

**Pimpri Chinchwad College of Engineering & Research,Ravet**

*With Affiliation to*

**Savitribai Phule Pune University, Pune**

**System Programming & Operating System Lab [310257]**

**For Third Year**

**Bachelor of Computer Engineering (2015 Course)**

(With effect from 2017-18)

***Pimpri Chinchwad***

***College of Engineering & Research, Ravet***

~~--------------------------------------------~~

***Department of Computer Engineering***

COMPUTER DEPARTMENT

# LABORATORY MANNUAL

**ACADEMIC YEAR 2018-19**

**LAB MANUAL**

**SUBJECT: - *System Programming and Operating System Lab***

***FACULTY NAME: - Prof.Shweta A Koparde***

**Assignment No: - A1**

**Aim**: - Design of Pass-I 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

**Pre-requisites**:- Basic System Programming.

**Software Requirements**:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives: - 1.** To design and implement Pass I of two pass assembler.

1. To implement basic language translator
2. Convert Assembly Language Program(ASP) to Object Code.

**Outcomes: -** After completion of these program students will be able to accept input of ALP Convert to an intermediate file.

## Theory:-

Assemblers typically make two or more passes through a source program in order to

Resolve forward references in a program. A forward reference is defined as a type of instruction in the code segment that is referencing the label of an instruction, but the assembler has not yet encountered the definition of that instruction. Assembler reads the entire source program and constructs a symbol table of names and labels used in the program, that is, name of data fields and programs labels and their relative location (offset) within the segment.

Pass 1 determines the amount of code to be generated for each instruction.

**DATA STRUCTURES USED IN PASS1 ASSEMBLER**

Our simple assembler uses two major internal data structure; the Operation Code table (OPTAB) and the Symbol Table (SYMTAB). OPTAB is used to look up mnemonic operation codes and translate them to their machine language equivalents. SYMTAB is used to store values (addresses) assigned to labels.

We also need a Location counter LOCCTR. This is a variable that is used to help in the assignment of addresses. LOCCTR is initialized to the beginning address specified in the START statement. After each source statement is processed, the length of the assembled instructions or date area to be generated is added to LOCCTR. Thus whenever we reach a label in the source program, the current value of LOCCTR gives the address to be associated with that label.

The Operation code Table must contain the mnemonic operation code and its machine language equivalent. In more complex assembles, this table also contains information about instruction format and length. During Pass 1, OPTAB is used to look up and validate operations coded in the source program. In pass 2, it is used to translate the operation codes to machine language. Actually, in our simple SIC assembler, both of these processes could be done together in either PASS 1 or PASS 2. However, for a machine that has instructions of different lengths, we must search OPTAB in the first pass to find the instructions length for incrementing LOCCTR. Likewise , we must have the information from OPTAB is PASS 2 to tell us which instruction format to use in assembling the instruction ,and my peculiarities of the object code instruction. We have chosen to retain this structure in the current discussion because it is typical of most real assembles.

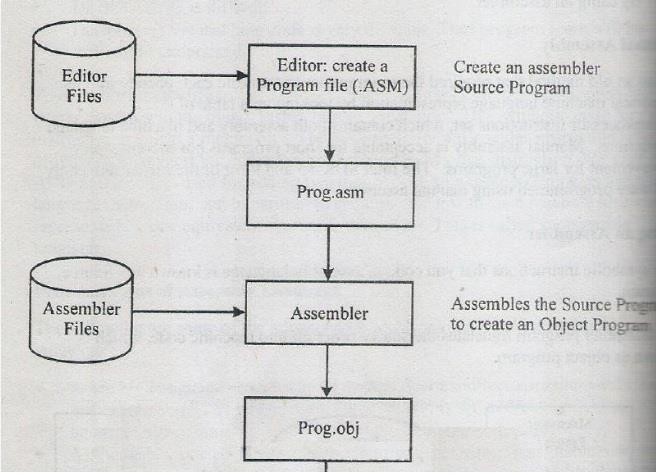
OPTAB is usually organized as a hash table, with mnemonic operation code as the key. The hash table organization is particularly appropriate, since it provides fast retrieval with a minimum of searching. In most cases, OPTAB is a static table – that is, entries are not normally added to or deleted from it. In such cases it is possible to design a special hashing function or other data structure to give optimum performance for the particular set of keys being stored. Most of the time, however, a general –purpose hashing method is used.

The Symbol table (SYMTAB) includes the name and value (address) for each label in the source program, together with flags to indicate error conditions (e.g., a symbol defined in two different places). This table may also contain other information about the data area or instruction labeled – for example, its type or length. During Pass 1 of the assembler, labels are entered intoSYMTAB as they are encountered in the source program, along with their assigned addresses (form LOCCTR). During Pass 2, symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.

SYMTAB is usually organized as a hash table for efficiency or insertion and retrieval. Since entries are rarely deleted from this table, efficiency of deletion is not an important consideration. Because SYMTAB is used heavily throughout the assemble, care should be taken in the selection of a hashing function. Programmers often select many labels that have similar characteristics – for example, labels that start or end with the same characters (like LOOP1, LOOP2) or are of the same length. It is important that the hashing function used perform well with such no-random keys. Division of the entire key by a prime table length often gives good result.

It is possible to both passes of the assembler to read the original source program as input. However, there is certain information (such as location counter values and error flags for statements) that can or should be communicated between the two passes. For this reason, Pass usually writes an INTERMEDIATE FILE that contains each source statement together with its assigned address, error indicators, etc, This file is used as the input to Pass 2.This working copy of the source program can also be used to retain the results of certain operations that may be performed during Pass 1, so these need not be performed again during Pass2.

Similarly, pointers into OPTAB and SYMTAB may retained for each operation code and symbol used. This avoids the need to repeat many of the table-searching operations. We assume for simplicity that the source lines are written in a fixed format with fields LABEL, OPCODE, and OPERAND. If one of these fields contains a character string that represents a number, we denote its numeric value with the prefix # ( for example , # [OPERAND]).



## Algorithm:-

1. Loc ctr = ‘0’ (default value) Littab ptr = ‘1’

Pool tab ptr = ‘1’

1. While the next statement is not END statement.
   1. If label is present then

This label = get the name of label

Store [this label, loc counter] in SYMTAB

* 1. If an LTORG statement then
     1. Process literals in LITTAB and allocate memory.
     2. Pooltab ptr = Pool tab ptr + 1;
     3. Littab ptr = Littab ptr +1;
  2. If instruction is START or ORIGIN then Loc counter = value specified in operand field;
  3. If an EQU statement then
     1. this. address = value of [address spec];
     2. Store [this. label, this. address] in Symbol Table.
  4. If a declaration statement then (DC/DS)
     1. Code = code of declaration statement;
     2. Size = size of memory area required by DC/DS.
     3. Loc counter =
     4. Generate IC (DL,code).
  5. If imperative statement then
     1. Code = machine opcode from OPTAB
     2. Loc ctr = loc ctr + instruction length from OPTAB
     3. Generate Intermediate Code (IS,code);

1. (Processing of END statement)
   1. Perform step 2(b)
   2. Generate IC (AD,02)
   3. Go to Pass 2

## Conclusion:-

Thus pass one of two passes assembler is implemented and the symbol table is generated successfully.

## Assignment No: - A2

**Aim**: - Design of Pass-II of two passes assembler

**Problem Statement**: - To write programs to implement pass two of a two pass assembler.

**Pre-requisites**:- Basic System Programming.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives: - 1.** To design and implement Pass II of two pass assembler.

1. To implement basic language translator
2. Convert intermediate file to Target file.

**Outcomes: -** After completion of this program students will be able to accept input of Intermediate file & Convert to Output/Target file.

## Theory:-

**What is a single pass assembler?**

It is an assembler that generally generates the object code directly in memory for immediate execution.

It passes through your source code only once and you are done. Now let us see how a two pass assembler works.

Simple, while on its way, if the assembler encounters an undefined label, it puts it into a symbol table along with the address where the undefined symbol’s value has to be placed when the symbol is found in future

## WHY DO WE NEED A TWO-PASS ASSEMBLER?

As explained, the one-pass assembler cannot resolve forward references of data symbols.

It requires all data symbols to be defined prior to being used. A two-pass assembler solves this dilemma by devoting one pass to exclusively resolve all (data/label) forward references and then generate object code with no hassles in the next pass.

If a data symbol depends on another and another depends on yet another, the assembler resolved this recursively.

***PASS II OF THE ASSEMBLER***

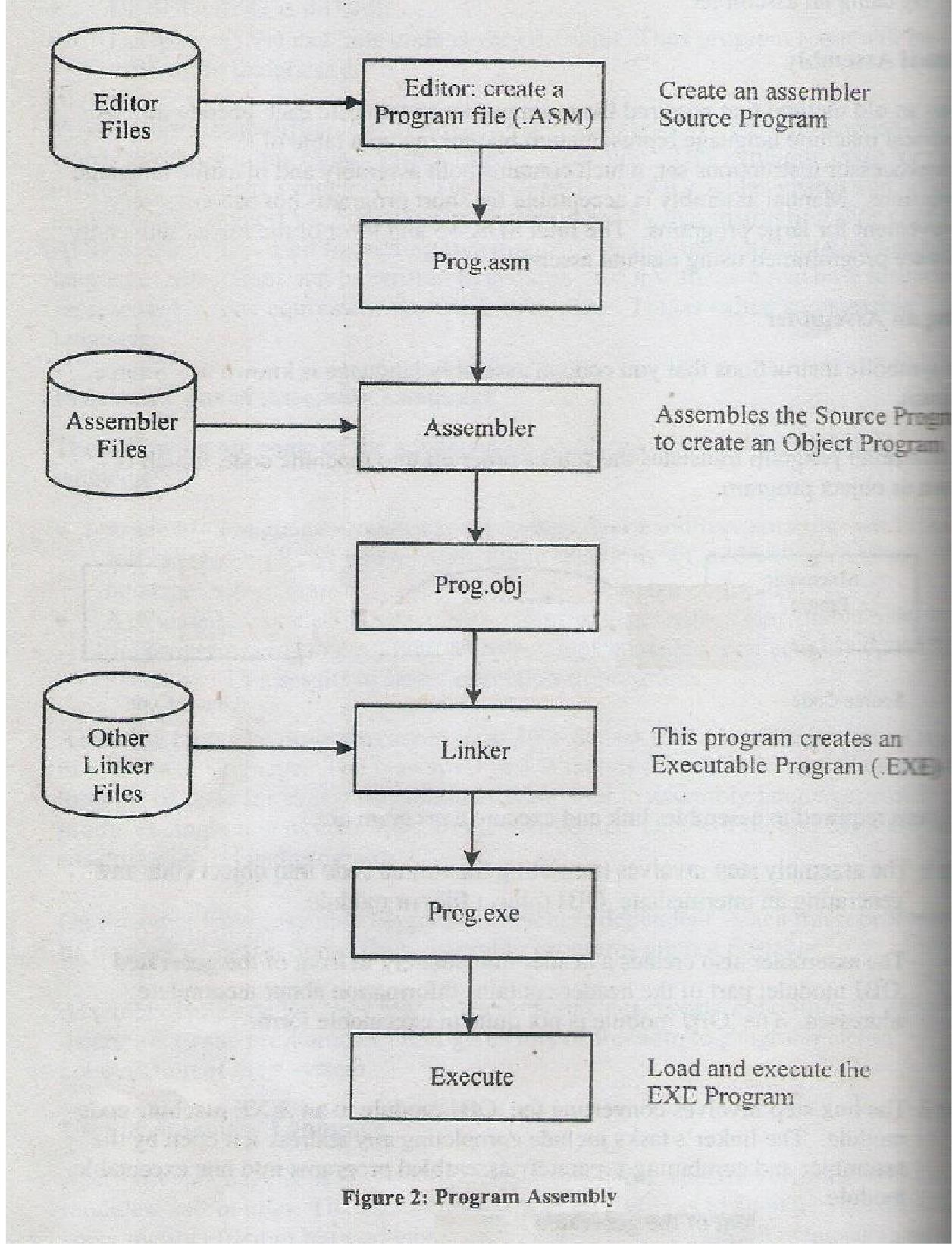
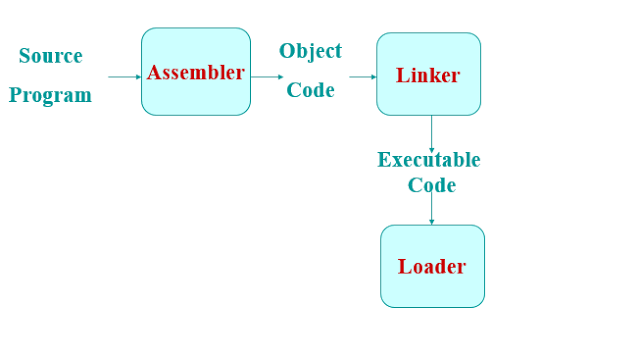


Fig. Steps for Pass II of two pass assembler.

***HOW IT WORKS:-***

1. *READ I/P OF PASS1 AS INTERMEDIATE FILE*
2. *SYNTHESIZE THE TARGET PROGRAM*

**Algorithm:-**

1. Code Area Address = address of code area (where target code is to be enabled) Pool tab ptr = ‘1’

Loc ctr = o (defined)

1. While the next statement is not an END statement.
   1. Clear memory buffer area.
   2. If an LTORG statement.
      1. Process the literals and assemble the literals in memory buffer.
      2. Size = size of memory area reqd. for literals.
      3. Pooltab ptr = Pool tab ptr + 1
   3. If a START or ORIGIN statement then

Loc ctr = value specified in operand field. size = 0

* 1. If a declaration statement
     1. If a DC statement

Assemble the constant in memory buffer

* + 1. If DS statement Generate machine code

Size = size of memory req. by DC/DS

* 1. If an imperative statement
     1. Assemble instruction in memory buffer.
     2. Size = size reqd. to store instruction
  2. If size != 0

Store the memory buffer code in code area address.

Loc ctr = loc ctr + size

1. END statement
   1. Perform steps 2(a) and 2(f)
   2. Write code area into o/p file.

## Conclusion:-

Thus pass II of two passes assembler is implemented and .exe target file is generated.

## Assignment No: -A3

**Aim**: - Design of a MACRO PASS-1

**Problem Statement**: - To write programs to implement pass I of a two pass macro processor.

**Pre-requisites**:- 1. Explain what is meant by pass of macro-processor.

* 1. Explain the need of PASS-I of two pass macro-processor.
  2. Define the term macro.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives: -** 1. To study of how Macro Processor produces expanded source program.

2. To understand Macro Language and Macro Processor in details as macro Definition, macro calls, etc.

**Outcomes: -** MACRO PASS-1 contains all macro definitions and its processing. Also to create MNT, MDT and argument list array.

## Theory:-

**1: Macro processor (Definition)**

A macro processor is a program that reads a file (or files) and scans them for certain keywords. When a keyword is found, it is replaced by some text. The keyword/text combination is called a Macro.

## Basic tasks performed by Macro processor:

1. Recognize macro definition
2. Save the definition
3. Recognize Call
4. Expanded calls and substitute arguments.

## Macro definition part

It consists of

1. Macro prototype Statement – This declares the name of macro and the types of parameters.
2. Model Statement – It is a statement for which assembly language statement is generated during macro expansion.
3. Preprocessor Statement – It is used to perform auxiliary function during macro expansion.

## Macro Call & Expansion

The operation define by macro can be used by writing a macro name in the mnemonic field And its operand field. Appearance of macro name in the mnemonic field leads to a macro call.

1. **Implementation Logic**
2. **Definition Processing: -** Scan all macro definitions and for each macro definition enter the Macro name in macro name table (MNT). Store entire macro definition in macro definition Table (MDT) and add auxiliary information in MNT such as no positional parameters (#PP) no Of key word parameters (#KP) , macro definition table position (MDTP) etc.
3. **Macro Expansion: -** Examine all statement in assembly source program to detect the macro Calls. For each macro call locate the macro in MNT, retrieve MDTP, establish the Correspondence between formal and actual parameters and expand the macro.
4. **Data Structure required for macro definition processing**
5. **Macro Name Table (MNT):-** Fields Name of macro, #PP(Number of positional parameters), #KP (Number of keyword parameters), MDTP (Macro Definition Table Pointer), KPDTP (Keywords Parameters Default Table Position).

Call replaces such statements by sequence of statements comprising the macro. This is known as Macro expansion.

## Algorithm/Pseudo Code

Before processing any definition initialize KPDTAB\_ptr, MDT\_ptr to 0 and MNT\_ptr to

-1. These table pointers are common to all macro definition. For each macro definition perform the

Following steps.

A one pass macro processor that alternate between macro definition and macro expansion in a Recursive way is able to handle recursive macro definition. Because of one pass structure, the **Definition of macro must appear in the source program before any statements that invoke that macro.**

## Algorithm:

A one-pass macro processor that alternate between macro definition and macro expansion algorithms.

## Algorithm:

begin {macro processor}

EXPANDING : = FALSE

while OPCODE ≠ ‘END’ do begin

GETLINE PROCESSLINE

end {while} end {macro processor}

procedure PROCESSLINE begin

search NAMTAB for OPCODE if found then

EXPAND

else if OPCODE = ‘MACRO’ then DEFINE

else write source line to expanded file end {PROCESSLINE}

## Algorithm:

procedure EXPAND begin

EXPANDING : = TRUE

get first line of macro definition {prototype} from DEFTAB set up arguments from macro invocation in ARGTAB

write macro invocation to expanded file as a comment while not end of macro definition do

begin

GETLINE PROCESSLINE

end {while}

EXPANDING : = FALSE end {EXPAND}

procedure GETLINE begin

if EXPANDING then

begin get next line of macro definition from DEFTAB substitute arguments from ARGTAB for positional notation

end {if}

else

read next line from input file end {GETLINE}

## Conclusion:

Thus pass I of Macro processor is implemented and .MNT, MDT & ALA file is generated.

## Assignment No: - A4

**Aim**: - Design of a MACRO PASS-2

**Problem Statement**: - To write programs to implement pass II of a two pass macro processor.

**Pre-requisites**:- 1. Explain what is meant by pass of macro-processor.

* 1. Explain the need of PASS-II of two pass macro-processor.
  2. Define the term macro.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives: -** 1. To study of how Macro Processor produces expanded source program.

2. To understand Macro Language and Macro Processor in details as macro Definition, macro calls, etc.

**Outcomes: -** MACRO PASS-2 contains all macro definitions and its processing. Also to create Final output into ALA.

## Theory:-

**1: Macro processor (Definition)**

A macro processor is a program that reads a file (or files) and scans them for certain keywords. When a keyword is found, it is replaced by some text. The keyword/text combination is called a Macro.

## 2. Basic tasks performed by Macro processor:

1. Recognize macro definition
2. Save the definition
3. Recognize Call
4. Expanded calls and substitute arguments.

In two-pass macro-preprocessor, you have two algorithms to implement, first pass and second pass. Both the algorithms examine line by line over the input data available. Two algorithms to implement two-pass macro-preprocessor are:

## Pass 1 Macro Definition

* **Pass 2 Macro Calls and Expansion**
  + **Pass 1 Macro Definition**

Pass 1 algorithm examines each line of the input data for macro pseudo opcode. Following are the steps that are performed during Pass 1 algorithm:

1. Initialize MDTC and MNTC with value one, so that previous value of MDTC and MNTC is Set to value one.
2. Read the first input data.
3. If this data contains MACRO pseudo opcode then A.Read the next data input.
   1. Enter the name of the macro and current value of MDTC in MNT.
   2. Increase the counter value of MNT by value one.
   3. Prepare that argument list array respective to the macro found.
   4. Enter the macro definition into MDT. Increase the counter of MDT By value one.
   5. Read next line of the input data.

G.Substitute the index notations for dummy arguments passed in Macro.

1. Increase the counter of the MDT by value one.
2. If mend pseudo opcode is encountered then next source of input data Is read.
3. Else expands data input.
4. If macro pseudo opcode is not encountered in data input then
5. A copy of input data is created.
6. If end pseudo opcode is found then go to Pass 2.
7. Otherwise read next source of input data.

## Pass 2 Macro Calls and Expansion

Pass two algorithm examines the operation code of every input line to check whether it exist in MNT or not.

Following are the steps that are performed during second pass algorithm:

1. Read the input data received from Pass 1.
2. Examine each operation code for finding respective entry in the MNT.
3. If name of the macro is encountered then
   1. A Pointer is set to the MNT entry where name of the macro is found. This pointer is called Macro Definition Table Pointer (MDTP).
   2. Prepare argument list array containing a table of dummy arguments.
   3. Increase the value of MDTP by value one.
   4. Read next line from MDT.
   5. Substitute the values from the arguments list of the macro for Dummy arguments.
   6. If mend pseudo opcode is found then next source of input data is Read.
   7. Else expands data input.
4. When macro name is not found then create expanded data file.
5. If end pseudo opcode is encountered then feed the expanded source file to Assembler for processing.
6. Else read next source of data input.

## Conclusion:

Thus pass II of Macro processor is implemented and ALA file is generated.

## Assignment No: - B1

**Aim**: - Design DLL using VB 6.0

**Problem Statement**: - To write programs to create DLL to perform math’s operation using VB.

**Pre-requisites**:- 1. Explain what is meant by DLL

1. Explain the need of VB
2. Define the term forms & DLL.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Windows 7 |
| 3 | S/W name | VB 6.0 |

**Hardware Requirements**: - No

**Objectives: -** 1. To study of how DLL working with Active EXE file.

2. To understand forms design, generating .dll files, browsing .dll file, executing Projects using VB 6.0.

**Outcomes: -** Mathematical operations like addition, subtraction, multiplication & division Carried out using .dll files and VB 6.0 forms.

## Theory:-

**Dynamic Link Library**

A dynamic link library (DLL) is a collection of small programs that can be loaded when needed by larger programs and used at the same time.

## Advantage of DLL

The advantage of DLL files is space is saved in random access memory (RAM) because the files

**Don’t get loaded** into RAM together with the main program.

# What is Visual Basic?

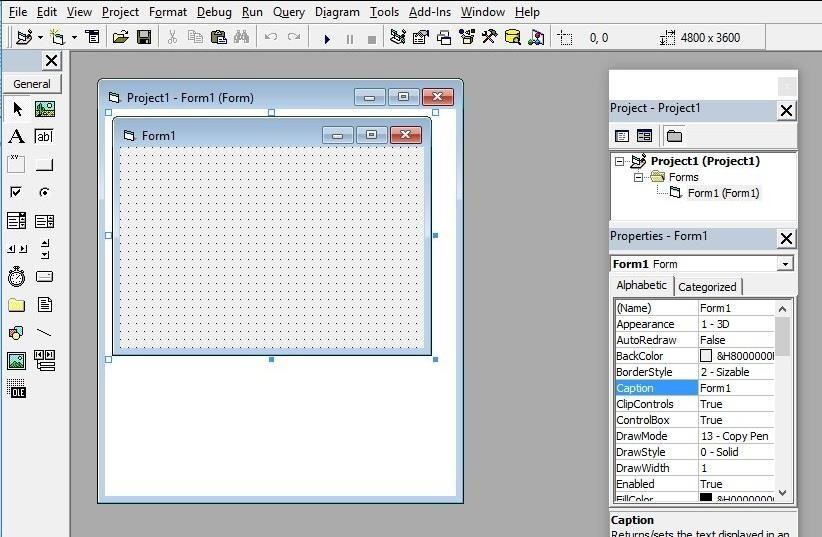
Visual Basic is a third-generation event-driven programming language first released by Microsoft in 1991. It evolved from the earlier DOS version called BASIC. **BASIC** means **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. Since then Microsoft has released many versions of Visual Basic, from Visual Basic 1.0 to the final version Visual Basic 6.0. Visual Basic is a user-friendly programming language designed for beginners, and it enables anyone to develop GUI window applications easily.

In 2002, Microsoft released Visual Basic.NET (VB.NET) to replace Visual Basic 6. Thereafter, Microsoft declared VB6 a legacy programming language in 2008. Fortunately, Microsoft still provides some form of support for VB6. VB.NET is a fully object-oriented programming language implemented in the .NET Framework. It was created to cater for the development of the web as well as mobile applications. However, many developers still favor Visual Basic 6.0 over its successor Visual Basic.NET.



You can choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications that we could create; however, we shall concentrate on creating Standard EXE programs (EXE means executable). Before you begin, you must think of an application that preferably has commercial, educational or recreational value. Next, click on the Standard EXE icon to go into the actual Visual Basic 6 programming environment.

When you start a new Visual Basic 6 Standard EXE project, you will be presented with the Visual Basic 6 Integrated Development Environment (IDE). The Visual Basic 6 Integrated Programming Environment is shown in Figure 1.2. It consists of the toolbox, the form, the project explorer and the properties window.



The Form is the primary building block of a Visual Basic 6 application. A Visual Basic 6 application can actually comprise many forms, but we shall focus on developing an application with one form first. We will learn how to develop applications with multiple forms later. Before you proceed to build the application, it is a good practice to save the project first. You can save the project by selecting **Save** Project from the File menu, assign a name to your project and save it in a certain folder. You shall now proceed to learn Visual Basic programming from the next lesson onwards.

## Conclusion:

Thus mathematical operations (Add, sub etc) carried out using concept DLL in VB 6.0

## Assignment No: - B2

**Aim**: - Design Lex program for to generate token of given input file

**Problem Statement**: - Write a program using Lex specifications to implement lexical analysis Phase of compiler to generate tokens of subset of Java program.

**Pre-requisites**:- LEX 110, LEX 120, LEX 130, LEX 140, LEX 160, 250

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | LEX Tool (flex) |

**Hardware Requirements**: - No

**Objectives: -** 1. To understand LEX Concepts

1. To implement LEX Program
2. To study about Lex & Java
3. To know important about Lexical analyzer

**Outcomes: -** After completion of this assignment students will be able to:

1. Understand the concept of LEX Tool
2. Understand the lexical analysis part
3. It can be used for data mining concepts.

## Theory:-

Lex stands for Lexical Analyzer. Lex is a tool for generating Scanners. Scanners are programs that recognize lexical patterns in text. These lexical patterns (or regular Expressions) are defined in a particular syntax. A matched regular expression may have an associated action. This action may also include returning a token. When Lex receives input in the form of a file or text, it takes input one character at a time and continues until a pattern is matched, then lex performs the associated action (Which may include returning a token). If, on the other hand, no regular expression can be matched, further processing stops and Lex displays an error message. Lex and C are tightly coupled. A .lex file (Files in lex have the extension .lex) is passed through the lex utility, and produces output files in C. These file(s) are coupled to produce an executable version of the lexical analyzer. Lex turns the users expressions and actions into the host general – purpose language; the generated program is named yylex. The yylex program will recognize

expressions in a stream (called input in this memo) and perform the specified actions for each expression as it is detected.

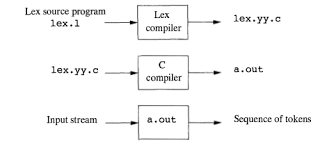
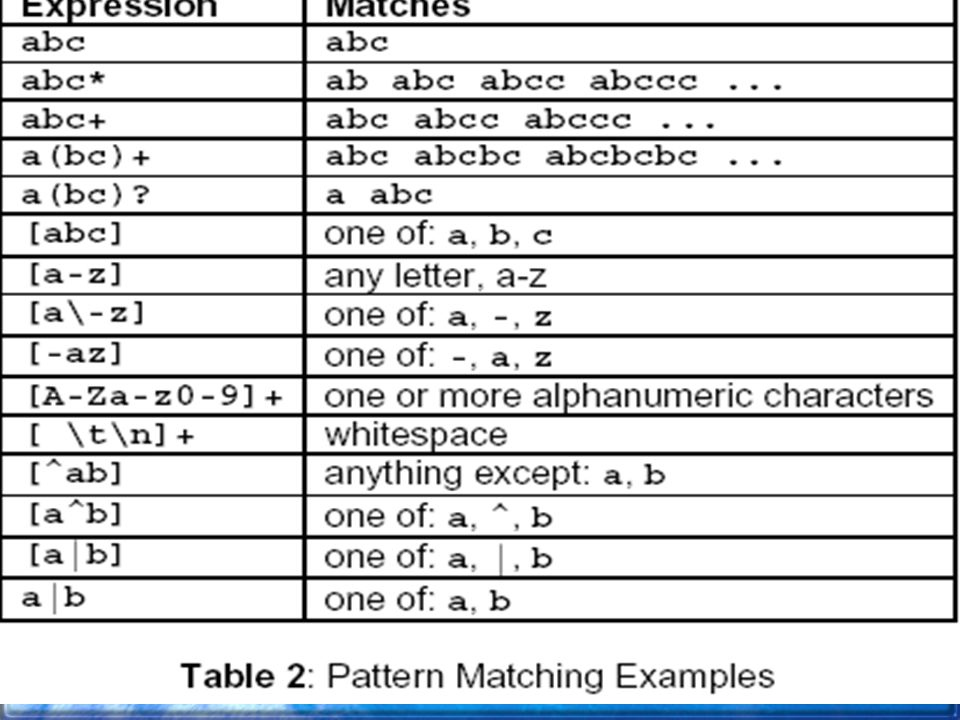


Fig:Overview of Lex Tool



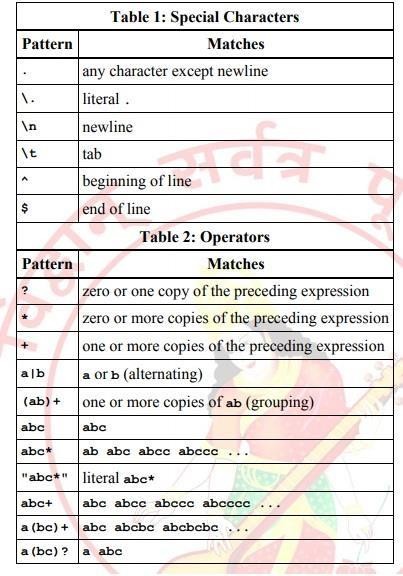
## Regular Expression in Lex: -

A Regular expression is a pattern description using a Meta language. An expression is made up of symbols. Normal symbols are characters and numbers, but there are other symbols that have special meaning in Lex. The following two tables define some of the symbols used in Lex and give a few typical examples.

## Programming in Lex: -

Programming in Lex can be divided into three steps: 1. Specify the

pattern-associated actions in a form that Lex can understand. 2. Run Lex over this file to generate C code for the scanner. 3. Compile and link the C code to produce the executable scanner. Note: If the scanner is part of a parser developed using Yacc, only steps 1 and 2 should be performed. A Lex program is divided into three sections: the first section has global C and Lex declaration, the second section has the patterns (coded in C), and the third section has supplement C functions. Main (), for example, would typically be founding the third section. These sections are delimited by %%.so, to get back to the word to the word-counting Lex program; let’s look at the composition of the various program sections.



Regular expressions are used for pattern matching. A character class defines a single character and normal operators lose their meaning. Two operators supported in a character class are the hyphen ("- ") and circumflex ("^"). When used between two characters the hyphen represents a range of characters. The circumflex, when used as the first character, negates the expression. If two patterns match the same string the longest match wins. In case both matches are the same length, then the first pattern listed is used.

Input to Lex is divided into three sections with %% dividing the sections. This is best illustrated by example. The first example is the shortest possible lex file: %% Input is copied to output one character at a time. The first %% is always required as there must always be a rules section. However if we don’t specify any rules then the default action is to match everything and copy it to output. Defaults for input and output are stdin and stdout, respectively.

## Conclusion:-

Thus, I have studied lexical analyzer and implemented an application for lexical analyzer to perform scan the program and generates token of subset of java.

## Assignment No: - B3

**Aim**: - Design Lex program to count no. of words, lines and characters of given input file.

**Problem Statement**: - Write a program using Lex specifications to implement lexical analysis Phase of compiler to count no. of words, lines and characters of given Input file.

**Pre-requisites**:- LEX 110, LEX 120, LEX 130, LEX 140, LEX 160, 250

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | LEX Tool (flex) |

**Hardware Requirements**: - No

**Objectives: -** 1. To understand LEX Concepts.

1. To implement LEX Program for no’s of count.
2. To study about Lex & Java.
3. To know important about Lexical analyzer.

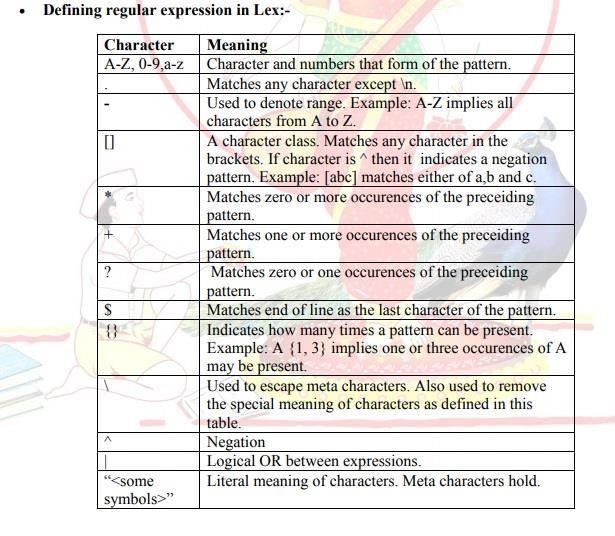
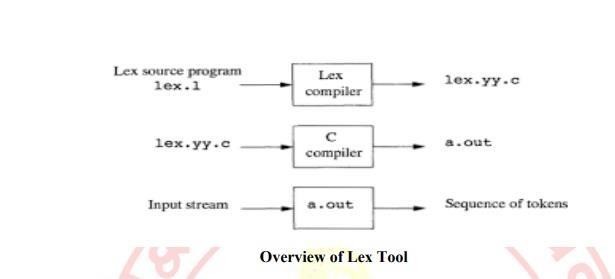
**Outcomes: -** After completion of this assignment students will be able to:

1. Understand the concept of LEX Tool
2. Understand the lexical analysis part
3. It can be used for data mining concepts.

## Theory:-

Lex stands for Lexical Analyzer. Lex is a tool for generating Scanners. Scanners are programs that recognize lexical patterns in text. These lexical patterns (or regular Expressions) are defined in a particular syntax. A matched regular expression may have an associated action. This action may also include returning a token. When Lex receives input in the form of a file or text, it takes input one character at a time and continues until a pattern is matched, then lex performs the associated action (Which may include returning a token). If, on the other hand, no regular expression can be matched, further processing stops and Lex displays an error message. Lex and C are tightly coupled. A .lex file (Files in lex have the extension .lex) is passed through the lex utility, and produces output files in C. These file(s) are coupled to produce an executable version of the lexical analyzer. Lex turns the users expressions and actions into the host general – purpose language; the generated program is named yylex. The yylex program will recognize

Expressions in a stream (called input in this memo) and perform the specified actions for each expression as it is detected.



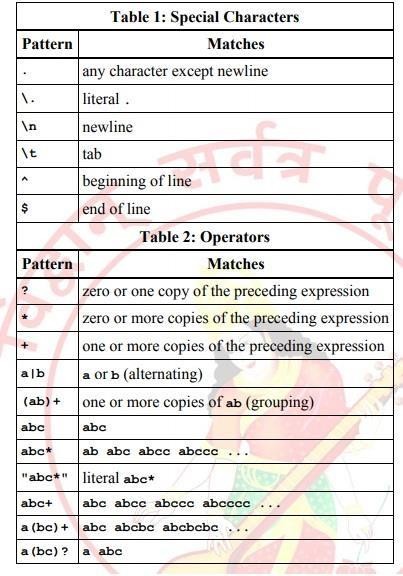
## Regular Expression in Lex: -

A Regular expression is a pattern description using a Meta language. An expression is made up of symbols. Normal symbols are characters and numbers, but there are other symbols that have special meaning in Lex. The following two tables define some of the symbols used in Lex and give a few typical examples.

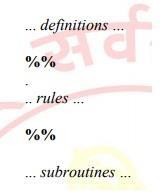
## Programming in Lex: -

Programming in Lex can be divided into three steps: 1. Specify the

Pattern-associated actions in a form that Lex can understand. 2. Run Lex over this file to generate C code for the scanner. 3. Compile and link the C code to produce the executable scanner. Note: If the scanner is part of a parser developed using Yacc, only steps 1 and 2 should be performed. A Lex program is divided into three sections: the first section has global C and Lex declaration, the second section has the patterns (coded in C), and the third section has supplement C functions. Main (), for example, would typically be founding the third section. These sections are delimited by %%.so, to get back to the word to the word-counting Lex program; let’s look at the composition of the various program sections.



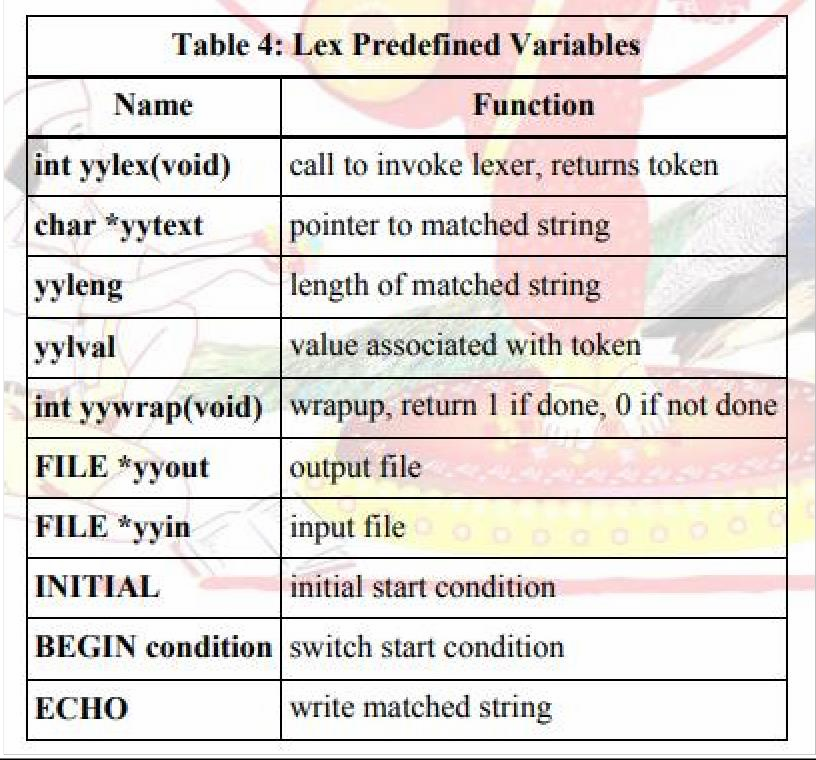
Regular expressions are used for pattern matching. A character class defines a single character and normal operators lose their meaning. Two operators supported in a character class are the hyphen ("- ") and circumflex ("^"). When used between two characters the hyphen represents a range of characters. The circumflex, when used as the first character, negates the expression. If two patterns match the same string the longest match wins. In case both matches are the same length, then the first pattern listed is used.

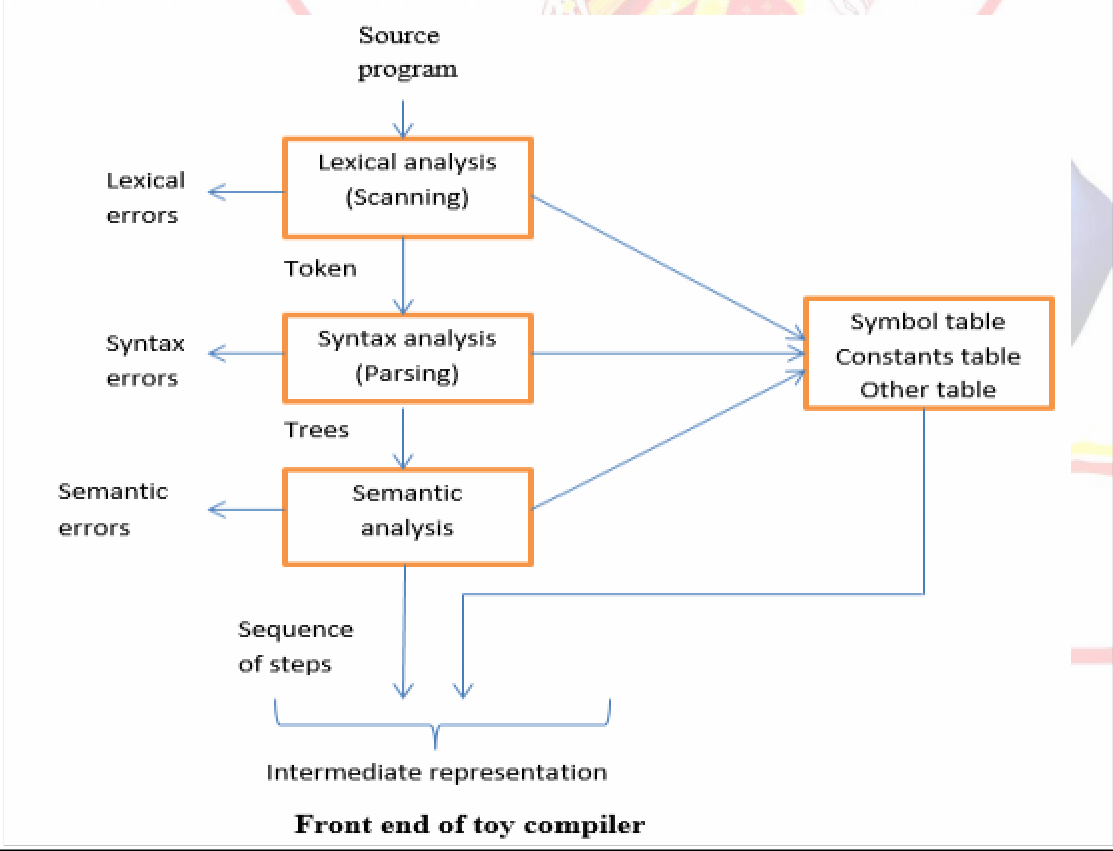


Input to Lex is divided into three sections with %% dividing the sections. This is best illustrated by example. The first example is the shortest possible lex file: %% Input is copied to output one character at a time. The first %% is always required as there must always be a rules section. However if we don’t specify any rules then the default action is to match everything and copy it to output. Defaults for input and output are stdin and stdout, respectively.

Two patterns have been specified in the rules section. Each pattern must begin in column one. This is followed by whitespace (space, tab or newline) and an optional action associated with the pattern. The action may be a single Statement, or multiple C statements, enclosed in braces. Anything not starting in column one is copied verbatim to the generated C file. We may take advantage of this behavior to specify comments in our lex file. In this example there are two patterns, "." and "\n", with an ECHO action associated for each pattern. Several macros and variables are predefined by lex. ECHO is a macro that writes code matched by the pattern. This is the default action for any unmatched strings. Typically, ECHO is defined as: #define ECHO fwrite (yytext, yyleng, 1, yyout) Variable yytext is a pointer to the matched string (NULL- terminated) and yyleng is the length of the matched string. Variable yyout is the output file and defaults to stdout. Function yywrap is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required. Every C program requires a main function. In this case we simply call yylex that is the main entry point for lex . Some implementations of lex include copies of main and yywrap in a library thus eliminating the need to code them explicitly. This is why our first example, the shortest lex program, functioned properly.

HOW THE INPUT IS MATCHED: - When the generated scanner is run, it analyzes its input looking for strings, which match any of its patterns. If it finds more than one match, it takes the one matching the most text. If it finds two or more matches of the same length, the rule listed first in the flex input file is chosen. Once the match is determined, the text corresponding to the match (called the token) is made available in the global character pointer „yytext‟,and its length in the global integer „yyleng‟.The action corresponding to the matched pattern is then executed, and then the remaining input is scanned for another match. If no match is found, then the default rule is executed: the next character in the input is considered matched and copied to the standard output.





**Conclusion:-**

Thus, I have studied lexical analyzer and implemented an application for lexical analyzer to count total number of words, chard and line etc.

## Assignment No: - B4

**Aim**: - Design Lex & Yacc program to validate type and syntax of variable declaration in Java.

**Problem Statement**: - Write a program using Yacc specifications to implement lexical analysis Phase of compiler to validate type and syntax of variable declaration in Java.

**Pre-requisites**:- LEX 110, LEX 120, LEX 130, LEX 140, LEX 160, 250

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | FLEX, YACC (LEX & YACC) |

**Hardware Requirements**: - No

**Objectives: -** 1. To understand LEX & YACC Concepts.

1. To implement LEX Program & YACC programs.
2. To study about Lex & Yaac specification.
3. To know important about Lexical analyzer and Syntax analysis.

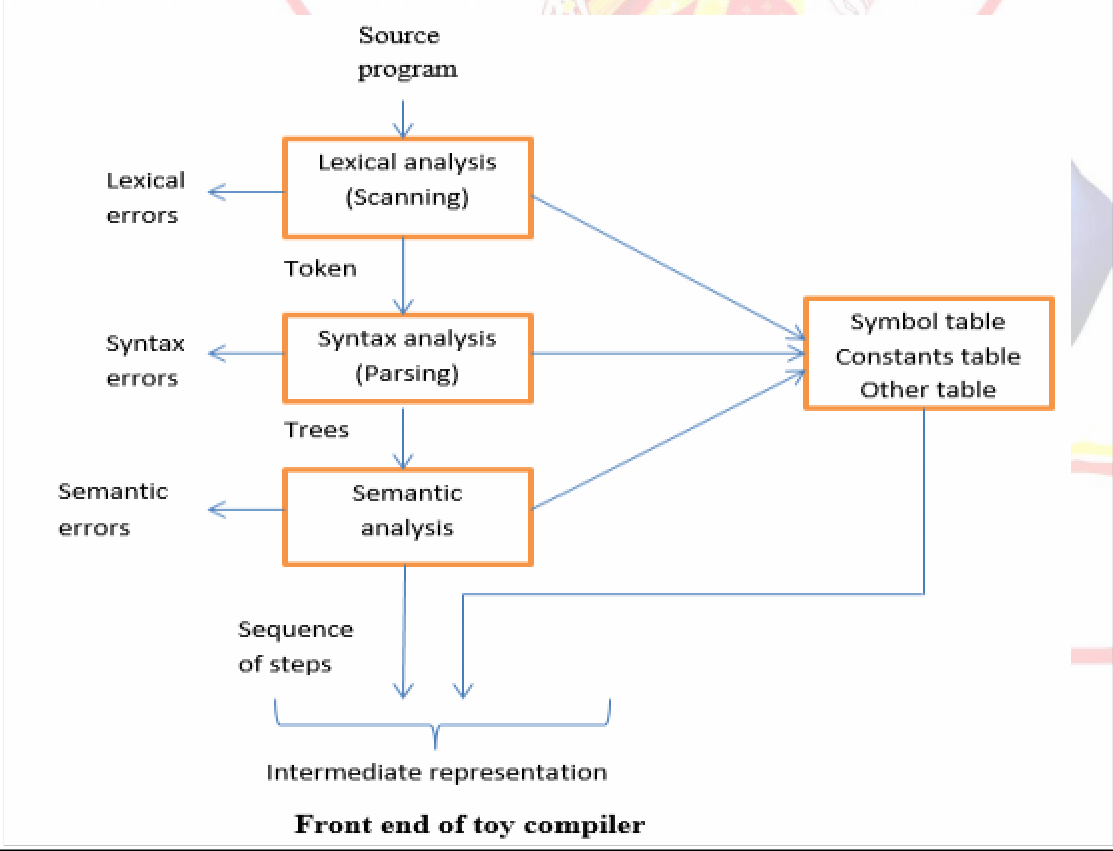
**Outcomes: -** After completion of this assignment students will be able to:

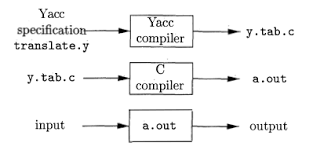
1. Understand the concept of LEX & YACC Tool.
2. Understand the lexical analysis & Syntax analysis part.
3. It can be used for data mining and checking (validation) concepts.

## Theory:-

Yacc (Yet another Compiler-Compiler) is a computer program for the UNIX operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus– Naur Form (BNF). Yacc is supplied as a standard utility on BSD and AT&T UNIX. GNU based Linux distributions include Bison, a forward-compatible Yacc replacement. Yacc is one of the automatic tools for generating the parser program. Basically Yacc is a LALR parser generator. The Yacc can report conflicts or ambiguities (if at all) in the form of error messages. LEX and Yacc work together to analyse the program syntactically. Yacc is officially known as a “parser”. Its job is to analyze the structure of the input stream, and operate of the “big picture”. In the course of its normal work, the parser also verifies that the input is syntactically sound.

YACC stands for “Yet another Compiler Compiler” which is a utility available from UNIX.





**Structure of a yacc file:** A yacc file looks much like a lex file:

...definitions...

%%

...rules...

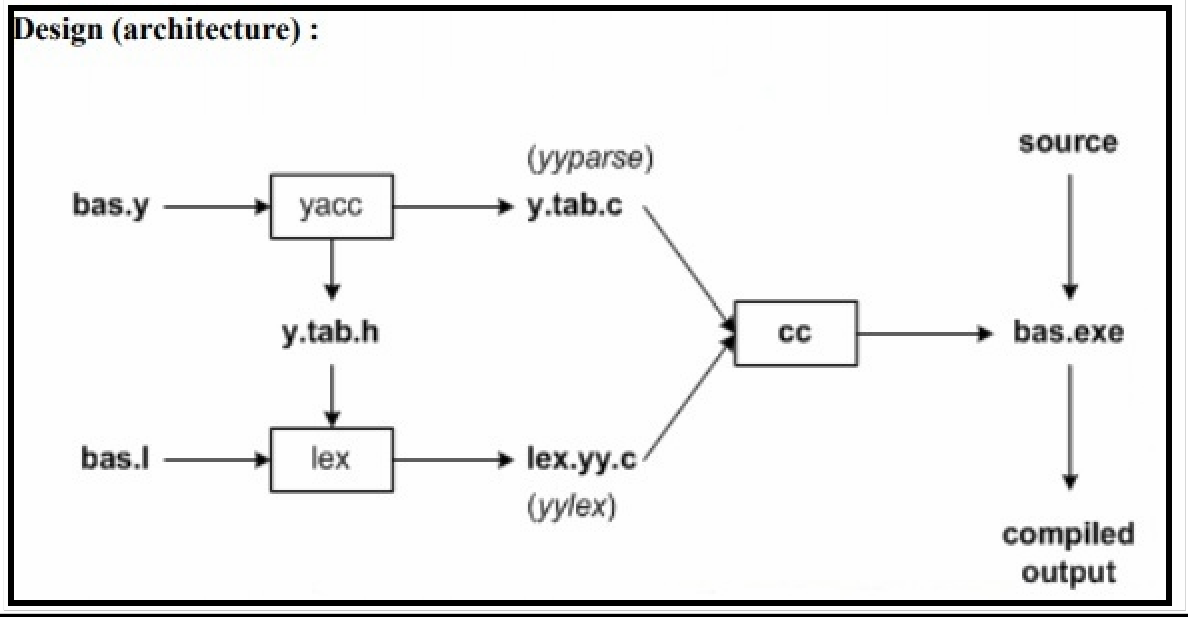
%%

...code...

Definitions As with lex, all code between %{ and %} is copied to the beginning of the resulting C file. Rules As with lex, a number of combinations of pattern and action. The patterns are now those of a context-free grammar, rather than of a regular grammar as was the 3 case with lex code. This can be very elaborate, but the main ingredient is the call to yyparse, the grammatical parse. Input to yacc is divided into three sections. The definitions section consists of token declarations and C code bracketed by “%{“ and “%}”. The BNF grammar is placed in the rules section and user subroutines are added in the subroutines section. This is best illustrated by constructing a small calculator that can add and subtract numbers. We’ll begin by examining the linkage between lex and yacc. Here is the definitions section for the yacc input file:

Grammars for yacc are described using a variant of Backus Naur Form (BNF). This technique, pioneered by John Backus and Peter Naur, was used to describe ALGOL60. A BNF grammar can be used to express context-free languages. Most constructs in modern programming languages can be represented in BNF. For example, the grammar for an expression that multiplies and adds numbers is:

1. E -> E + E
2. E-> E\* E
3. E -> id



## Conclusion:-

Thus, I have studied lexical analyzer, syntax analysis and implemented Lex & Yacc application for Syntax analyzer to validate the given infix expression.

## Assignment No: - B5

**Aim**: - Design Lex & Yacc program to recognize simple and compound sentences given in input File.

**Problem Statement**: - Write a program using Yacc specifications to implement lexical analysis Phase of compiler to recognize simple and compound sentences given in Input file.

**Pre-requisites**:- LEX 110, LEX 120, LEX 130, LEX 140, LEX 160, 250

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | FLEX, YACC (LEX & YACC) |

**Hardware Requirements**: - No

**Objectives: -** 1. To understand LEX & YACC Concepts.

1. To implement LEX Program & YACC programs.
2. To study about Lex & Yaac specification.
3. To know important about Lexical analyzer and Syntax analysis.

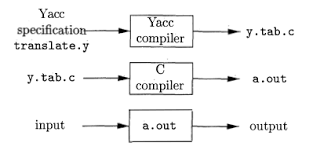
**Outcomes: -** After completion of this assignment students will be able to:

1. Understand the concept of LEX & YACC Tool.
2. Understand the lexical analysis & Syntax analysis part.
3. Understand the Simple and Compound sentence.

## Theory:-

Yacc (Yet another Compiler-Compiler) is a computer program for the UNIX operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus– Naur Form (BNF). Yacc is supplied as a standard utility on BSD and AT&T UNIX. GNU based Linux distributions include Bison, a forward-compatible Yacc replacement. Yacc is one of the automatic tools for generating the parser program. Basically Yacc is a LALR parser generator. The Yacc can report conflicts or ambiguities (if at all) in the form of error messages. LEX and Yacc work together to analyse the program syntactically. Yacc is officially known as a “parser”. Its job is to analyze the structure of the input stream, and operate of the “big picture”. In the course of its normal work, the parser also verifies that the input is syntactically sound.

YACC stands for “Yet another Compiler Compiler” which is a utility available from UNIX.



**Structure of a yacc file:** A yacc file looks much like a lex file:

...definitions...

%%

...rules...

%%

...code...

Definitions As with lex, all code between %{ and %} is copied to the beginning of the resulting C file. Rules As with lex, a number of combinations of pattern and action. The patterns are now those of a context-free grammar, rather than of a regular grammar as was the 3 case with lex code. This can be very elaborate, but the main ingredient is the call to yyparse, the grammatical parse. Input to yacc is divided into three sections. The definitions section consists of token declarations and C code bracketed by “%{“ and “%}”. The BNF grammar is placed in the rules section and user subroutines are added in the subroutines section. This is best illustrated by constructing a small calculator that can add and subtract numbers. We’ll begin by examining the linkage between lex and yacc. Here is the definitions section for the yacc input file:

Grammars for yacc are described using a variant of Backus Naur Form (BNF). This technique, pioneered by John Backus and Peter Naur, was used to describe ALGOL60. A BNF grammar can be used to express context-free languages. Most constructs in modern programming languages can be represented in BNF. For example, the grammar for an expression that multiplies and adds numbers is:

1. E -> E + E
2. E-> E\* E
3. E -> id

## Translating, Compiling and Executing A Yacc Program:

The Lex program file consists of Lex specification and should be named .l and the Yacc program consists of Yacc specifications and should be named .y. following command may be issued to generate the parser .

Lex <filename>.l Yacc –d<filename> .y cc lex.yy.c y.tab.c –ll

./a.out

Yacc reads the grammar description in .y and generates a parser, function yyparse, in file y.tab.c. The –d option causes yacc to generate the definitions for tokens that are declared in the .y and place them in file y.tab.h. Lex reads the pattern descriptions in .l, includes file y.tab.h, and generates a lexical analyzer, function yylex, in the file lex.yy.c

Finally, the lexer and the parser are compiled and linked (-ll) together to form the output file, a.out (by default).

The execution of the parser begins from the main function, which will be ultimately call yyparse () to run the parser. Function yyparse () automatically calls yylex () whenever it is in need of token.

## Lexical Analyzer for YACC:

The user must supply a lexical analyzer to read the input stream and communicate tokens (with values, if desired) to the parser. The lexical analyzer is an integer-valued function called yylex. The function returns an integer, the token number, representing the kind of token read. If there is a value associated with that token, it should be assigned to the external variable yylval.

The parser and the lexical analyzer must agree on these token numbers in order for communication between them to take place. The numbers may be chosen by Yacc, or chosen by the user. In either case, the ``# define'' mechanism of C is used to allow the lexical analyzer to return these numbers symbolically. For example, suppose that the token name DIGIT has been defined in the declarations section of the Yacc specification file. The relevant portion of the lexical analyzer might look like:

yylex(){

extern int yylval; int c;

. . .

c = getchar();

. . . switch( c ) {

. . . case '0': case '1':

. . .

case '9':

yylval = c-'0'; return( DIGIT );

. . . }

. . .

The intent is to return a token number of DIGIT, and a value equal to the numerical value of the digit. Provided that the lexical analyzer code is placed in the programs section of the specification file, the identifier DIGIT will be defined as the token number associated with the token DIGIT.

This mechanism leads to clear, easily modified lexical analyzers; the only pitfall is the need to avoid using any token names in the grammar that are reserved or significant in C or the parser; for example, the use of token names if or while will almost certainly cause severe difficulties when the lexical analyzer is compiled. The token name error is reserved for error handling, and should not be used naively.

As mentioned above, the token numbers may be chosen by Yacc or by the user. In the default situation, the numbers are chosen by Yacc. The default token number for a literal character is the numerical value of the character in the local character set. Other names are assigned token numbers starting at 257.

When Yacc generates, the parser (by default y.tab.c, which is C file), it will assign token numbers for all the tokens defined in Yacc program. Token numbers will be assigned using”#define”and will be copied, by default, to y.tab.h file. The lexical analyzer wills reads from this file or any further use.

## Comparing Sentence Types:-

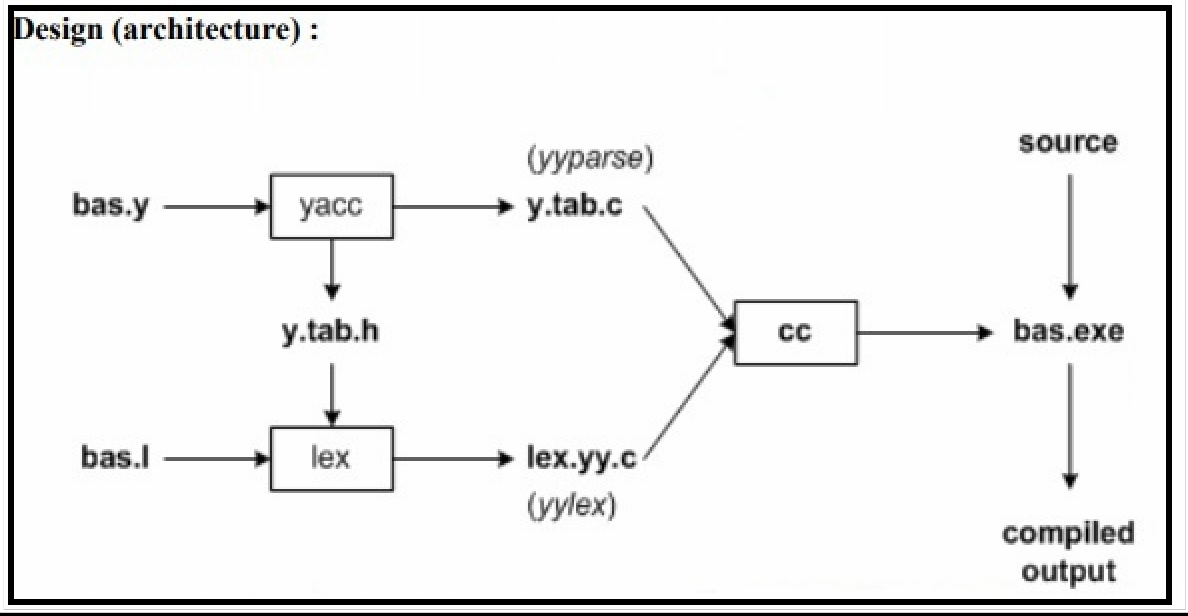
Sentences give structure to language, and in English, they come in four types: simple, compound, complex and compound-complex. When you use several types together, your writing is more interesting. Combining sentences effectively takes practice, but you'll be happy with the result.

1. The simple sentence is an independent clause with one subject and one verb. For example: we are the Indian.
2. The Compound sentence is two or more independent clause, joined with comma, semicolon & conjunctions.

## Application: -

YACC is used to generate parsers, which are an integral part of compiler.

## Design Architecture:-

****

**Conclusion:-**

Thus, I have studied lexical analyzer, syntax analysis and implemented Lex & Yacc application for Syntax analyzer to validate the given infix expression.

## Assignment No: - C1

**Aim**: - Implement Job scheduling algorithm

* 1. FCFS
  2. Shortest Job first
  3. Priority
  4. Round Robin

**Problem Statement**: - To write programs to implement FCFS, SJF, Priority & RR Scheduling Algorithms.

**Pre-requisites**:- 1. Explain Uniprocessor scheduling.

2. Define FCFS, SJF, RR, Priority.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives: -** To Simulate and compare different CPU scheduling algorithm.

* + 1. FCFS
    2. Shortest Job first
    3. Priority
    4. Round Robin

**Outcomes: -** After completion of this assignment students will be able to:

How CPU scheduling decides the processing of upcoming process request by Considering different scheduling algorithms.

## Theory:-

**Problem Explanation:**

CPU scheduling deals with the problem of deciding which of the processes in the ready queue Is to be allowed to utilize the CPU. The criteria for selection for a algorithm are,

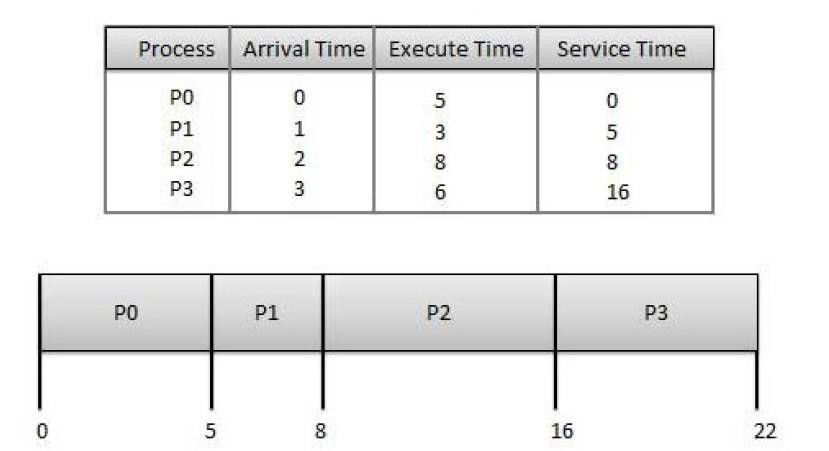
1. The maximum throughput
2. Least turnaround time.
3. Minimum waiting time.
4. Maximum CPU utilization.
5. Also the variance in response time must be minimum. In Preemptive job, a currently

Executing job can be removed and a new job can take its place, however in Non-Preemptive this is not possible.

## FIRST COME FIRST SERVE:

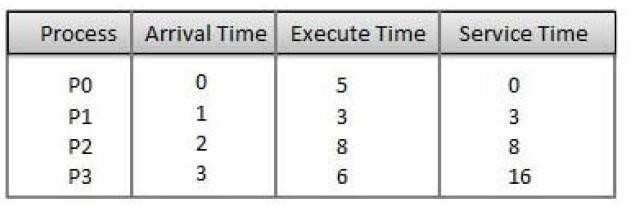
This is the simplest CPU scheduling algorithm. The Process that request the CPU first, is the one to which it is allocated first. The algorithm is implemented using a job queue. When a process Requests the CPU it is added at the tail of the job queue.

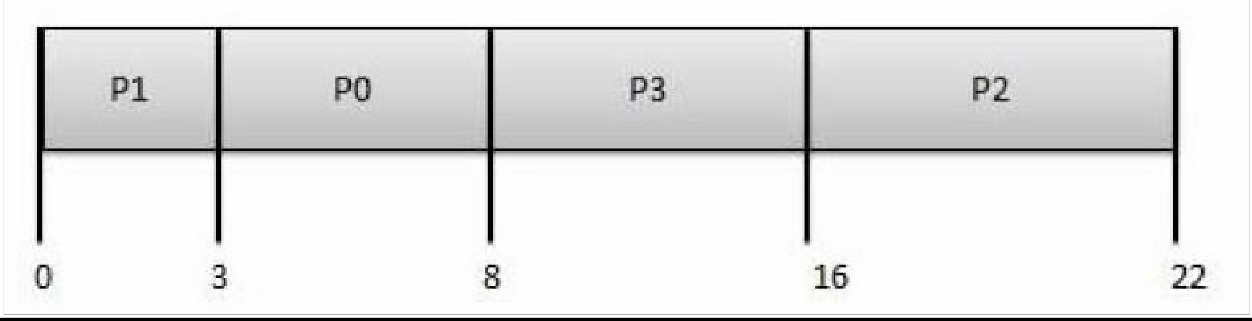
The CPU is allocated to the process at the head of the queue. However the TAT varies, which is not favorable.



## SHORTEST JOB FIRST:

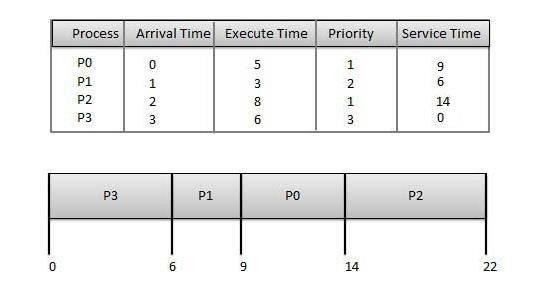
This algorithm associates with it the length of the next CPU burst. When the CPU is available, it is assigned to that job with the smallest CPU burst. This algorithm provides the minimum average waiting time. The major problem with this knows the CPU burst of a job.





## PRIORITY BASEDSCHEDULING:

* + 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.

**Wait time** of each process is as follows −

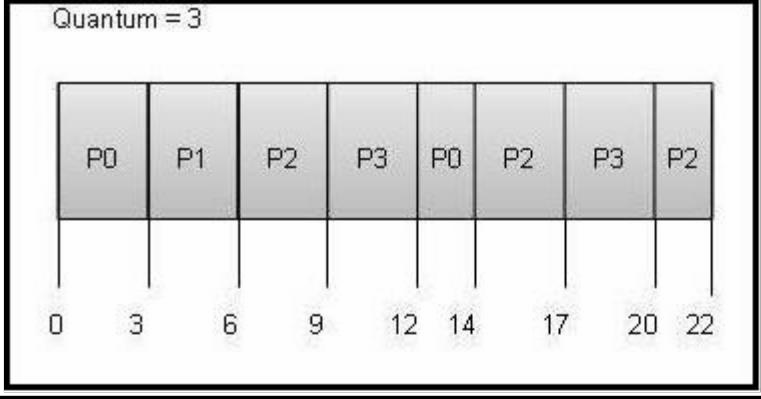
|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | 9 - 0 = 9 |
| P1 | 6 - 1 = 5 |
| P2 | 14 - 2 = 12 |
| P3 | 0 - 0 = 0 |

## SHORTEST REMAINING TIME:

* + Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
  + The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
  + Impossible to implement in interactive systems where required CPU time is not known.
  + It is often used in batch environments where short jobs need to give preference.

## ROUND ROBINSCHEDULING:

* + 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 processes.



## Implementation Logic:-

1. Choose how to represent a process & how to keep track of different parameters of all Processes.
2. Choose with which rules the scheduler will choose the next process.
3. Design an algorithm that applies the above rules. Take into consideration arrival time, Priority, quantum etc appropriately.
4. Implement the algorithm.

## Conclusion:-

Thus, I have studied FCFS, SJF, Priority & RR algorithms with examples which are based on it.

## Assignment No: - C2

**Aim**: - Bankers algorithm for deadlock detection and avoidance.

**Problem Statement**: - To write a program to implement Bankers algorithm for detection & Avoidance of deadlock.

**Pre-requisites**:- 1.Explain Deadlock prevention & avoidance.

* 1. Define deadlock with an example.
  2. Define Banker's algorithm.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit Plus, JDK 1.7 |

**Hardware Requirements**: - No

**Objectives:** - 1) To study bankers algorithm to avoid deadlock.

1. To study the safety algorithm for finding out whether or not a system is in a safe State.
2. To study the resource-request algorithm theory.

**Outcomes: -** The algorithm avoids deadlock by denying or postponing the request if it finds that accepting the request could put the system in an unsafe state.

## Theory:-

**Deadlock:** A set of processes is in deadlock state when every process in the set is waiting for an Event that can only caused by another process in the set. Examples of such processes are resources acquisition and release. Deadlock prevention algorithm prevent deadlock situation by restraining how requests can be made. The restrains ensure that at least one of the necessary conditions for deadlock can’t occur and hence deadlock can’t hold. A side effect of this method is low device utilization and reduced system throughput. An alternative method requires additional information about resources available and resources currently allocated to each process, also the future requires and release of each process to decide if the current request can be satisfied or must wait to avoid a possible future deadlock. There are various algorithms for deadlock avoidance, which differ in the amount and type of information required.

## The Banker's Algorithm:

This is a deadlock avoidance algorithm. The name was chosen since this algorithm can be used in a banking system to ensure that the bank never allocate its available cash in such a way that it can no longer satisfy further request for cash.

When a new process enters a system, it must declare the maximum no. Of instances of each resource type that it may need. This no. 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 this resource will leave the system in a safe state. The resource are allocated otherwise, the process must wait until some another process releases enough resource.

## Required Database:

Let n be the no. Of processes in the system and m be the no. Of resource type. Available: A vector of length m indicates the no. Of available resources of each type. If Available [j] =k, there are k instances of resource type Rj available.

Max: An n\*m matrix defines the maximum demand of each process. If Max [i, j] =k, then Pi May request almost k instances of resource type Rj.

Allocation: An n\*m matrix defines the no. Of resources of each type currently allocated to Each process. If Allocation [i,j]=k, then process Pi is currently allocated k instances of resource type Rj.

Need: An n\*m matrix indicates the remaining resource need of each process. If need [i, j] =k, Then Pi may need k more instance of resource type Rj to compute its task. Need [i, j] =Max [i, j]-Allocation [i, j].

## Safety algorithm:

The algorithm for finding out whether or not a system is in a safe state can be describes as follows:

Let Work and Finish be vectors of length m & n respectively. Initialize Work: =Available And Finish[i]:= false for i=1,2,...,n.

Find an i such that both Finish[i] =false Need[i] <=Work

If no such i exists, go to step 4. Work: =Work Allocation[i].

Finish[i]:=true go to step 2.

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

## Resource-Request algorithm:

Let request[i] be the request vector for process P[i], If request [i, j] =k, then process P[i] wants k Instance of resource type R[j].When a request for resource is made by process p[i], the following Actions are taken:

If Request[i] <=Need[i], go to step 2.Otherwise, raise an error condition since the process has Exceeded its maximum claim.

If Request[i] <=Available, go to step 3.otherwise, P[i] must wait, since the resource are not Available.

Have the system pretend to have allocated the request resource to process 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 the resulting resource-Allocation state is safe the transaction is completed and process P[i] is Allocated its resources. However, if new state is unsafe, then P[i] must wait for request[i] and the old resource-allocation state is resorted.

## Conclusion:

The goal of this algorithm is to handle all the requests without entering into the unsafe state. So that this algorithm is not widely used in the real world is because to use it, the operating system must know the maximum amount of resources that every process is going to need at all time.

## Assignment No: - C3

**Aim**: - Implement UNIX system calls like for process management.

**Problem Statement**: - To write a program to implement UNIX system calls like for process Management.

**Pre-requisites**:- 1.Explain concept of system call.

2. Explain state diagram working of new process.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | C Turbo or GCC |

**Hardware Requirements**: - No

**Objectives:** - 1) To understand UNIX system call.

1. To understand Concept of process management.
2. Implementation of some system call of OS.

**Outcomes: -** After completion of this assignment students will be able to:

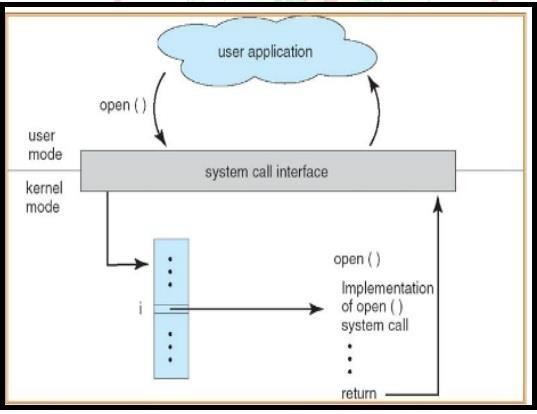
* Knowledge of System call
* Compare system call and system function
* Application of System call.

## Theory:- SYSTEM CALL:

* When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a system call.
* When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a context switch.
* Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations:

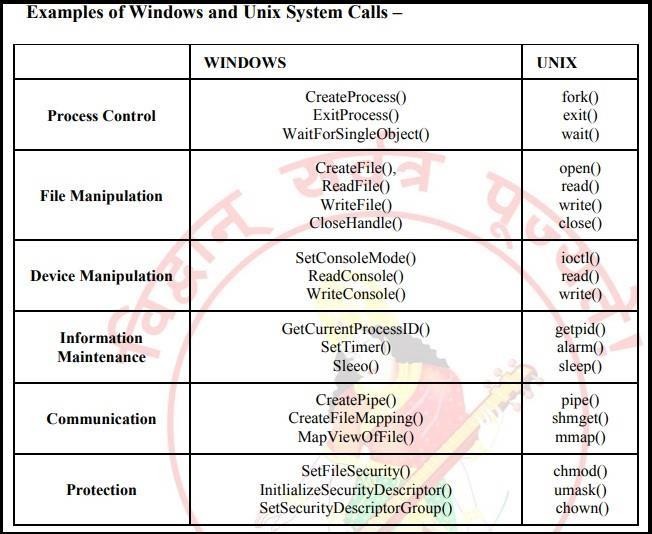
* Creating, opening, closing and deleting files in the file system.
* Creating and managing new processes.
* Creating a connection in the network, sending and receiving packets.
* Requesting access to a hardware device, like a mouse or a printer.
* To understand system calls, first one needs to understand the difference between kernel mode and user mode of a CPU. Every modern operating system supports these two modes.



## KERNEL MODE:-

* + When CPU is in kernel mode, the code being executed can access any memory address and any hardware resource.
  + In user mode, if any program crashes, only that particular program is halted.
  + That means the system will be in a safe state even if a program in user mode crashes.
  + Hence, most programs in an OS run in user mode.

## SYSTEM CALLS BASICS:-



* Since system calls are functions, we need to include the proper header files
  + E.g., for getpid() we need
    - #include<sys/types.h>
    - #include<unistd.h>
* Most system calls have a meaningful return value
  + Usually, -1 or a negative value indicates an error
  + A specific error code is place in a global variable called: errno
  + To access errno you must declare it: extern int errno;

## SYSCALLS FOR PROCESSES:

* pid\_t fork (void)
* Create a new child process, which is a copy of the current process.
* Parent return value is the PID of the child process.
* Child return value is 0.
* int execl (char \*name, char \*arg0, ..., (char \*) 0)
* Change program image of current process.
* Reset stack and free memory.
* Start at main ().
* Also see other versions: execlp (), execv (), etc.
* pid\_t wait (int \*status)
* Wait for a child process (any child) to complete.
* Also see waitpid () to wait for a specific process.
* void exit (int status)
* Terminate the calling process.
* Can also achieve with a return from main ().
* int kill (pid\_t pid, int sig)
* Send a signal to a process.
* Send SIGKILL to force termination

## UNIX SYSTEM CALLS :-

* **Ps command :**

The ps (i.e., process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs).

A process, also referred to as a task, is an executing (i.e., running) instance of a program. Every process is assigned a unique PID by the system.

**The basic syntax of ps is:** ps [options]

## fork command :

The fork () system call is used to create processes. When a process (a program in execution) makes a fork () call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.

## Join command :

The join command in UNIX is a command line utility for joining lines of two files on a common field. It can be used to join two files by selecting fields within the line and joining the files on them. The result is written to standard output.

**Join syntax:** Join [option]….. file1 file2

## Exec() command :

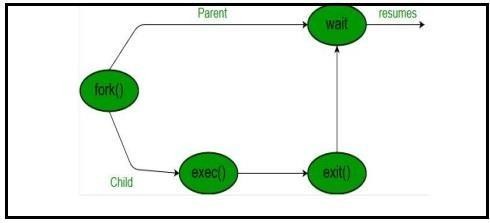
The exec () system call is also used to create processes. But there is one big difference between fork () and exec () calls. The fork () call creates a new process while preserving the parent process. But, an exec () call replaces the address space, text segment, data segment etc. of the current process with the new process.

## Wait() command :

A call to wait () blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction

Child process may terminate due to any of these:

* It calls exit ();
* It returns (an int) from main.
* It receives a signal (from the OS or another process) whose default action is to terminate.



## Conclusion:

Thus, the process system call program is implemented and studied various system calls.

## Assignment No: - C4

**Aim**: - Study assignment on process scheduling algorithms in Android and Tizen.

**Problem Statement**: - To study assignment on process scheduling algorithms in Android and Tizen.

**Objectives:** - 1) To understand Android OS.

1. To understand Tizen OS.
2. To understand Concept of process management.

**Outcomes: -** After completion of this assignment students will be able to:

* Knowledge of Android and tizen OS.
* Study of process management in android and tizen OS.
* Application of android and tizen OS.

## Theory:- ANDROID OS:-

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touch screen mobile devices such as smart phones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The operating system has since gone through multiple major releases, with the current version being 8.1 "Oreo", released in December 2017.

The android is a powerful operating system and it supports large number of applications in Smartphone. These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it’s free and any one can use it. The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

The android development supports with the full java programming language. Even other packages that are API and JSE are not supported. The first version 1.0 of android development kit (SDK) was released in 2008 and latest updated version is jelly bean.

## Some android versions:

* Gingerbread (2.3)
* Honeycomb (3.0)
* Ice Cream Sandwich (4.0)
* Jelly Bean (4.3/4.2/4.1)
* Kit Kat (4.4)
* Lollipop (5.0)
* Marshmallow (6.0)
* Nougat (7.0)
* Oreo (8.0)

## Advantages:-

1. Support 2D & 3D Graphics.
2. Support multiple languages.
3. Java support.
4. Faster web browser.
5. Support audio, video etc.

## Disadvantages:-

1. Slow response.
2. Heat.
3. Advertisement etc.

## TIZEN OS:-

Tizen is a mobile operating system developed by Samsung that runs on a wide range of Samsung devices, including Smartphone’s; tablets; in-vehicle infotainment (IVI) devices; smart televisions; smart cameras; smart watches; Blue-ray players; smart home appliances (refrigerators, lighting, washing machines, air conditioners, ovens/microwaves); and robotic vacuum cleaners.

In 2010 Samsung was developing the Samsung Linux Platform (SLP) for the LiMo Foundation, whilst Intel and Nokia were leading the MeeGo project, another open source Linux mobile OS. In 2011 the MeeGo project was abandoned by its peers with Intel joining forces with Samsung to create Tizen, a new project based on code from SLP. The Linux Foundation also cancelled support of MeeGo in favor of Tizen. In 2013 Samsung merged its homegrown Bada project into Tizen.

The Tizen Association was formed to guide the industry role of Tizen, including requirements gathering, identifying and facilitating service models, and overall industry marketing and education. Members of the Tizen Association represent major sectors of the mobility industry. Current members include: Fujitsu, Huawei, Intel, KT, NEC Casio, NTT DoCoMo, Orange, Panasonic, Samsung, SK Telecom, Sprint and Vodafone.

Samsung announced in November 2016 that they would be collaborating with Microsoft to bring

.Net support to Tizen.

Samsung is currently the only Tizen member developing and using the operating system.

On January 1, 2012, the LiMo Foundation was renamed Tizen Association. The Tizen Association works closely with the Linux Foundation, which supports the Tizen open source project.

* + April 30, 2012: Tizen 1.0 released.
  + February 18, 2013: Tizen 2.0 released.
  + May 20, 2017: Tizen 3.0 released.

The first Tizen tablet was shipped by Systena in October 2013. Part of a development kit exclusive to Japan, it was a 10-inch quad-core ARM with 1920×1200 resolution.

On February 21, 2016, Samsung announced the Samsung Connect Auto, a connected car solution offering diagnostic, Wi-Fi, and other car-connected services. The device plugs directly into the OBD-II port underneath the steering wheel.

## Android vs. Tizen Operating system:

1. **Easy and Convenient Navigation:** Scrolling and navigation becomes smooth with Tizen.
2. **Fast and Lightweight:** Tizen Operating System is easy to operate and fast as compared to Google’s Android Wear.
3. **UI:** Touch Wiz UI.
4. **Resizable boxes:** One of the amazing features of Tizen is its ability to dynamically resize the Icons on screen to display more information or less.
5. **Enhanced Processors:** Tizen 3.0 will bring 64 bit processors with it, compatible with x86 Processors and 64 bit RAM, which Google is also anticipating with its Update.
6. **IoT Devices:** Tizen 3.0 is compatible with Artik cloud which will extend cloud services for IoT devices.

## Advantages of using Tizen OS:-

* + It is an open source Operating System.
  + The OS is Compatible with various mobile platforms. Application built on Tizen can be launched on iOS and Android too with few changes.
  + The Tizen OS is so flexible to offer many applications and adapt too, with little changes.
  + Immense personalization capability supported by ARM x86 processor.

## PROCESS SCHEDULING ALGORITHMS IN ANDROID AND TIZEN OS:

1. **Normal scheduling:-**

Android is based on Linux and uses the Linux kernel’s scheduling mechanisms for determining scheduling policies. This is also true for Java code and threads. The Linux’s time sliced scheduling policy combines static and dynamic priorities. Processes can be given an initial priority from 19 to -20 (very low to very high priority). This priority will assure that higher priority processes will get more CPU time when when needed. These levels are however dynamic, low level priority tasks that do not consume their CPU time will fine their dynamic priority increased. This dynamic behavior results is an overall better responsiveness.

In terms of dynamic priorities it is ensured that lower priority processes will always have a lower dynamic priority than processes with real-time priorities.

Android uses two different mechanisms when scheduling the Linux kernel to perform process level scheduling.

## Real-time scheduling:-

The standard Linux kernel provides two real-time scheduling policies, SCHED\_FIFO and SCHED\_RR. The main real-time policy is SCHED\_FIFO. It implements a first-in, first-out scheduling algorithm. When a SCHED\_FIFO task starts running, it continues to run until it voluntarily yields the processor, blocks or is preempted by a higher-priority real-time task. It has no time slices. All other tasks of lower priority will not be scheduled until it relinquishes the CPU. Two equal-priority SCHED\_FIFO tasks do not preempt each other. SCHED\_RR is similar to SCHED\_FIFO, except that such tasks are allotted time slices based on their priority and run until they exhaust their time slice. Non-real-time tasks use the SCHED\_NORMAL scheduling policy (older kernels had a policy named SCHED\_OTHER).

## Fixed-priority pre-emptive scheduling:-

Fixed-priority preemptive scheduling is a scheduling system commonly used in real-time systems. With fixed priority preemptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to execute.

Preemptive scheduling is often differentiated with cooperative scheduling, in which a task can run continuously from start to end without being preempted by other tasks. To have a task switch, the task must explicitly call the scheduler. Cooperative scheduling is used in a few RTOS such as Salvo or TinyOS.

## Conclusion:

Thus, I have studied concept of process scheduling of Android and Tizen Operating System.

## Assignment No: - D

**Aim**: - Implementing page replacement algorithm.

* 1. LRU
  2. Optimal

**Problem Statement**: - To write a program to implement LRU & Optimal algorithm for Page Replacement.

.

**Pre-requisites**:- 1. Explain the concept of virtual memory.

1. Define page replacement algorithm: LRU & Optimal.
2. Explain address translation in paging system.
3. Explain Belady's Anomaly.

## Software Requirements:-

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | Edit plus, JDK 1.6 |

**Hardware Requirements**: - No

**Objectives:** - 1) To study concept of Pages replacement.

2) To study the best algorithm for page replacement: Optimal, LRU

**Outcomes: -** After completion of this assignment students will be able to:

How to divided the program in equal size pages to optimized its execution by Replacing the pages by considering different page replacement algorithms.

## Theory:-

Whenever there is a page reference for which the page needed in memory, that event is called page fault or page fetch or page failure situation. In such case we have to make space in memory for this new page by replacing any existing page. But we cannot replace any page. We have to replace a page which is not used currently. There are some algorithms based on them. We can select appropriate page replacement policy. Designing appropriate algorithms to solve this problem is an important task because disk I/O is expensive.

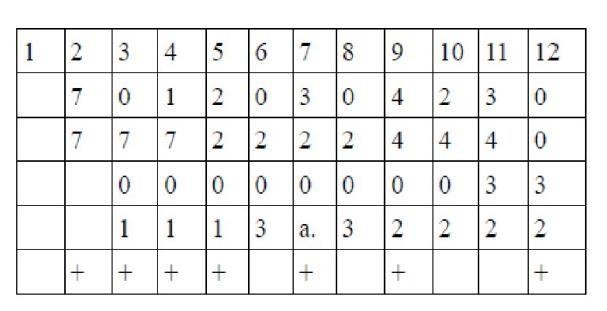
## There are several algorithms to achieve.

1. Last recently used (LRU)
2. Optimal

## LRU page replacement:

The main difference between FIFO and optimal page replacement is that the FIFO algorithms Uses the time when the page was brought in to memory and the. Optimal algorithm uses the time when a page is to be used. If we use the recent past as an approximation of the future then we will replace the page that has not been used for the longest period of time. This approach is called as least recently used (LRU) algorithm.

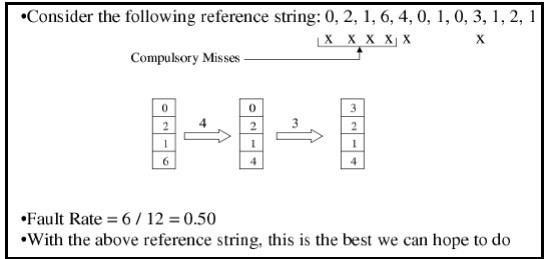
LRU replacement associates with each page must be replaced. LRU chooses that page that has Not been used for the longest period of time. Now, consider reference string 7,0,1,2,0,3,4,2,3,0,3 with three memory frames or blocks The first three reference cases page fault that fill the empty frames.



## Optimal page replacement:

The algorithm has lowest page fault rate of all algorithm. This algorithm state that: Replace the page which will not be used for longest period of time i.e future knowledge of reference string is required.

* + Often called Balady's Min Basic idea: Replace the page that will not be referenced for the Longest time.
  + Impossible to implement



## ALGORITHM FOR LRU:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value



1. Compare counter label and stack
2. Select the Optimal page by counter value
3. Stack them according the selection.
4. Print pages with fault pages.
5. Stop the process.

## Conclusion:

Thus, I have studied page replacement algorithms of LRU & Optimal.