### Assignment No 01

**TITLE :** To design a Pass-1 of two Pass assembler

**PROBLEM STATEMENT :** Design suitable data structures and implement pass-I of a two-pass assembler for pseudo- machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives.

**OBJECTIVE :** Design suitable data structures and develop a subset of an assembler Subset should consist of a few instructions from each category and few assembler directives.

**THEORY :** An assembler is a computer program for translating assembly language essentially, a mnemonic representation of machine language — into object code.

To make writing and reading programs written in machine language more convenient, mnemonic(symbol) were used for each machine instruction. These mnemonics were then later translated into machine language. Such a mnemonic language is called **assembly language**. Assemblers are program to automatically translate assembly language into machine language.

Source program

(in Assembly language)

Assembler

I/P O/P

Object program

(Machine language) and other information for the loader.

Data Bases

Other than producing the machine language, the assembler must also produce the information for the loader to use eg. externally defined symbols must be noted and passed to the loader.

For translation purpose assembler must do the following:

1. Generate instructions:
   1. Evaluate the mnemonic in the operation field to produce its machine code.
   2. Evaluate the subfields – find the value of each symbol, process literals, and assign addresses.
2. Process pseudo ops.

These tasks can be grouped into two passes or sequential scans over the input; associated with each task are one or more assembler modules.

Pass I : Purpose – define symbols and literals.

1. Determine length of machine instructions.
2. Keep track of Location Counter (LC).
3. Remember values of symbols until pass 2.
4. Process some psuedo ops, e.g., EQU,DS
5. Remember literals.

### Data structures:

Pass I data bases:

1. Input source program.
2. A Location Counter (LC), used to keep track of each instruction’s location.
3. A machine operation table (MOT), that indicates the symbolic mnemonic for each and its length.
4. A pseudo – operation table (POT) , that indicates the symbolic mnemonic and action to be taken for each pseudo \_ op in pass 1.
5. A symbol table (ST), that is used to store each literal encountered and its corresponding value.
6. A literal table (LT) that is used to store each literal encountered and its corresponding assigned location.
7. A copy of the input, to be passed to pass 2. This is to be stored in a secondary storage device.

### Algorithm of Pass I:

1. loc\_cntr:=0;(default value) pooltab\_ptr:=1;POOLTAB[1]:=1; littab\_ptr:=1;
2. While next statement is not an END statement
   1. If label is present then this\_label:=symbol in label field;

Enter(this\_label, loc\_cntr) in SYMTAB.

* 1. If an LTORG statement then
     1. Process literals LITTAB[POOLTAB[pooltab\_ptr]]…LIITLAB

[littab\_ptr-1] to allocate memory and put the address in the address field.Update loc\_cntr accordingly.

* + 1. Pooltab\_ptr :=pooltab\_ptr+1; (iii)POOLTAB[pooltab\_ptr]:littab\_ptr;
  1. If a STSRT or ORIGIN statement then loc\_cntr :=value specified in operand field;
  2. If an EQU statement then
     1. this\_addr :=value of <address spec>;
     2. Correct the symtab entry for this\_label to (this\_label,this\_addr).
  3. If a declaration statement then
     1. code :=code of the declaration statement;
     2. size :=size of memory area required by DC/DS.
     3. loc\_cntr :=loc\_cntr+size;
     4. Generate IC ‘(DL,code)…’.
  4. If an imperative statement then
     1. code:=machine opcode from OPTAB;
     2. loc\_cntr:=loc\_cntr+instruction length from OPTAB;
     3. If operand is a literal then

this\_literal:=literal in operand field; LITTAB[littab\_ptr]:=this\_literal; littab\_ptr:=littab\_ptr+1;

Else(i.e. operand is a symbol) this\_entry:=SYMTAB entry number of operand; Generate IC ‘(IS,code)(S,this\_entry)’;

1. (processing of END statement)
2. Perform step 2(b).
3. Generate IC ‘(AD,02)’.
4. Go to Pass II.

### CONCLUSION:

**FAQs:**

1. What are different jobs performed by pass1 and pass 2 of assembler?
2. What is LC? When value of LC is updated?
3. Explain general format of
   1. intermediate code generation and
   2. machine language instruction.
4. How DC statement are processed?
5. How information is entered in LTTAB and POOLTAB?
6. What are pros and cons of single pass assembler?
7. What are different jobs performed by pass 2 of assembler?
8. How DS statement are processed?
9. How information is entered in SYMBTAB?
10. Enlist the data structures used in design of assembler?

### Assignment No 02

**TITLE:** To design a Pass-2 of two pass assembler.

**PROBLEM STATEMENT:** Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

**OBJECTIVE:** Implementation of pass-II of a two-pass assembler. The output of assignment-1 should be input for this assignment.

### THEORY:

An assembler is a computer program for translating assembly language essentially, a mnemonic representation of machine language into object code.

To make writing and reading programs written in machine language more convenient, mnemonic (symbol) were used for each machine instruction. These mnemonics were then later translated into machine language. Such a mnemonic language is called **assembly language**. Assemblers are program to automatically translate assembly language into machine language.

Source program

(in Assembly language)

tial scans over the input; asso

I/P

Object program

O/P

(Machine language) and

other information for the loader.

or sequen

Assembler

Data Bases

These tasks can be grouped into two passes task are one or more assembler modules.

Pass II: Purpose – general object program.

1. Look up value of symbols.
2. Generate instructions.
3. Generate data ( DS, DC and literals)
4. Process pseudo ops.

### Data structures:

Pass II data bases:

1. Copy of source program input to Pass I.
2. Location Counter (LC).

ciated with each

1. MOT, that indicates for each instruction (a) symbolic mnemonic;(b) length;(c) binary machine opcode, and (d) format .
2. POT, that indicates for each pseudo-op the symbolic mnemonic and the action to be taken in Pass II.
3. The ST prepared by Pass I, containing each label and its corresponding value.
4. A Base table (BT), that indicates which registers are currently specified as specified as base registers by USING pseudo-ops and what are the specified contents of these registers.
5. Work- space that is used to hold each instruction as its various parts are being assembled together.
6. Output of Pass II.

### Instruction Format:



**Algorithm of Pass II:**

1. code\_area\_address:=address of code\_area pooltab\_ptr :=1;

loc\_cntr := 0;

1. While next statement is not an END statement
   1. clear machine\_code\_buffer;
   2. If an LTORG statement then
      1. Process literals LITTAB [POOLTAB [pooltab\_ptr]]...LITTAB[lit\_tab\_ptr-1] to allocate memory and put address in the address Field. Update loc\_cntr accordingly.
      2. Pooltab\_ptr:=pooltab\_ptr+1;
      3. POOLTAB[pooltab\_ptr]:=littab\_ptr;
   3. If a STRAT or ORIGIN statement then
      1. Loc\_cntr := value specifid in operand field;
      2. Size:=0;
   4. If a declaration statement then
      1. If a DC statement then assemble the constant in machine\_code\_buffer;
      2. Size:=size of memory area required by DC/DS;
   5. If an imperative statement then
      1. Get operand address from SYMTAB or LITTAB
      2. Assemble instruction in machine\_code\_buffer.
      3. Size:=size of instruction;
   6. If size ≠ 0 then

Move contents of machine\_code\_buffer to the address code\_area\_address+loc\_cntr; Loc\_cntr:=loc\_cntr+size

1. (Processing of END statemnen)
2. Perform step 2(b) & 2(f).
3. Write Code\_area into output file.

### CONCLUSION:

**FAQs:**

1. What are different jobs performed by pass 2 of two assembler?
2. Explain general format of machine language instruction.
3. Enlist the data structures used in design of pass II assembler?
4. What is the difference between one pass and two pass assembler?
5. Why the task of assembler is dived into two passes?
6. How the forward reference is handled by one pass assembler?

### Assignment No 03

**TITLE :** To design Pass I of Two Pass macro Processor

**PROBLEM STATEMENT :** Design suitable data structures and implement pass-I of a two-pass macro- processor using OOP features in Java

**OBJECTIVE :** Design suitable data structures and develop a subset of a macro processor.

**THEORY :** A **macro** is a unit of specification for program generation through expansion. It consists of a name , a set of formal parameters and a body of code. The use of macro name with a set of actual parameters is replaced by some code generated from its body. This is called **macro expansion**.

A **macro definition** is enclosed between a macro header statement and a macro –end statement. These are typically located at the start of the program. A macro definition consists of

1. A macro prototype statement: it declares the name of macro and the names & kinds of its parameter.
2. One or more model statements: it is a statement from which an assembly language statement may be generated during macro expansion
3. Macro preprocessor statements: it is used to perform auxiliary functions during macro expansion.

### The macro prototype statement has the following format.

<macro name>[<formal parameter spec.>]

### <formal parameter spec.> has the syntax

&<parameter name>[<parameter kind>]

### Types of Parameters:

**Positional parameters:-** a positional parameter is written as &<parameter name> e.g. &SAMPLE.

**Keyword parameter:-** < parameter name> is an ordinary string and <parameter kind> is the string ‘=‘

### Example:

MACRO

INCR &MEM\_VAL, &INCR\_VAL, &REG MOVER &REG, &MEM\_VAL

ADD &REG , &INCR\_VAL MOVEM &REG, &MEM\_VAL MEND

* **Macro head & macro end** - MACRO & MEND
* **Prototype statement** - indicates that 3 parameter MEM\_VAL, INCR\_VAL, REG exists for the macro
* **Model statement-** statements with the operation code MOVER, ADD & MOVEM

### Macro preprocessor:

The macro preprocessor accepts an assembly program containing definition and calls and translates it into an assembly program which does not contain any macro definitions or calls.

Obj.code



Source code

with macro

Macro

processor

Expanded

code

Assembler

### Data structures used:-

**Macro name table (MNT)** is designed to hold the names of all macros defined in a program. A macro name is entered in this table when a macro definition is processed along with its MDT counter.

**Macro definition table(MDT)** The body of macro is stored in a table called the macro definition table (MDT) for use during macro expansion.

**Parameter name table(PNT)** A list of actual parameters is designed to hold the formal parameters during the expansion of a macro call and macro definition.

### Algorithm

1. mnt\_counter =1 mdt\_counter=1 mec = 1
2. while next statement is not End
   1. if word is MACRO

write macroname in MNT along with mdt\_counter mnt\_counter++

mdt\_counter++

make argument list array of parameters while next word is not endm

if argument list array to find the positional value and store that in mdt preceding with #

else

write word in MDT

ii if word present in mnt (i.e macro call) mdt\_counter=1

while mec>0

if next word is ‘ENDM’ if stack is not empty

pop the top of stack & set new value of mdt\_counter me--

else

if word present in mnt (i.e. nested call)

push argument list on stack & (mdt\_counter+1)

make argument list array of new macro which is called mdt\_counter = mdt\_counter in mnt stored in front of

macroname.

else

mec++

if word is present in argument list array(is arg.), store

positional value

else

1. Store the word in output file.

### CONCLUSION:

**FAQ:**

1. Describe macro processor.
2. Compare MACRO with Subroutine?
3. Explain what is MNT and its content?
4. Explain what is MDT and its content?
5. Explain data structures used in design of macro processor?
6. What are positional and keyword parameters?
7. What is lexical substitution, explain its types?
8. What are pros and cons of single pass macro processor?
9. What is macro?

### Assignment No 04

**TITLE :** To design Pass II of Two Pass macro Processor.

**PROBLEM STATEMENT :** Write a Java program for pass-II of a two-pass macro-processor. The output of assignment-3 (MNT, MDT and file without any macro definitions) should be input for this assignment.

**OBJECTIVE :** Design suitable data structures and develop a subset of a macro processor.

### THEORY :

**Macro preprocessor:**

The macro preprocessor accepts an assembly program containing definition and calls and translates it into an assembly program which does not contain any macro definitions or calls.

Obj.code



Source code

with macro

Macro

processor

Expanded

code

Assembler

A macro call leads to **macro expansion.** During macro expansion ,the macro call statement is replaced by a sequence of assembly statements.

### Conditional Macro Expansion

Conditional Macro Expansion is a powerful tool to control which instructions are to be generated by the macro-processor to be the input to the assembler. Conditional assembly takes place during the code generation phase and only then it is assembled.

Conditional directives AIF and AGO are used for such purposes. Syntax and semantics of the 2 are :-

LABEL: AIF (condition) action Where

* 1. Label may be a “permanent” label which will also occur in the code passed to the assembler or it may be an expansion time label. In the latter case it is distinguished by putting a mark say “%” before the actual label.
  2. The condition is a Boolean expression and can involve logical comparison among variables, constants, macro arguments etc. However, the truth value has to be evaluated during macro expansion time constants could be used. Again a distinction has to be made between normal variables (or constants) and expansion time variables or constants. Any expansion time variables can be preceeded by a special symbol say “&”.
  3. The action may be branching to some label or execution of a single instruction if the condition is evaluated to be “true”, which can be interpreted by the conditional macro processor.

LABEL : AGO label of a branch target

Where, conditional macro processor unconditionally branches to the target label.

The conditional macro processor should have the capability of evaluating logic and arithmetic expressions involving expansion time variables and constants as well as assigning values to expansion time variables. Depending on such capabilities, these processors require a set of commands or verbs which are to be interpreted and evaluated.

### Algorithm

1. mnt\_counter =1 mdt\_counter=1 mec = 1
2. while next statement is not End
   1. if word is MACRO

write macroname in MNT along with mdt\_counter mnt\_counter++

mdt\_counter++

make argument list array of parameters while next word is not endm

if argument list array to find the positional value and store that in mdt preceding with #

else

write word in MDT

ii if word present in mnt (i.e macro call) mdt\_counter=1

while mec>0

if next word is ‘ENDM’ if stack is not empty

pop the top of stack & set new value of mdt\_counter me--

else

if word present in mnt (i.e. nested call)

push argument list on stack & (mdt\_counter+1)

make argument list array of new macro which is called

else

mdt\_counter = mdt\_counter in mnt stored in front of macroname.

mec++

if word is present in argument list array(is arg.), store

positional value

else

1. Store the word in output file.

### CONCLUSION:

**FAQ:**

* 1. Explain what is macro expansion?
  2. Explain the work of pass II in two pass macro processor?
  3. Explain the work of pass I in two pass macro processor?
  4. What are nested macro calls?
  5. Enlist the tasks involved in macro expansion.

### Assignment No 05

**TITLE :** Write a Dynamic Link Library for simple calculator and Test it.

**PROBLEM STATEMENT :** Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).

**OBJECTIVE :** To understand the use of DLL and utilize the dll by defining it.

### THEORY :

A dynamic**-**link library (DLL) is a module that contain functions and data that can be used by another module (application or DLL).A DLL can define two kinds of functions:

### Exported

* + 1. **Internal**

The **exported** functions are intended to be called by other modules, as well as from within the DLL where they are defined.

**Internal** functions are typically intended to be called only from within the DLL where they are defined.

Although a DLL can export data, its data is generally used only by its functions. However, there is nothing to prevent another module from reading or writing that address. DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily. DLLs also help reduce memory overhead when several applications use the same functionality at the same time, because although each application receives its own copy of the DLL data, the applications share the DLL code.

The Windows application programming interface (API) is implemented as a set of dynamic-link libraries, so any process that uses the Windows API uses dynamic linking.

### About Dynamic-Link Libraries:

Dynamic linking allows a module to include only the information needed to locate an exported DLL function at load time or run time. Dynamic linking differs from the more familiar static linking, in which the linker copies a library function's code into each module that calls it.

### Types of Dynamic Linking:

There are two methods for calling a function in a DLL:

* In *load-time dynamic linking*, a module makes explicit calls to exported DLL functions as if they were local functions. This requires you to link the module with the import library for the DLL that contains the functions. An import library supplies the system with the information needed to load the DLL and locate the exported DLL functions when the application is loaded.
* In *run-time dynamic linking*, a module uses the LoadLibrary or LoadLibraryEx function to load the DLL at run time. After the DLL is loaded, the module calls the GetProcAddress function to get the addresses of the exported DLL functions. The module calls the exported DLL functions using the function pointers returned by **GetProcAddress**. This eliminates the need for an import library.

### DLLs and Memory Management:

Every process that loads the DLL maps it into its virtual address space. After the process loads the DLL into its virtual address, it can call the exported DLL functions. The system maintains a per-thread reference count for each DLL. When a thread loads the DLL, the reference count is incremented by one. When the process terminates, or when the reference count becomes zero (run-time dynamic linking only), the DLL is unloaded from the virtual address space of the process. Like any other function, an exported DLL function runs in the context of the thread that calls it. Therefore, the following conditions apply:

* The threads of the process that called the DLL can use handles opened by a DLL function. Similarly, handles opened by any thread of the calling process can be used in the DLL function.
* The DLL uses the stack of the calling thread and the virtual address space of the calling process.
* The DLL allocates memory from the virtual address space of the calling process.

### Advantages of Dynamic Linking:

Dynamic linking has the following advantages over static linking:

* + Multiple processes that load the same DLL at the same base address share a single copy of the DLL in physical memory. Doing this saves system memory and reduces swapping.
  + When the functions in a DLL change, the applications that use them do not need to be recompiled or relinked as long as the function arguments, calling conventions, and return values do not change. In contrast, statically linked object code requires that the application be relinked when the functions change.
  + A DLL can provide after-market support. For example, a display driver DLL can be modified to support a display that was not available when the application was initially shipped.
  + Programs written in different programming languages can call the same DLL function as long as the programs follow the same calling convention that the function uses. The calling convention (such as C, Pascal, or standard call) controls the order in which the calling function must push the arguments onto the stack, whether the function or the calling function is responsible for cleaning up the stack, and whether any arguments are passed in registers. For more information, see the documentation included with your compiler.

A potential disadvantage to using DLLs is that the application is not self-contained; it depends on the existence of a separate DLL module. The system terminates processes using load-time dynamic linking if they require a DLL that is not found at process startup and gives an error message to the user. The system does not terminate a process using run-time dynamic linking in this situation, but functions exported by the missing DLL are not available to the program.

### Dynamic-Link Library Creation:

To create a Dynamic-Link Library (DLL), you must create one or more source code files, and possibly a linker file for exporting the functions. If you plan to allow applications that use your DLL to use load-time dynamic linking, you must also create an import library.

### Creating Source Files:

The source files for a DLL contain exported functions and data, internal functions and data, and an optional entry-point function for the DLL. You may use any development tools that support the creation of Windows-based DLLs.

If your DLL may be used by a multithreaded application, you should make your DLL "thread-safe". You must synchronize access to all of the DLL's global data to avoid data corruption. You must also ensure that you link only with libraries that are thread-safe as well. For example, Microsoft Visual C++ contains multiple versions of the C Run-time Library, one that is not thread-safe and two that are.

### Exporting Functions

How you specify which functions in a DLL should be exported depends on the tools that you are using for development. Some compilers allow you to export a function directly in the source code by using a modifier in the function declaration. Other times, you must specify exports in a file that you pass to the linker.

For example, using Visual C++, there are two possible ways to export DLL functions: with \_declspec modifier or with a .def file. If you use the \_declspec modifier, it is not necessary to

use a .def file.

### Creating an Import Library

An import library (.lib) file contains information the linker needs to resolve external references to exported DLL functions, so the system can locate the specified DLL and exported DLL functions at run time.

### CONCLUSION:

**FAQs:**

1. Explain advantages of using DLL?
2. Explain disadvantages of using DLL?
3. What is DLL? Why it is called as dynamic linking? 4)Explain types of Dynamic linking.
4. How DLL is created and called?
5. What you mean by dynamic linking with import?
6. What you mean by dynamic linking without import?
7. compare static linking with dynamic linking?
8. What do you mean by DDE?
9. What do you mean by call back functions in DLL?

``

# GROUP - B

## EXPERIMENT NO : 1

### 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

### Software Requirements:

JDK/Eclipse

### Hardware Requirement:

* + M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

### 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.
* The primary objective of scheduling is to optimize system performance in accordance with the criteria deemed most important by the system designers.

### What is 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.

``

### What is scheduler?

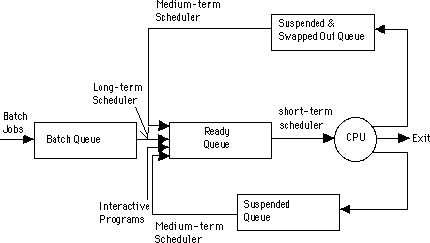
1. Scheduler in an OS module that selects the next job to be admitted into the system and the next process to run.
2. Primary objective of the scheduler is to optimize system performance in accordance with the criteria deemed by the system designers. In short, scheduler is that module of OS which schedules the programs in an efficient manner.

### 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 operating system.

* Long term scheduler
* Medium term scheduler
* Short term scheduler.

``

### Long Term Scheduler

* The long term scheduler, when present works with the batch queue and selects the next batch job to be executed.
* Batch is usually reserved for resource intensive (processor time, memory, special I/O devices) low priority programs that may be used fillers of low activity of interactive jobs.
* Batch jobs usually also contains programmer-assigned or system-assigned estimates of their resource needs such as memory size, expected execution time and device requirements.
* Primary goal of long term scheduler is to provide a balanced mix of jobs.

### Medium Term Scheduler

* After executing for a while, a running process may because suspended by making an I/O request or by issuing a system call.
* When number of processes becomes suspended, the remaining supply of ready processes in systems where all suspended processes remains resident in memory may become reduced to a level that impairs functioning of schedulers.
* The medium term scheduler is in charge of handling the swapped out processes.
* It has little to do while a process is remained as suspended.

### Short Term Scheduler

* The short term scheduler allocates the processor among the pool of ready processes resident in the memory.
* Its main objective is to maximize system performance in accordance with the chosen set of criteria.
* Some of the events introduced thus for that cause rescheduling by virtue of their ability to change the global system state are:
* Clock ticks
* Interrupt and I/O completions
* Most operational OS calls
* Sending and receiving of signals
* Activation of interactive programs.
* Whenever one of these events occurs ,the OS involves the short term scheduler.

### Scheduling Criteria :

* **CPU Utilization:**

Keep the CPU as busy as possible. It range from 0 to 100%. In practice, it range from 40 to 90%.

### Throughput:

Throughput is the rate at which processes are completed per unit of time.

``

### Turnaround time:

This is the how long a process takes to execute a process. It is calculated as the time gap between the submission of a process and its completion.

### Waiting time:

Waiting time is the sum of the time periods spent in waiting in the ready queue.

### Response time:

Response time is the time it takes to start responding from submission time. It is calculated as the amount of time it takes from when a request was submitted until the first response is produced.

### Non-preemptive Scheduling :

In non-preemptive mode, once if a process enters into running state, it continues to execute until it terminates or blocks itself to wait for Input/Output or by requesting some operating system service.

### Preemptive Scheduling :

In preemptive mode, currently running process may be interrupted and moved to the ready State by the operating system.

When a new process arrives or when an interrupt occurs, preemptive policies may incur greater overhead than non-preemptive version but preemptive version may provide better service.

It is desirable to maximize CPU utilization and throughput, and to minimize turnaround time, waiting time and response time.

### Types of scheduling Algorithms

* In general, scheduling disciplines may be pre-emptive or non-pre-emptive .
* In batch, non-pre-emptive implies that once scheduled, a selected job turns to completion. There are different types of scheduling algorithms such as:
  + FCFS(First Come First Serve)
  + SJF(Short Job First)
  + Priority scheduling
  + Round Robin Scheduling algorithm

### First Come First Serve Algorithm

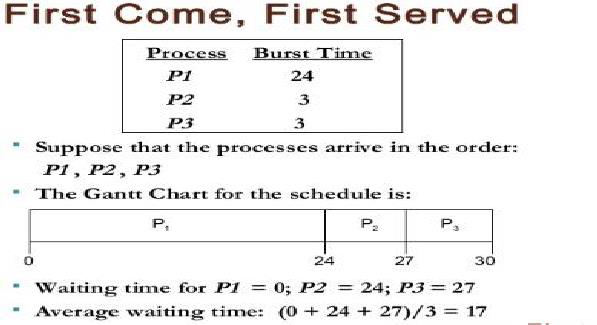
* FCFS is working on the simplest scheduling discipline.
* The workload is simply processed in an order of their arrival, with no pre-emption.
* FCFS scheduling may result into poor performance.
* Since there is no discrimination on the basis of required services, short jobs may considerable in turn around delay and waiting time.

``

### Advantages

* Better for long processes
* Simple method (i.e., minimum overhead on processor)
* No starvation

### Disadvantages

* Convoy effect occurs. Even very small process should wait for its turn to come to utilize the CPU. Short process behind long process results in lower CPU utilization.
* Throughput is not emphasized.

**Note : solve complete e.g. as we studied in practical(above is just sample e.g.). you can take any e.g.**

## Shortest Job First Algorithm :

* his is also known as **shortest job first**, or SJF
* This is a non-preemptive, pre-emptive scheduling algorithm.
* Best approach to minimize waiting time.
* Easy to implement in Batch systems where required CPU time is known in advance.
* Impossible to implement in interactive systems where required CPU time is not known.
* The processer should know in advance how much time process will take.

### Advantages

* It gives superior turnaround time performance to shortest process next because a short job is given immediate preference to a running longer job.
* Throughput is high.

### Disadvantages

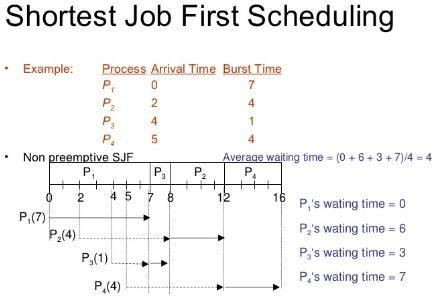
* Elapsed time (i.e., execution-completed-time) must be recorded, it results an additional overhead on the processor.
* Starvation may be possible for the longer processes.

### This algorithm is divided into two types:

* + Pre-emptive SJF
  + Non-pre-emptive SJF

### Pre-emptive SJF Algorithm:

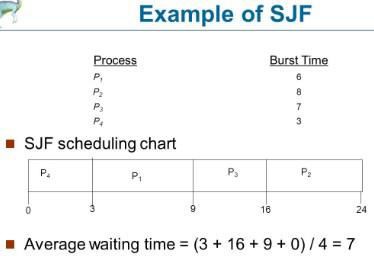
In this type of SJF, the shortest job is executed 1st. the job having least arrival time is taken first for execution. It is executed till the next job arrival is reached.



### Note : solve complete e.g. as we studied in practical(above is just sample e.g.). you can take any e.g.

**Non-pre-emptive SJF Algorithm:**

In this algorithm, job having less burst time is selected 1st for execution. It is executed for its total burst time and then the next job having least burst time is selected.



**Note : solve complete e.g. as we studied in practical(above is just sample e.g.). you can take any e.g.**

## Round Robin Scheduling :

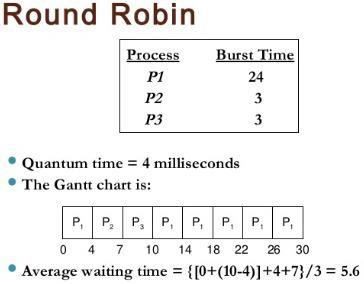
* Round Robin is the preemptive process scheduling algorithm.
* Each process is provided a fix time to execute, it is called a **quantum**.
* Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
* Context switching is used to save states of preempted processe

### Advantages

* Round-robin is effective in a general-purpose, times-sharing system or transaction-processing system.
* Fair treatment for all the processes.
* Overhead on processor is low.
* Overhead on processor is low.
* Good response time for short processes.

### Disadvantages

* Care must be taken in choosing quantum value.
* Processing overhead is there in handling clock interrupt.
* Throughput is low if time quantum is too small.



**Note : solve complete e.g. as we studied in practical(above is just sample e.g.). you can take any e.g.**

# Priority Scheduling :

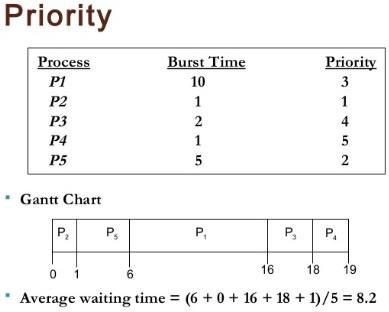
* Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
* Each process is assigned a priority. Process with highest priority is to be executed first and so on.
* Processes with same priority are executed on first come first served basis.
* Priority can be decided based on memory requirements, time requirements or any other resource requirement.

### Advantage

* Good response for the highest priority processes.

### Disadvantage

* Starvation may be possible for the lowest priority processes.



### Note : solve complete e.g. as we studied in practical(above is just sample e.g.). you can take any e.g.

1. **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

1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. 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

1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. 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

1. 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)
2. 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).

Step 7: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process.

### Priority Scheduling : Algorithms :

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.

#### System Programming & OS Laboratory

``

#### Third Year Computer Engineering

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

1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 7: Stop the process

### Note: you can write algorithm & procedure as per your program/concepts

1. **Flowchart :**

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

1. **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 execute first.

### References :

[**https://www.studytonight.com/operating-system/cpu-scheduling**](https://www.studytonight.com/operating-system/cpu-scheduling)[**https://www.go4expert.com/articles/types-of-scheduling-t22307/**](https://www.go4expert.com/articles/types-of-scheduling-t22307/)[**https://en.wikipedia.org/wiki/Scheduling\_(computing)**](https://en.wikipedia.org/wiki/Scheduling_(computing))[**https://www.tutorialspoint.com/operating\_system/os\_process\_scheduling\_algorithms.htm**](https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm)

### Oral Questions: [Write short answer ]

1. Scheduling? List types of scheduler & scheduling.
2. List and define scheduling criteria.
3. Define preemption & non-preemption.
4. State FCFS, SJF, Priority & Round Robin scheduling.
5. Compare FCFS, SJF, RR, Priority.

``