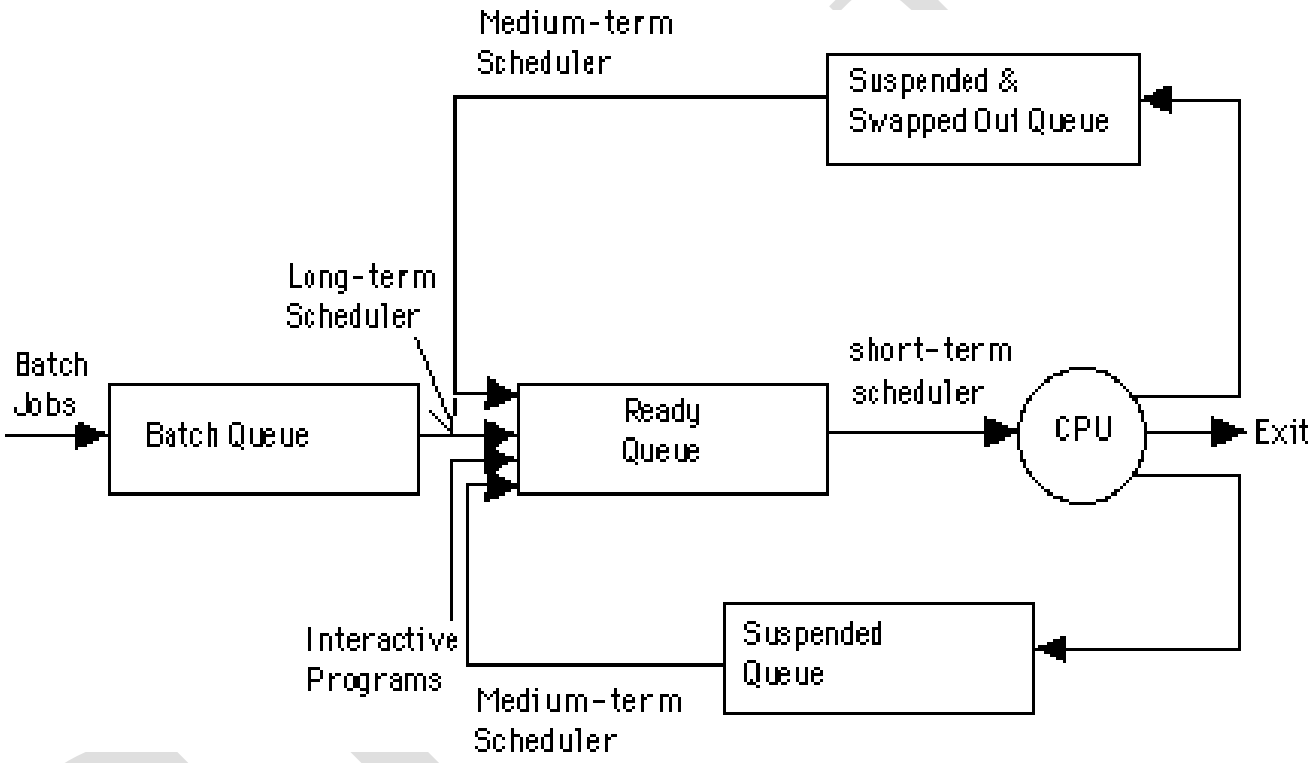
• Primary objective of scheduling is to optimize system performance.

• Scheduling provides the ease to CPU to execute the processes in efficient manner.

Types of schedulers

In general, there are three different types of schedulers which may co-exist in a complex operatingsystem.

- Long term scheduler
- Medium term scheduler
- Short term scheduler.



You Have to Write This all ans by yourself

- Difference Between Long term scheduler, Medium term scheduler& Short term scheduler?
- Scheduling Criteria?
- Difference Between Non-preemptive Scheduling & Preemptive Scheduling?
- Short Note on Types of scheduling Algorithms(with Example)
    - FCFS(First Come First Serve)
    - SJF(Short Job First)
    - Priority scheduling

- Round Robin Scheduling algorithm

2. Algorithms(procedure) :

FCFS :

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Set the waiting of the first process as '0' and its burst time as its turn around time

Step 5: for each process in the Ready Q calculate

Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SJF :

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the

CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting

according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

RR :

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time Step 4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

(e) Average waiting time = Total waiting Time / Number of process

(f) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process.

Priority Scheduling :

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time, priority

Step 4: Start the Ready Q according the priority by sorting according to lowest to highest burst time and process.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

Flowchart :

Note: you should draw flowchart as per algorithm/procedure as above

Conclusion:

Hence we have studied that-

- CPU scheduling concepts like context switching, types of schedulers, different timing parameter like waiting time, turnaround time, burst time, etc.
- Different CPU scheduling algorithms like FIFO, SJF,Etc.
- FIFO is the simplest for implementation but produces large waiting times and reduces system performance.
- SJF allows the process having shortest burst time to exec

LAB ASSIGNMENT NO.: 04

Write a program to simulate Page replacement algorithm..

1. *Objectives:*

- To understand Page replacement policies
- To understand paging concept
- To understand Concept of page fault, page hit, miss, hit ratio etc

2. *Problem Statement:*

Write a java program to implement Page Replacement Algorithm FIFO, LRU and OPT.

3. *Outcomes:*

After completion of this assignment students will be able to:

- Knowledge of Page Replacement Policies in OS
- Implemented LRU & OPT Page replacement Policies
- Understood concept to f paging.

4. *Software Requirements:*

Latest jdk., Eclipse

5. *Hardware Requirement:*

- M/C Lenovo Think center M700Ci3,6100,6thGen.H81, 4GBRAM,500GBHDD

6. *Theory Concepts:*

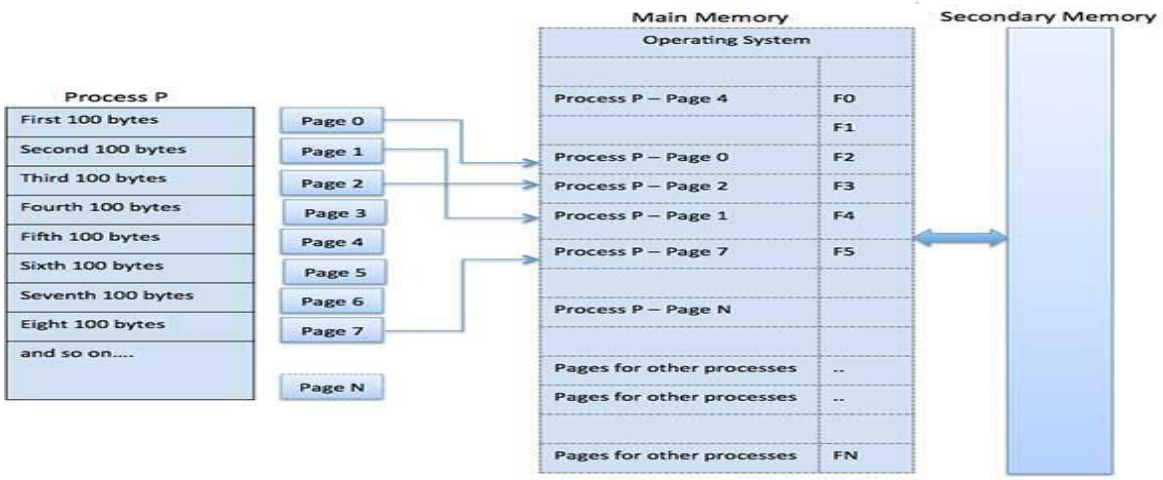
Paging:

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages**(size is power of 2, between 512 bytes and

8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

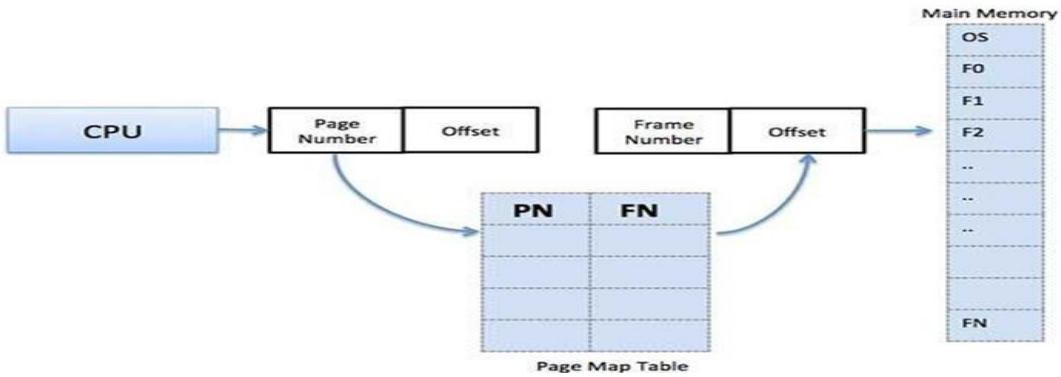
Page address called **logical address** and represented by **page number** and the **offset**.

LogicalAddress =Page number+pageoffset

Frame addressiscalled **physicaladdress** andrepresented byaframenumberandthe **offset**.

PhysicalAddress= Framenumber+page offset

A data structure called **page map table**is used to keep track of the relation between a page of a process to a frame in physical memory



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging−

- Paging reduces external fragmentation,but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

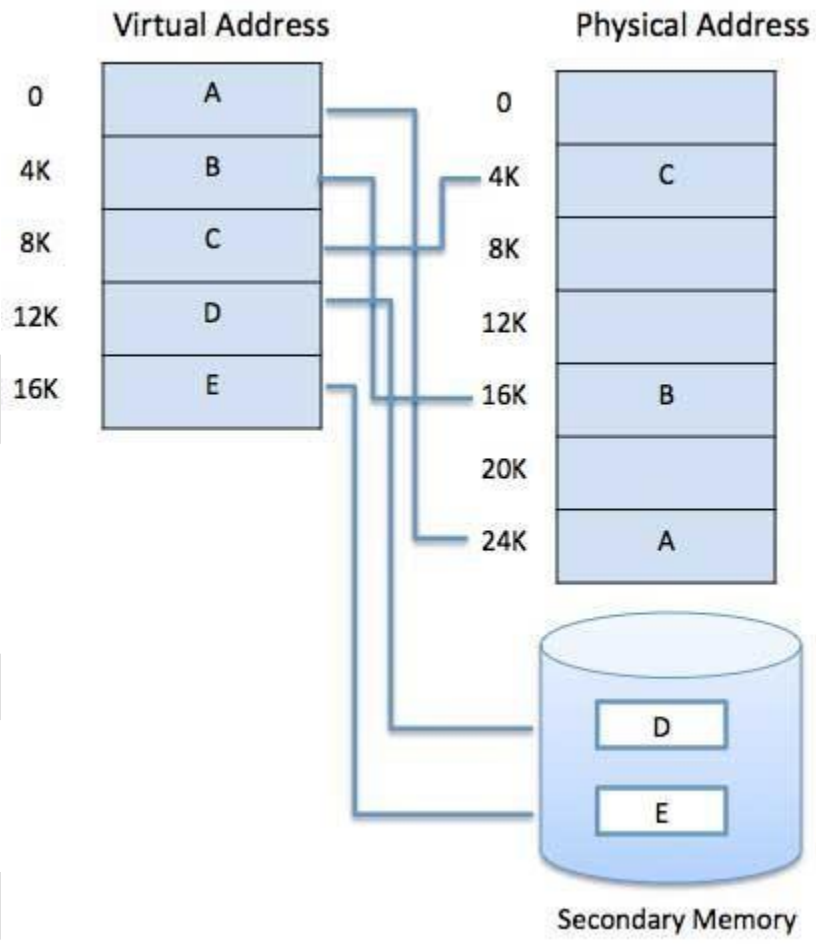
The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.

- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −

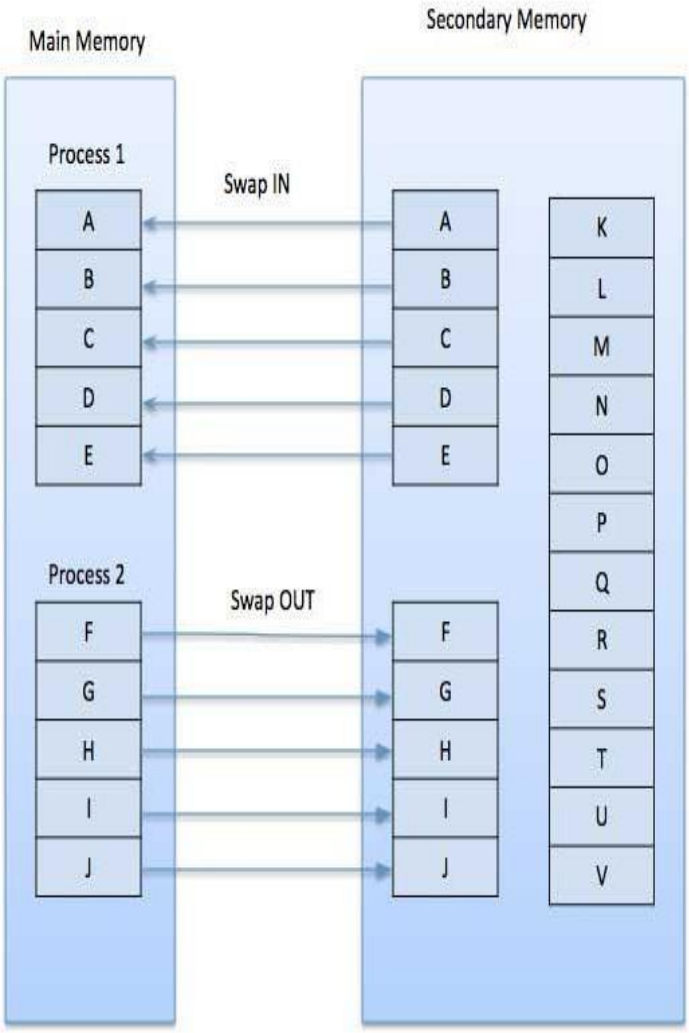


Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

DemandPaging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in

advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging–

- Large virtual memory.
- More efficient use of memory.
- Thereis nolimit ondegreeof multiprogramming.

Disadvantages

- Numberoftablesandtheamountofprocessoroverheadforhandlingpageinterruptsare greater than in the case of the simple paged management techniques.

Page Replacement Algorithm:

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Page fault:

A**pagefault**(some times called #PF, PF or hard **fault**)is a type of exception raised by computer hardware when a running program accesses a memory **page** that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process.

Page hit :

A **hit** is a request to a web server for a file, like a web **page**, image, JavaScript, or Cascading Style Sheet. When a web **page** is downloaded from a server the number of "**hits**" or "**page hits**" is equal to the number of files requested.

Page frame:

The **page frame** is the storage unit (typically 4KB in size)where as the **page** is the contents that you would store in the storage unit ie the **page frame**. For eg) the RAM is divided into fixed size blockscalled**pageframes**whichistypically4KBinsize, andeach

Page frame can store 4KB of data ie the **page**.

Page table:

A **page table** is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses.

Reference String:

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.

- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100,then the reference string is 1,2,6,12,0,0

First In First Out(FIFO)algorithm:

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Advantages

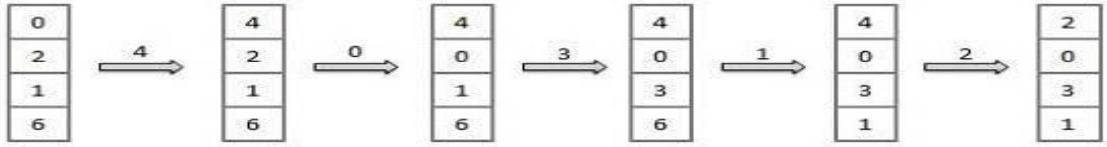
1. Simple to understand and implement
2. Does not cause more overhead

Disadvantages

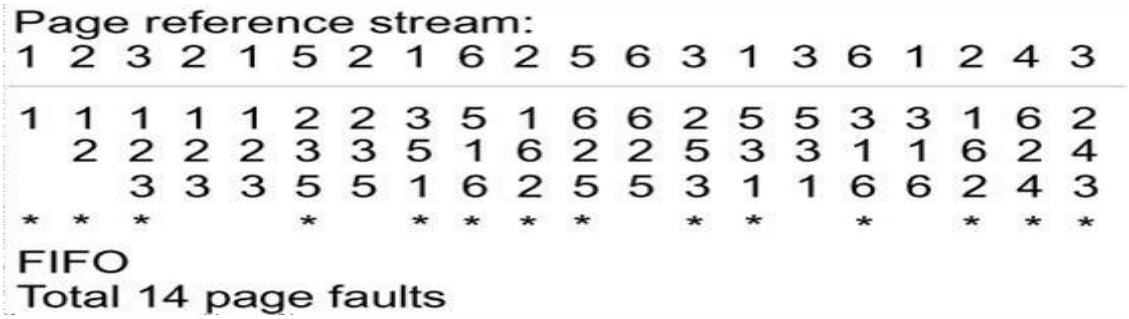
1. Poor performance
2. Doesn't use the frequency of the last used time and just simply replaces the oldest page.
3. Suffers from Be lady's anomaly.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 9 / 12 = 0.75



Note: you can take other example also. This just for reference .(you must calculate page fault, page hit and hit ratio

Least Recently Used(LRU)algorithm:

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Advantages

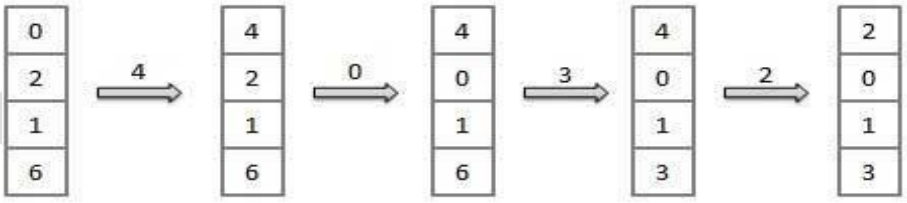
1. It is open for full analysis
2. Doesn't suffer from Be lady's anomaly
3. Often more efficient than other algorithms

Disadvantages

1. It requires additional data structures to be implemented
2. More complex
3. High hardware assistance is required

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page reference stream:

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1	1	3	2	1	5	2	1	6	2	5	6	6	1	3	6	1	2
	2	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4
		3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
*	*	*			*			*		*		*	*				*	*	*

LRU
Total 11 page faults

Optimal Page algorithm:

- An optimal page-replacement algorithm has the lowest page-faultrate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Advantages

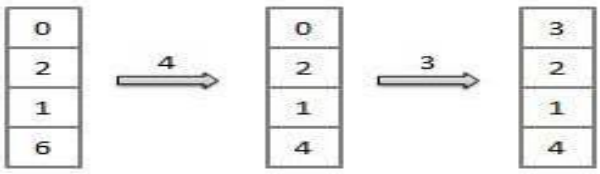
1. Excellent efficiency
2. Less complexity
3. Easy to use and understand
4. Simple data structures can be used to implement
5. Use das the bench mark for other algorithms

Disadvantages

1. More time consuming
2. Difficult for error handling
3. Need future awareness of the programs, which is not possible every time

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



Fault Rate = 6 / 12 = 0.50

Page reference stream:

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	2	2	2
	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	4	4
		3	3	3	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3
*	*	*			*			*				*	*				*	*	

Optimal
Total 9 page faults