AISSMS COE, Pune 411001

Computer Engineering Department

Third Year

310243: System Programming & OS Laboratory

Laboratory Manual

AISSMS COE, Pune 411001

Certificate

This is certify that _____

Mr./Miss./Mrs. ***Pratik Pingale***

Class ***TE*** *Roll No.* ***19CO056***

Has completed all the practical work satisfactorily in the subject of Laboratory Practice – I as prescribed by the Pune University in the academic year 2020-2021 in the Department of Computer Engineering of the institute.

	Prof. Monali Deshmukh	Prof. Anil Kadam
Date: 14/12/2021	Subject In-charge	Head of Department

AISSMS COE, Pune 411001

INDEX

Sub: 310243: System Programming & OS Laboratory **Class:** T.E.Computer

Assignment No :- 1

Title:
Design Suitable data Structure and implement pass-I of a two-pass assembler for pseudo-machine in Java using Object oriented feature. Implementation Should Consist of a few instruction from each Category and few assembler directives.

Objectives:
- To understand data Structures to be used in pass I of an assembler.
- To implement pass I of an assembler

Problem Statement:
Write a ~~prolog~~ program to Create pass-I Assembler

Outcomes:
After Completion of this assignment Students will be abe to:
- Understand the Capncept of Pass-I Assembler
- Understand the programming language of Java.

Software Requirements:
- Linux OS, JDK 1.7

Hardware Requirement:
- 4GB RAM, 500GB HDD

Assembly Language Statement :-

These are three types of Statements Imperative Declarative, Assembly directives. An imperative Statements indicates an action to be performed during the execution of assembled program. Each imperative Statement usually translates into one machine instruction. Declarative Statement eg DS reserves area of memory and associates names with them. DC Constructs memory word Containing Constants. Assembler directives instruct the assembler to perform Certain actions during assembly of a program.

e.g. START <constant> directive indicates that the first world of the target program generated by assembler should be placed at memory word with address

Function of Analysis And Synthesis Phase :-

Analysis Phase :-

Isolate the label operation Code and operand fields of a Statement.

Enter the Symbol found in label field and address of next available machine word into Symbol table.

Validate the mnemonic Operation Code by Looking it up in the mnemonics table. Determine the machine Storage requirements of the Statement by Considering the mnemonic Operation Code and operand fields of the Statement.

Design of a Two Pass Assembler :-

Task performed by the passes of two-pass assembler are as follows:

Pass I :-

Seperate the Symbol, mnemonic opcode and operand fields.

Determine the Storage-required for every assembly language Statement and update the location Counter.

Build the Symbol table and the literal table.

Construct the intermediate Code for every assembly language Statement.

Pass II :-

Synthesize the target Code by processing the intermediate Code generated during pass I

Intermediate code Representation :

The intermediate code Consists of a Set of IC units, each IC unit Consisting of the following three, fields.

1. Address
2. Representation of the mnemonic opcode
3. Representation of operands

Mnemonic field -

The mnemonic field Contains a pair of the form:

- Statement class , Code

Where Statement class can be one of IS, DL and AD Standing for imperative Statement, declaration Statement and assembler directive, respectively. for imperative Statement, code is the instruction opcode in the machine language for declaration and assembler directives, following are the Codes.

Declaration Statements		Assembler directives	
		START	01
DC	01	END	02
DS	02	ORIGIN	03
		EQU	04
		LTORG	05

Algorithms : Pass 1

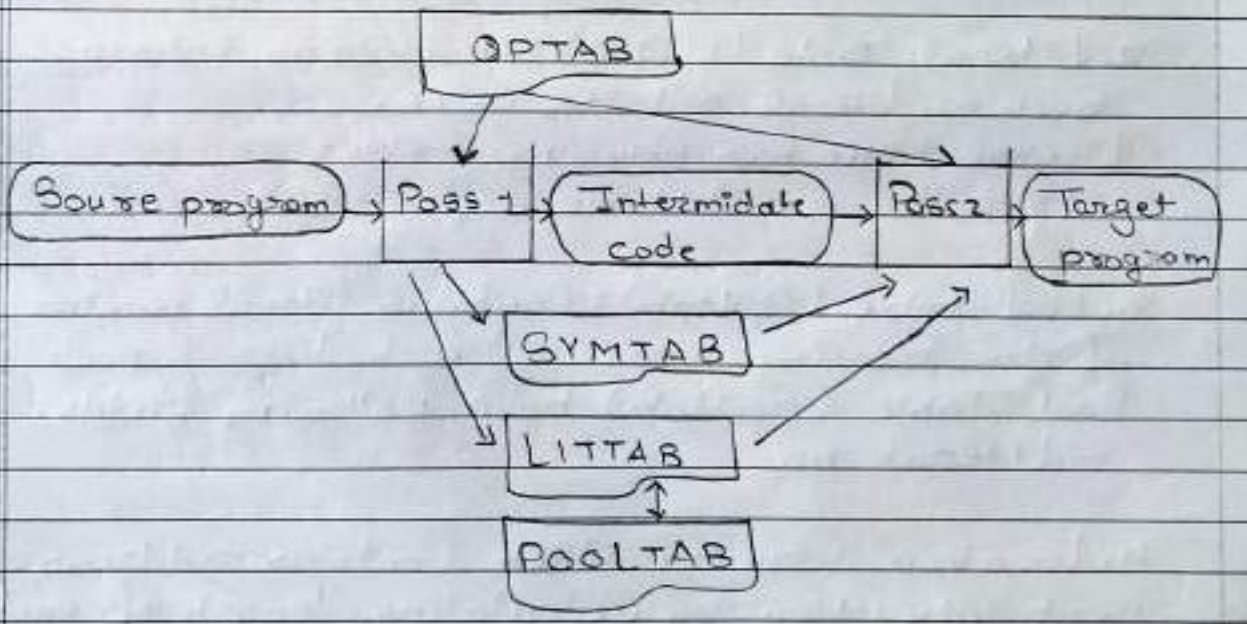
- Initialize location Counter, entries of all tables as zero.
- Read Statements from input file one by one.
- While next Statement is not END Statement.

1. Tokenize or Seprate out input Statement as label, numonic, operand 1, operand 2.
2. If label is present insert label into Symbol Table
3. If the Statement is LTORG Statement process it by making it's entry into literal table pool table, and allocation memory.
4. If Statement is START or ORIGEN Process location Counter accordingly.
5. If an EQU Statement, assign Value to Symbol by Correcting entry in Symbol table.

by Considering the mnemonic operation Code and operand fields of the statement.
Calculate the address of the address of the first machine word following the target Code generated for this statement.

Synthesis Phase :
Obtain the machine operation Code Corresponding to the mnemonic operation Code by Searching the mnemonic table.
Obtain the address of the operand from the Symbol table.
Synthesize the machine instruction or the machine form of the Constant as the Case may be.

Data Structures of a two Pass Assembler :



Data Structure of Assembler:

a) Operation Code table (OPTAB): This is used for storing mnemonic, operation Code and class of instruction.

Structure of OPTAB is a follows

b) Data Structure updated during translation: Also Called as translation time data Structure. They are

1. Symbol table (SYMTAB): It Contains entires Such Symbol, it's address and value.
Symbol Table have following fields:
- Name of Symbol - Symbol Address - Value

2. Literal Table (LITTAB): It contains entires Such as literal and it's value.
Literal table has following fields:
- Literal - Address of Literal

3. Pool table (Pooltab): Contains literal number of the Starting literal of each literal pool.
Pool Table (pooltab) have following fields.
- Literal no

4. Location Counter which Contains address of next instruction by calculating length of each instruction.

6. For declarative Statement update Code, Size and allocation Counter.
7. Generate intermediate Code.
8. Pass this intermediates Code to pass-2.

Conclusion : Thus, I have studied visual programming and implemented dynamic link library application for arithmetic operation.

Assignment No. 1(A)

Name: Pratik Pingale

Class: TE Roll No. 19CO056
Subject: SPOS
****
***
*

Problem Statement: 1. (A)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.

SUPPORT CLASSES:-

INSTtable.java

```
import java.util.HashMap; public

class INSTtable {

        HashMap<String, Integer> AD,RG,IS,CC,DL;
        public INSTtable()
        {
                AD=new HashMap<>(); CC =
                new HashMap<>(); IS = new
                HashMap<>(); RG = new
                HashMap<>(); DL=new
                HashMap<String, Integer>();
                DL.put("DC", 01);
                DL.put("DS", 02);
                IS.put("STOP",0);
                IS.put("ADD",1);
                IS.put("SUB",2);
                IS.put("MULT",3);
                IS.put("MOVER",4);
                IS.put("MOVEM",5);
                IS.put("COMP",6);
                IS.put("BC",7);
                IS.put("DIV",8); IS.put("READ",9);
                IS.put("PRINT",10);
                CC.put("LT",1);
                CC.put("LE",2);
                CC.put("EQ",3);
                CC.put("GT",4);
                CC.put("GE",5);
                CC.put("ANY",6);
                AD.put("START",1);
                AD.put("END",2);
                AD.put("ORIGIN",3);
                AD.put("EQU",4);
                AD.put("LTORG",5);
                RG.put("AREG",1);
                RG.put("BREG",2);
                RG.put("CREG",3);
                RG.put("DREG",4);

        }
        public String getType(String s)
        { s=s.toUpperCase(); if(AD.containsKey(s))
```

```
return "AD";
```

```
                else if(IS.containsKey(s)) return
                "IS"; else if(CC.containsKey(s))
                return
                        "CC"; else
                if(DL.containsKey(s))
                return
                        "DL"; else
                if(RG.containsKey(s))
                return
                                        "RG";
                                    return "";

        }

        public int getCode(String s)
        { s = s.toUpperCase(); if(AD.containsKey(s))
                return AD.get(s); else
                if(IS.containsKey(s)) return
                        IS.get(s);
                else
                if(CC.containsKey(s))
                return
                        CC.get(s);
                else
                if(DL.containsKey(s))
                return
                        DL.get(s);
                else
                if(RG.containsKey(s))
                return
                        RG.get(s); return -
                1;
        }

}
```

TableRow.java

```
        public class TableRow { String
        symbol; int addess,index; public
        String getSymbol() { return
        symbol;

        }
        public TableRow(String symbol, int addess) {
        super(); this.symbol = symbol; this.addess =
        addess; index=0; } public void setSymbol(String
        symbol) { this.symbol = symbol;
    }



        public TableRow(String symbol, int addess, int index) {
                super(); this.symbol = symbol; this.addess = addess;
                this.index = index;
        } public int getAddess() { return
        addess;
        } public void setAddess(int addess) {
```

```
            this.addess = addess;
}
public int getIndex() { return index;
} public void setIndex(int index) { this.index
= index;
```

```
        }
        }
```

MAIN CODE:-

PassOne.java

```
import java.io.BufferedReader; import
java.io.BufferedWriter; import
java.io.FileReader; import
java.io.FileWriter; import
java.io.IOException; import
java.util.ArrayList;
import
java.util.LinkedHashMap
; public class PassOne {
        int lc=0;
        int libtab_ptr=0,pooltab_ptr=0; int
        symIndex=0,litIndex=0;
        LinkedHashMap<String, TableRow>
        SYMTAB;
        ArrayList<TableRow> LITTAB;
        ArrayList<Integer> POOLTAB;
        private BufferedReader br;

        public PassOne()
        {
                SYMTAB =new LinkedHashMap<>();
                                LITTAB=new
                ArrayList<>(); POOLTAB=new
                ArrayList<>(); lc=0;
                                POOLTAB.add(0);
        }
        public static void main(String[] args) { PassOne
                one=new PassOne(); try
                { one.parseFile();
                }
    catch (Exception e) {
System.out.println("Error: "+e);// TODO: handle exception
                }
        }
        public void parseFile() throws Exception
        {String prev="";
                String line,code;
                br = new BufferedReader(new FileReader("input.asm")); BufferedWriter
                bw=new BufferedWriter(new
FileWriter("IC.txt"));                                  INSTtable lookup=new INSTtable();
                                                        while((line=br.readLine())!=null)
                                                        {

                                        String parts[]=line.split("\\s+"); if(!parts[0].isEmpty()) //processing of label
                                        { if(SYMTAB.containsKey(parts[0])) SYMTAB.put(parts[0],
new TableRow(parts[0], lc, SYMTAB.get(parts[0]).getIndex()));
                                                                else
                                                                SYMTAB.put(parts[0],new TableRow(parts[0],
lc, ++symIndex));                                                       }


                                                                        if(parts[1].eq
                                                                        uals("LTOR
                                                                        G"))
                                                                        { int
                                                                        ptr=POOLT
                                                                        AB.get(poolt
                                                                        ab_ptr);
                                                                        for(int
```

```
j=ptr;j<libta
  b_ptr;j++)
        {
     lc++;
```

```
                                        LITTAB.set(j, new
TableRow(LITTAB.get(j).getSymbol(),lc));

        code="(DL,01)\t(C,"+LITTAB.get(j).symbol+")";
                                    bw.write(code+"\n");
                                        }
                                    pooltab_ptr++; POOLTAB.add(libtab_ptr);
                                }
                                    if(parts[1].equals("START"))
                                  { lc=expr(parts[2]);
                                     code="(AD,01)\t(C,"+lc+")";
                                     bw.write(code+"\n");
                                      prev="START";
                                                }
                                                    else if(parts[1].equals("ORIGIN"))
                                                {
                                                        lc=expr(parts[2]);
                                                    String splits[]=parts[2].split("\\+"); //Same
for - SYMBOL //Add code


        code="(AD,03)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")"+Integer.pars eInt(splits[1]);
                                        bw.write(code+"\n");
                                  }
                                  //Now for EQU
                                  if(parts[1].equals("EQ
                                  U"))
                                  {
                                                                    int loc=expr(parts[2]);
                                                                     //below If
generated for them
                                                                    conditions are
{
                                                                    optional as no IC is

                                                                    if(parts[2].contains(

                                                                    "+"))

                                                                                String
                                                                                splits[]=parts[2].split("\\+");
code="(AD,04)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")"+Integer.pars eInt(splits[1]);

                                                    }
                                            else if(parts[2].contains("-"))
                                                    {
                                                        String splits[]=parts[2].split("\\-");

        code="(AD,04)\t(S,"+SYMTAB.get(splits[0]).getIndex()+")- "+Integer.parseInt(splits[1]);
                                                }
                                                else
                                                        {

        code="(AD,04)\t(C,"+Integer.parseInt(parts[2]+")");
                                                        }
                                                bw.write(code+"\n");
                                                        if(SYMTAB.containsKey(parts[0]))
                                                        SYMTAB.put(parts[0], new
TableRow(parts[0],loc,SYMTAB.get(parts[0]).getIndex())) ;
                                                    }
                                        SYMTAB.put(parts[0], new
                                                    else
```

```
TableRow(parts[0],loc,++symIndex));
                                        }

                                                if(parts[1].equals("DC"))
                                        {
```

```
                                                    l
                                                    c
                                                    +
                                                    +
                                                    ;
                                                    i
                                                    n
                                                    t
constant=Integer.parseInt(parts[2].replace("","")); code="(DL,01)\t(C,"+constant+")";
                                                    bw.write(code+"\n");
                                            }
                                                        else if(parts[1].equals("DS"))
                                            {


                                                    int size=Integer.parseInt(parts[2].replace("",
"","));

                                                                code="(DL,02)\t(C,"+size+")"; bw.write(code+"\n");
                                                            /*if(prev.equals("START"))
                                                                { lc=lc+size-

        1;//System.out.println("here");



                                                                                    }
                                                                                    else

*/      lc=lc+size;
                                                                                        prev="";
                                                                    }
                                                                        if(lookup.getType(parts[1]).equals("IS"))
                                            { code="(IS,0"+lookup.getCode(parts[1])+")\t"; int
                                                    j=2; String code2="";
                                                    while(j<parts.length)
                                                    { parts[j]=parts[j].replace(",", "");

                        if(lookup.getType(parts[j]).equals("RG"))
                                                            {

                                                } else
                    code2+=lookup.getCo
                    de(parts[j])+"\t";
                                                                                    if(parts[j].contains("="))
                                            {                                       {


parts[j]=parts[j].replace("=",
"").replace("", "");

                                                                                            LITTAB.add(
                                                    new TableRow(parts[j], -1,++litIndex));
                                                    libtab_ptr++;


                        code2+="(L,"+(litIndex)+")";

                            ind=SYMTAB                                                          else
                                        TableRow(part
if(SYMTAB.c                             s[j],-1,++symI
                            .get(parts[j]).g            ndex));
                                        ind=SYMTAB
ontainsKey(pa                           .get(parts[j]).g
                            etIndex();  etIndex();                                  }
                                                                                    lc++;
rts[j]))
```

```
                    {                   int
                                                        code2+=
} j++;                                                  "(S,0"+ind+")";
                    }
                    else
                    {
                                                        SYMTAB.
                                                        put(parts[j]
                                                        , new int
                                                        code2+=
                                                        "(S,0"+ind
                    }                                   +")";
```

```
                                        code=code+code2; bw.write(code+"\n");
                                }

                                    if(parts[1].equals("END"))
                                   { int ptr=POOLTAB.get(pooltab_ptr);
                                      for(int j=ptr;j<libtab_ptr;j++)
                                        {     lc++;     LITTAB.set(j,     new
TableRow(LITTAB.get(j).getSymbol(),lc));

        code="(DL,01)\t(C,"+LITTAB.get(j).symbol+")";
                                                    bw.write(code+"\n");
                                        } pooltab_ptr++;
                                        POOLTAB.add(libtab_ptr);
                                        code="(AD,02)";
                                        bw.write(code+"\n");
                                }

                        }
                        bw.close();
                        printSYMTA
                        B();
                        //Printing Literal
                        table
                        PrintLITTAB();
                        printPOOLTAB();
        }
        void PrintLITTAB() throws IOException
        {
                        BufferedWriter bw=new BufferedWriter(new
                        FileWriter("LITTAB.txt")); System.out.println("\nLiteral Table\n");
                        //Processing LITTAB
                        for(int i=0;i<LITTAB.size();i++)
                        {
                                TableRow row=LITTAB.get(i);

        System.out.println(i+"\t"+row.getSymbol()+"\t"+row.getAddes
        s());
        bw.write((i+1)+"\t"+row.getSymbol()+"\t"+row.getAddess()+"\
        n");
                        } bw.close();
        }
        void printPOOLTAB() throws IOException
        {
                        BufferedWriter bw=new BufferedWriter(new
                        FileWriter("POOLTAB.txt")); System.out.println("\nPOOLTAB");
                        System.out.println("Index\t#first");
                        for (int i = 0; i < POOLTAB.size(); i++) {
                                System.out.println(i+"\t"+POOLTAB.get
                                (i));
                                bw.write((i+1)+"\t"+POOLTAB.get(i)+"\
                                n");
                        } bw.close();
        }
        void printSYMTAB() throws IOException
        {
                        BufferedWriter bw=new BufferedWriter(new FileWriter("SYMTAB.txt"));
                        //Printing Symbol Table java.util.Iterator<String>
iterator = SYMTAB.keySet().iterator();
                        System.out.println("SYMBOL TABLE");
                        while (iterator.hasNext()) {
                                String key = iterator.next().toString(); TableRow
                                value = SYMTAB.get(key);

                                System.out.println(value.getIndex()+"
\t" + value.getSymbol()+"\t"+value.getAddess());
```

```
                                        bw.write(value.getIndex()+"
\t" + value.getSymbol()+"\t"+value.getAddess()+"\n");
                                } bw.close();
```

```
        }
        public int expr(String str)
        {
                int temp=0; if(str.contains("+"))
                {
                        String splits[]=str.split("\\+");

        temp=SYMTAB.get(splits[0]).getAddess()+Integer.parseInt(splits[1]);
                }
                else if(str.contains("-"))
                {
                                                        String
                                                splits[]=str.split(
                     (Integer.parseInt(splits[1]));     "\\-");
                                                } else  temp=SYMTAB
                                                .get(splits[0]).ge
                                                tAddess()-
                { temp=Integer.parseInt(str);
                }

                return temp;
        }
}
```

INPUT AND OUTPUT:-

IC.txt

```
(AD,01) (C,100)
(DL,02) (C,3)
(IS,04) 1       (S,03)
(IS,01) 1       (S,04)
(IS,05) 1       (S,05)
(AD,04) (S,1)+1
(IS,010)        (S,05)
(AD,03) (S,6)+1
(IS,00)
(DL,01)
(C,19)
(DL,01)(
C,17)
(AD,02)
```

INPUT.asm

```
ST

ART

100
```

```
A       DS      3
L1      MOVER
        AREG,B ADD
        AREG,
        C
        MOVEM AREG,    D
D       EQU     A+1
L2      PRINT
        D
        ORIGIN
        L2+1
        STOP
B       DC      '19
C       DC                                  '17
        END
```

POOLTAB.txt

1	0
2	0

SYBMTAB.txt

1	A	100
2	L1	103
3	B	108
4	C	109
5	D	101
6	L2	106

Assignment No :- 2

Title : Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 should be input for this Assignment.

Objectives:

- To understand data structures to be used in pass II of an assembler
- To implement pass I of an assembler

Problem Statement:

Write a program to Create pass-II Assembler

Outcomes:

After Completion of this assignment Students will be able to:

- Understand the Concept of Pass-II Assembler
- Understand the programming language of Java.

Software and Hardware Requirements:

- Software : Linux OS, jdk 1.7
- Hardware : 4GB Ram, 500 Gb Hdd

Theory:

Design of a Two Pass Assembler :-

Tasks performed by the passes of two-pass assembler are as follows :-

Pass I :-

Separates the Symbol, mnemonic opcode and operand fields.

Determine the storage-required for every assembly language statement and update the location Counter.

Build the Symbol table and the literal table.

Construct the intermediate Code for every assembly language Statement.

Pass II :-

Synthesize the target Code by processing the intermediate Code generated during pass I

Data Structure used by Pass II :

1. OPTAB : A table of mnemonic opcodes and related information
2. SYMTAB : The Symbol Table.
3. POOL-TAB and LITTAB : A table of literals used in the program
4. Intermediate Code generated by Pass I
5. Output file Containing Target Code / error listing

Algorithms :

1. Code_area_address = address of Code area;
   Pooltab_ptr = 1
   loc_cntr = 0
2. While next Statement is not an END Statement
   a) clear the machine_code_buffer
   b) if an LTORG Statement
      I) process literals in LITTAB [PoolTable]
         LITTAB [POOLTAB [pooltab-ptr+1]]-1
         Similar to processing of Constants in a dc Statement.
      II) Size = Size of memory area required for literals
      III) pooltab_ptr = pooltab_ptr+1
   c) if a START or ORIGIN Statement then
      I) loc-cntr = value Specified in Operand field
      II) Size = 0;
   d) if a declaration Statement
      I) if a DC Statement then assemble the Constant in machine_code_buffer
      II) Size = Size of memory area required by DC or DS;
   e) if an imperative Statement then
      I) get operand address from SYMTAB or LITTAB
      II) Assemble instruction in machine Code buffer.
      III) Size = Size of instruction;
      then

I) move Contents of machine code - buffer to the address code-area-address + loc-cntr

II) loc-cntr = loc_cntr + Size.

3) (Processing of END Statement)

Conclusion :

Thus, I have Studied visual programming and implemented dynamic link library application for arithmetic operation.

Assignment No.
2(A)

Name: Pratik Pingale
Class: TE Roll No. 19CO056
Subject: SPOS

****

**** **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**.**

```
import
java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;

public class Pass2 { public static void main(String[]
        Args) throws IOException{
                BufferedReader b1 = new BufferedReader(new
            FileReader("intermediate.txt")); BufferedReader b2 = new
            BufferedReader(new FileReader("symtab.txt")); BufferedReader b3 = new
            BufferedReader(new FileReader("littab.txt"));
            FileWriter f1 = new FileWriter("Pass2.txt");
            HashMap<Integer, String> symSymbol = new HashMap<Integer,
            String>(); HashMap<Integer, String> litSymbol = new
            HashMap<Integer, String>(); HashMap<Integer, String> litAddr =
            new HashMap<Integer, String>(); String s; int
            symtabPointer=1,littabPointer=1,offset;
            while((s=b2.readLine())!=null){ String word[]=s.split("\t\t\t");
            symSymbol.put(symtabPointer++,word[1]);
            while((s=b3.readLine())!=null){ String word[]=s.split("\t\t");
            litSymbol.put(littabPointer,word[0]);
            litAddr.put(littabPointer++,word[1]);
            }
            while((s=b1.readLine())!=null){
                if(s.substring(1,6).compareToIgnoreCase("IS,00")==0){
                f1.write("+ 00 0 000\n");
                }
                else if(s.substring(1,3).compareToIgnoreCase("IS")==0){
                        f1.write("+ "+s.substring(4,6)+" ");
```

```
if(s.charAt(9)==')'){
f1.write(s.charAt(8)+" ");
offset=3;
}
else{
```

```
f1.write("0
");
offset=0;
                        }
                        if(s.charAt(8+offset)=='S')

f1.write(symSymbol.get(Integer.parseInt(s.substring(10+offset,s.length()-1)))+"\n");
else f1.write(litAddr.get(Integer.parseInt(s.substring(10+offset,s.length()- 1)))+"\n");
                }
                 else
                        if(s.substring(1,6).compareToIgnoreCase("
                        DL,01")==0){ String
                        s1=s.substring(10,s.length()-1),s2=""; for(int
                        i=0;i<3-s1.length();i++) s2+="0";
        s2+=s1;
        f1.write("+ 00 0 "+s2+"\n");
                }
                 e
        lse{
        f1.write("\n
        ");
                }
             }
             f1.close();
             b1.close();
             b2.close();
             b3.close(); }

/*
OUTP
UT:
aditya@aditya-1011PX:~/Desktop/aditya_SPOS/Turn1/A
2$ javac Pass2.java aditya@aditya
1011PX:~/Desktop/aditya_SPOS/Turn1/A2$ java Pass2
aditya@aditya-1011PX:~/Desktop/aditya_SPOS/Turn1/A
2$
cat Pass2.txt
```

intermediate code -
(AD,01)(C,200)
(IS,054)(1)(SL,1)
(IS,04)(1)(S,1)
(IS,04)(3)(S,3)
(IS,01)(3)(L,2)
(IS,07)(6)(S,4)
(DL,01)(C,5)
(DL,01)(C,1)
(IS,02)(1)(L,3)
(IS,07)(1)(S,5)
(IS,00)
(AD,03)(S,2)+2
(IS,03)(3)(S,3)
(AD,03)(S,6)+1
(DL,02)(C,1)
(DL,02)(C,1)
(AD,02)
(DL,01)(C,1)

Symbol Table --

A	211	1
LOOP	202	1
B	212	1
NEXT	208	1
BACK	202	1
LAST	210	1

literal table --

5	206
1	207
1	213

machine code --

+ 04 1 206
+ 05 1 211
+ 04 1 211
+ 04 3 212
+ 01 3 207
+ 07 6 208
+ 00 0 005
+ 00 0 001

```
+ 02 1 213
+ 07 1 202
+ 00 0 000
+ 03 3 212    */
```

Assignment No :- 3

Title : Design Suitable data Structures and Implement pass - I of a two-pass macro-processor using OOP features in Java.

Objectives :
- To Identify and create the data Structures required in the design of macro processor.
- To Learn parameter processing in macro.
- To implement pass I of macroprocessor

Problem Statement :
Write a program to Create pass-I Macro-processor.

Outcomes :
After Completion of this assignment Students will be able to :
- Understand the Programming language of Java
- Understand the Concept of Pass-I Macro-processor

Software and Hardware Requirements :
- Software : Linux OS, JDK 1.7
- Hardware : 4GB Ram, 500 GB HDD

Theory :

MACRO

macro allows a Sequence of Source language Code to be defined once & then referred to by name each time it is to be referred. Each time this name occures in a program the Sequence of Codes is Substituted at that point.

A macro Consist of
1. Name of the macro
2. Set of parameters
3. Body of macro

Macros are typically defined at the Start of program. Macro definition Consists of :-
1. MACRO pseudo
2. MACRO name
3. Sequence of Statements
4. MEND pseudo opcode terminating

A macro is Called by writing the macro name with actual parameter is an assembly program The macro call has following Syntax <macro name>

Macro processor

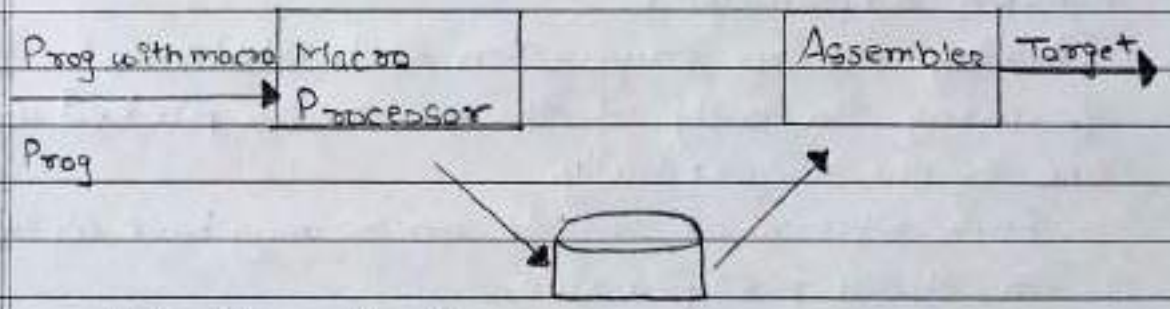


Fig. Assembly language program without macro.

Macro processor takes a Source program Containing macro defination & macro calls and translates into an assembly language program without any macro definition or calls. This program can now be handled over to a Conventional assembler to obtain the target language.

Macro Definition :-

Macros are typically defined at the start of a program. A macro definition Consists of

1. MACRO pseudo Code
2. MACRO name
3. Sequence of Statement
4. MEND pseudo opcode terminating macro defingtion.

Structure of a macro

Example

```
MACRO
INCR & ARG
ADD AREG & ARG
ADD BRA , & ARG
ADD CREG, & ARG
MEND
```

MACRO Expansion :

During macro expansion each Statement forming the body of the macro as picked up one by one Sequentially.

a. Each Statement inside macro may have as it is during expansion.

b. The name of a formal parameter which is preceded by the charater & during macro expansion an ordinary starting is retained without any modification. Formal parameters are replaced by actual parameters value.

When a call is found the Call processor sets a pointer the macro defination table pointer to the Corresponding macro defination started in MDT. The initial value of MDT is obtained from MDT index.

Design of macro processors:

Pass - I

Generate Macro Name Table (MNT)

Generate Macro Defination Table (MDT)

Generate IC i.e a Copy of Source Code without macro definations.

MNT:

- Macro Name
- MDT Index

MDT:

- Macro Statement

ALA:

- Argument

Specification of Data Bases

Pass 1 data bases

1. The input macro Source desk.
2. The output macro Source desk copy for use by passes?.
3. The macro defination table (MDT) used to Store the names of defined macros.
4. Macro name Table(MDT) used to state the name of defined macros.
5. The Macro defination table Counter used to indicate the next available entry in MNT
6. The macro name table Counter Counter (MNTC) used to indicate next available entry in MNT
7. The arguments list array (ALA) used to Substitute index markers for dummy arguments before Starting a macro defination.

Conclusion: Thus we have Successfully implemented pass-I of a two-pass Macro-processor

Assignment No.
3(A)

Name: Pratik Pingale
Class: TE Roll No. 19CO056
Subject: SPOS
****
***

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

```
import java.io.*; import
java.io.FileNotFoundException; import
java.util.Scanner; class macro
{
public static void main(String args[])throws IOException
{
int MDTC=1;
int MNTC=1;

int
index=1;

int
macroindex=
0;
String arg[]=new
String[10]; String
mname[]=new String[10];
String MNT [][]=new
String[10][10]; String MDT
[][]=new String[10][10];
String output =new Scanner(new File("file1.txt")).useDelimiter("\\Z").next();
String result[]=output.split("\n"); String
result1[]=output.split("[,\\s\\?]"); for(int
k=0;k<result1.length;k++)
{
if(result1[k].equals("MACRO")||result1[k].equals("macro"))
{
mname[macroindex]=result1[k+2]; macroindex++;

}

System.out.println("\nMACRO NAME TABLE\n————————————————————");
System.out.println("VALUE OF MDTC\tMNTC\tNAME"); for(int
```

```
k=0;k<macroindex;k++)
{
System.out.println("\t"+MDTC+"\t"+MNTC+"\t"+mname[k]);
MNTC=MNTC+1;
}
System.out.println("\n\nMACRO DEF TABLE\n————————————————————————");
System.out.println("INDEX\tCARD");
```

```
for(int i=1;i<result.length;i++)
{
System.out.println(MDTC+"\t"+result[i]);
MDTC=MDTC+1;
}
System.out.print("\n\nARGUMENT LIST ARRAY\n————————————————"); for(int
k=3;k<result1.length;k++)
{
if(result1[k].equals(mname[0]))
{
arg[0]=result1[k+1];
arg[1]=result1[k+2];
arg[2]=result1[k+3];
}
}
System.out.println("\nINDEX\t ARGUMENTS");
System.out.println("\n"+index+"\t"+arg[0]+"\n"+(index+1)+"\t"+arg[1]+"\n"+(index+2)+"\t"+arg[
2]+"\n ");
System.out.print("\n\nOUTPUT PROGRAM AFTER CALL\n————————————————"); for(int
k=6;k<result1.length;k++)
{
for(int i=3;i<6;i++)
{
if(result1[k].equals(result1[i]))
{
result1[k]=arg[i-3];
}}}
for(int k=6;k<result1.length;k++)
{
if(result1[k].equals("MEND"))
{

System.out.print("END"); break;

}
if(result1[k].equals(""))
System.out.println(); else
System.out.print(result1[k]+" ");
}}}
```

File1.txt
MACRO
ADDITION
&arg1,&arg2,&arg3
MOV ax,&arg1
ADD
ax,&arg2
ADD
ax,&arg3
MEND
ADDITION 34,45,44 END
/*

OUTPUT:

MACRO NAME TABLE
————————————
———— VALUE OF
MDTC MNTC NAME 1
1 ADDITION

MACRO DEF TABLE
————————————————
INDEX CARD
1 ADDITION &arg1,&arg2,&arg3
2 MOV ax,&arg1
3 ADD ax,&arg2
4 ADD ax,&arg3
5 MEND
6 ADDITION 34,45,44
7 END

ARGUMENT LIST ARRAY
————————————
INDEX ARGUMENTS

1 34
2 45
34
OUTPUT PROGRAM AFTER CALL
————————————
MOV ax 34
ADD ax 45

```
ADD ax 44
END */
```

Assignment No :- 4

Title : Write a Java program for pass-II of a two-pass macro-processor. The output of assignment 3 should be input for this assignment.

Objectives :

- To identify and create the data structures require in the design of macro processor.
- To learn parameter processing in macro
- To implement pass II of macroprocessor

Problem Statement :

Write a program to create pass-II macro-processor

Outcomes :

After Completion of this assignment Students will be able to :

- Understand the Programming language of Java
- Understand the Concept of Pass-II macro-processor.

Software and Hardware Requirement :-

- Software : Linux OS, JDK 1.7
- Hardware : 4GB Ram, 500 Gb HDD

Theory :

Pass II :

Replace every occurrence of macro call with macro definition.

There are four basic tasks that any macro ~~call~~ instruction process must perform:

1. Recognize macro definition:
A macro instruction processor must recognize macro definition identified by the MACRO and MEND pseudo-ops. This tasks can be Complicated when macro defination appears within macros. When MACROS and MENDs are nested, as the macro processor must recognize the nesting and Correctly match the last or or outer MEND with first MACRO. All intervening text, including nested MACROS and MENDS defines a Single macro instruction.

2. Save the definitions:
The processor must Store the macro instruction definition, which it will need for expanding macro calls.

3. Recognize calls:
The processor must recognize the macro calls that appear as operation mnemonics. This Suggests that macro names be handled as a type of op-code.

4. Expand calls and Substitute arguments:
The processor must Substitute for dummy

or macro defination arguments the corresponding arguments from a macro Call, the resulting Symbolic text Substitute for macro call. This text may Contain additional macro definition or Call.

Implementation of a 2 pass algorithm

1. We assume that our macro processor is functionally independent of the assembler and that the output text from the macro processor will be fed into the assembler.
2. The macro processor will make two independent Scans or passes over the input text, Searching first for macro definations and then for macro calls.
3. The macro processor cannot expand a macro call before having found and Saved the Corresponding macro definitions.
4. Thus we need two passes over the input text one to handle macro definitions and other to handle macro calls.
5. The first pass examines every operation code, will Save all macro definitions in a macro Definition Table and Save a Copy of the input text, minus macro definitions on the Secondary Storage.
6. The first pass also prepares a Macro Name Table along with macro Definition Table as Seen in the previous assignment that Successfully implemented pass-I of macro pre-processor.

The Second pass will now examine every operation mnemonic and replace each macro name with the appropriate text from the macro definitions.

Specification of Database :

Pass 2 database :

1. The copy of the input Source deck obtained from Pass-I
2. The output expanded Source deck to be used as input to the assembler
3. The macro Definition Table (MDT), created by pass I
4. The macro Name Table (MNT), created by pass I
5. The macro Definition Table Counter (MNTC), used to indicate the next line of text to be used during macro expansion
6. The Argument List Array (ALA), used to Substitute macro call arguments for the index markers in Stored macro definition

Conclusion :

Thus, we have Successfully implemented pass-II of a two-pass macro-processor.

Assignment No. 4(A)

Name: Pratik Pingale
Class: TE Roll No. 19CO056
Subject: SPOS
****

*** **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.

```
import java.io.*;
import
java.util.HashMap;
import
java.util.Vector;

public class macroPass2 { public static void
        main(String[] Args) throws IOException{
                BufferedReader b1 = new BufferedReader(new
                FileReader("intermediate.txt")); BufferedReader b2 = new
                BufferedReader(new FileReader("mnt.txt")); BufferedReader b3 =
                new BufferedReader(new FileReader("mdt.txt")); BufferedReader b4
                = new BufferedReader(new FileReader("kpdt.txt")); FileWriter f1 =
                new FileWriter("Pass2.txt");
                HashMap<Integer,String> aptab=new HashMap<Integer,String>();
                HashMap<String,Integer> aptabInverse=new HashMap<String,Integer>();
                HashMap<String,Integer> mdtpHash=new HashMap<String,Integer>();
                HashMap<String,Integer> kpdtpHash=new HashMap<String,Integer>();
                HashMap<String,Integer> kpHash=new HashMap<String,Integer>();
                HashMap<String,Integer> macroNameHash=new HashMap<String,Integer>();
                Vector<String>mdt=new Vector<String>();
                Vector<String>kpdt=new
                Vector<String>(); String s,s1; int
                i,pp,kp,kpdtp,mdtp,paramNo;
                while((s=b3.readLine())!=null)
                mdt.addElement(s);
                while((s=b4.readLine())!=null)

                        kpdt.addElement(s);
                while((s=b2.readLine())!=null){ String
                        word[]=s.split("\t"); s1=word[0]+word[1];
                        macroNameHash.put(word[0],1);
                        kpHash.put(s1,Integer.parseInt(word[2]
                        ));
                        mdtpHash.put(s1,Integer.parseInt(wor
```

```
            d[3]));
            kpdtpHash.put(s1,Integer.parseInt(wor
            d[4]));
    }
```

```
while((s=b1.readLine())!=null){ String b1Split[]=s.split("\\s");
    if(macroNameHash.containsKey(b1Split[0])){ pp=
    b1Split[1].split(",").length-b1Split[1].split("=").length+
    1; kp=kpHash.get(b1Split[0]+Integer.toString(pp));
    mdtp=mdtpHash.get(b1Split[0]+Integer.toString(pp));
    kpdtp=kpdtpHash.get(b1Split[0]+Integer.toString(pp))
    ; String actualParams[]=b1Split[1].split(",");
    paramNo=1;
        for(int          j=0;j<pp;j++){
            aptab.put(paramNo,
            actualParams[paramNo-1]);
            aptabInverse.put(actualParams[paramNo-1],pa
            ramNo); paramNo++;
        }
        i=kpdtp-1;
        for(int j=0;j<kp;j++){
            String temp[]=kpdt.get(i).split("\t");
            aptab.put(paramNo,temp[1]);
            aptabInverse.put(temp[0],paramN
            o); i++;
                paramNo++;
        }
        i=pp+1;
         while(i<=actualParams.length){
            String initializedParams[]=actualParams[i-1].split("=");

    aptab.put(aptabInverse.get(initializedParams[0].substring(1,initializedParams[0].lengt
h())),initial izedParams[1].substring(0,initializedParams[1].length())); i++;
        }
        i=mdtp-1; while(mdt.get(i).compareToIgnoreCase("MEND")!=0){
        f1.write("+ ");




mdt.get(i).charAt(++j)));                    else

    f1.write(mdt.get(i).charAt(j));
            }

                                    f1.write("\n"); i++;
                                }
    aptab.clear();

            }    aptabInverse.clear();
```

else

```
                }       f1.write("+
                "+s+"\n"); b1.close();
                b2.close();
                b3.close();
                b4.close();
                f1.close();
        }
}
/*
```

OUTPUT:

```
aditya@aditya-1011PX:~/Desktop/aditya_SPOS/Turn1/A4$ javac macroPass2.java
aditya@aditya- 1011PX:~/Desktop/aditya_SPOS/Turn1/A4$ java macroPass2
aditya@aditya- 1011PX:~/Desktop/aditya_SPOS/Turn1/A4$ cat Pass2.txt

Intermediate -
- M1
10,20,&b=CR
EG
M2 100,200,&u=&AREG,&v=&BREG

Kpdt—

a    AREG  b
        -       u
CREG
v       DREG

pass2— +
MOVE
AREG,10
+ ADD AREG,='1'
+ MOVER AREG,20
+ ADD AREG,='5'
+ MOVER &AREG,100 +
MOVER &BREG,200
+ ADD &AREG,='15'
+ ADD &BREG,='10'



MNT—
M1    2         2         1         1
M2    2         2         6         3


MDT --
```

MOVE
#3,#1

```
ADD #3,='1'
MOVER #3,#2
ADD
#3,='5'
MEND
MOVER
#3,#1
 MOVER #4,#2
  ADD #3,='15'
  ADD #4,='10'
MEND
*/
```

Assignment No :- 1

Title : Write a Java program to implement following Scheduling algorithms : FCFS, SJF, Priority and Round Robin

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 to implement following Scheduling algorithms : FCFS, SJF, Priority and Round Robin.

Outcomes :
- Knowledge Scheduling policies
- Compare difference Scheduling algorithms

Software Requirement and Hardware Requirement :
- Software : JDK / Eclipse
- Hardware : Dell i3, 6th Gen, 4GB Ram, 500Gb HDD

Theory :

CPU Scheduling :
- CPU Scheduling refer to a Set of policies and mechanisms built into the operating System

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.

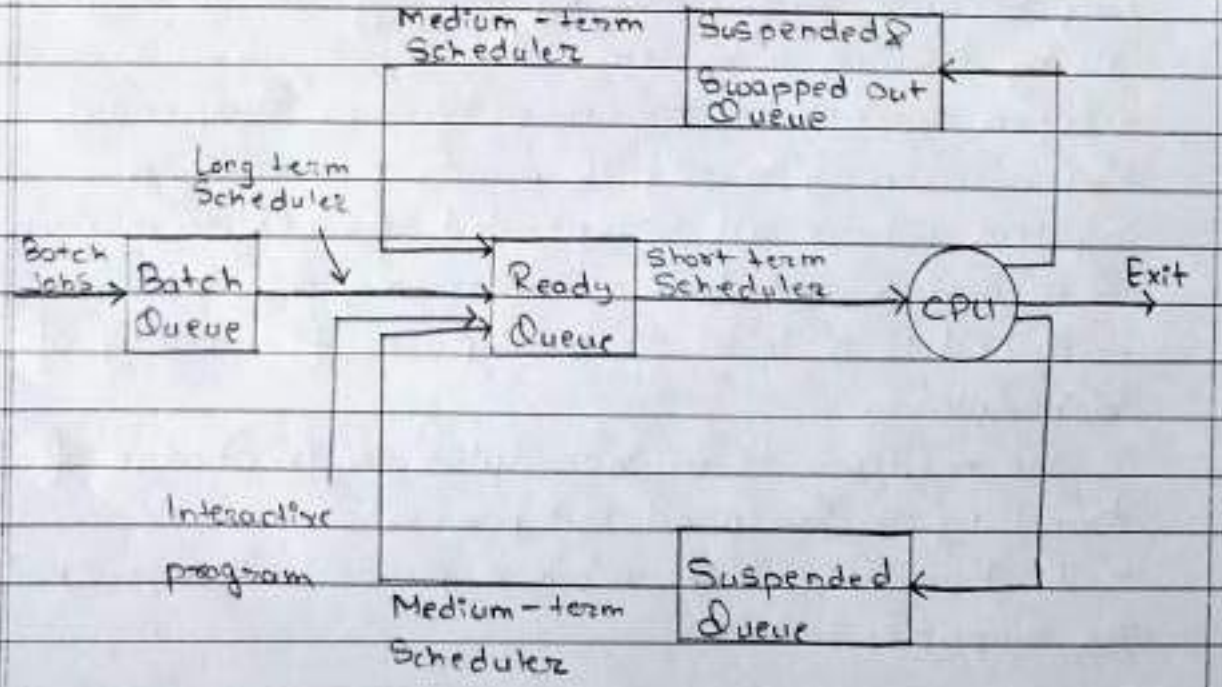
Necessity of Scheduling

- Scheduling is required when no. of jobs are to be performed by CPU.
- Scheduling provides meachanism 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.

Type of Schedulers

In general, there are three different type 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 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 resources 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 System where all Suspended processes remains & 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 far 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 event occurs, the OS involves the short term scheduler.

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

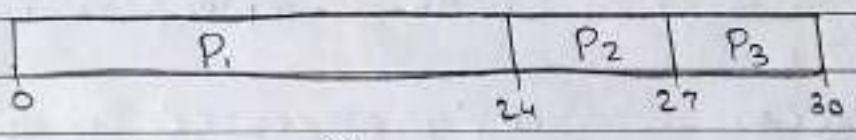
First Come First Serve Algorithm
- FCFS is working on the Simplest Scheduling discipline
- The workload is Simply processed in an order of their arrival, with no pre-emption.
- FCFS Scheduling may result into poor performance. Since there is no discrimination on the basics of required Services, short jobs may Considerable in
- turn around delay and waiting time.
Advantages
- Better for long processes
- Simple method
- 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.

First Come, First Served :-

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arive in the order: P_1, P_2, P_3
- The Gannt Chart for the Schedule is:



- waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0+24+27)/3 = 17$

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 interactive System where required CPU time is known in advance.

Advantages :-
- It gives Superior turnaround time performance to Shortest process next because a short job is given immediate preferance to a running longer job.
- Throughput is high.

Disadvantages :-
- Elapsed time must be recorded, it results an additional over head on the processor.
- Starvation may be possible for the longer processes.

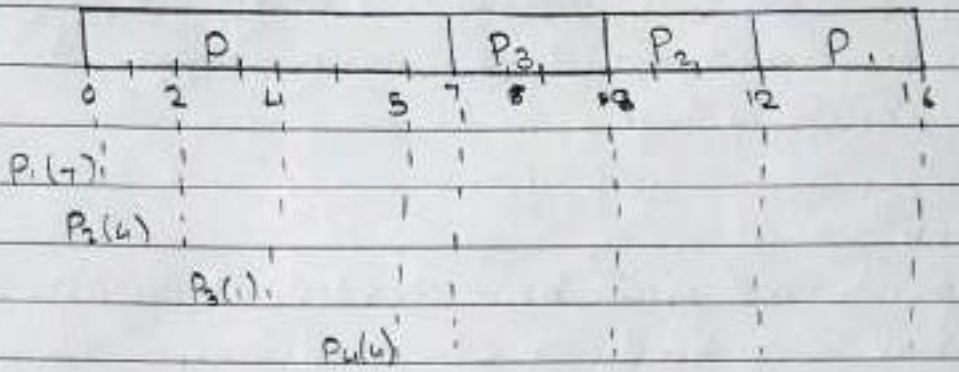
This algorithm is divided into two types:
- Pre-emptive SJF
- Non-pre-emptive SJF

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.

Shortest job first Scheduling

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- No preemptive SJF Average waiting Time = (0+6+3+7)/4 = 4



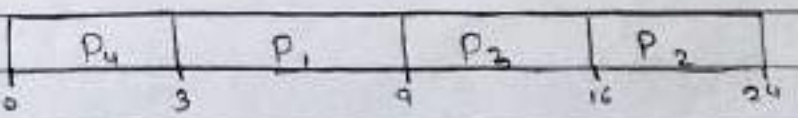
P1's waiting time = 0
P2's waiting time = 6
P3's waiting time = 3
P4's waiting time = 7

Non-pre-emptive SJF Algorithm:
In this algorithm, job having less burst less is Selected 1st for execution. It is executed for its total burst time and then the next job having least burst time is Selected.

Example of SJF :-

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Scheduling chart

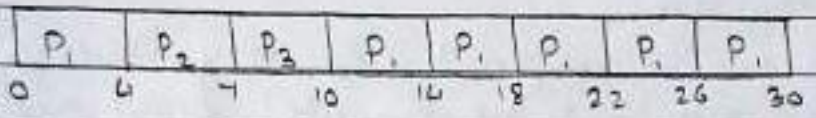


- Average waiting time = (3+16+9+0)/4 = 7

Round Robin Scheduling:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Quantum time = 4 milliseconds
- The Gantt chart is:



- Average waiting time = {[0+(10-4)]+4+7}/3 = 5.6
- 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.

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.

Priority Scheduling:

- Priority Scheduling is a non-preemptive algorithm and one of the most common Scheduling algorithm in batch Systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and soon.
- Processes with same priority are executed on first come first served basis.

Advantage:

- Good response for the highest priority processes.

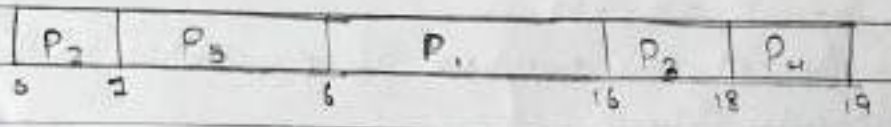
Disadvantages:

- Starvation may be possible for the lowest priority processes.

Priority:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Grantt chart



- Average waiting time = (6+0+16+18+1)/5 = 8.2

Algorithms: FCFS

Step 1: Start the process
Step 2: Accept the number of process 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
(a) Waiting time for process (n) = waiting time of process (n-1) + Burst time of process (n-1)
(b) Turn around time for Process (n) = waiting time of Process (n) + Burst time for process (n)
Step 6: Calculate
(a) Average waiting time = Total waiting Time / Number of process
(b) Average Turnaround time = Total Turn-around Time / Number of process
Step 7: Stop the process.

SJF:

Step 1: Start 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 Burn time by Sorting according to lowest to higher 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

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

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

Step 7: Calculation

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

(d) Average Turnaround time = total turnaround time / number of process

Step 8: Stop the process

RR:

Step 1: Start the process

Step 2: Accept the number of process 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

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

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

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

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

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

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

Step 7: Stop the process.

Priority Scheduling:

Step 1: Start the process

Step 2: Accept the number of process 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 ® according the priority by Sorting according to lowest to highest burst time and process.

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

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

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

(f) Turn around time for Process (n) = waiting time for process (n) & Step 6: Calculate

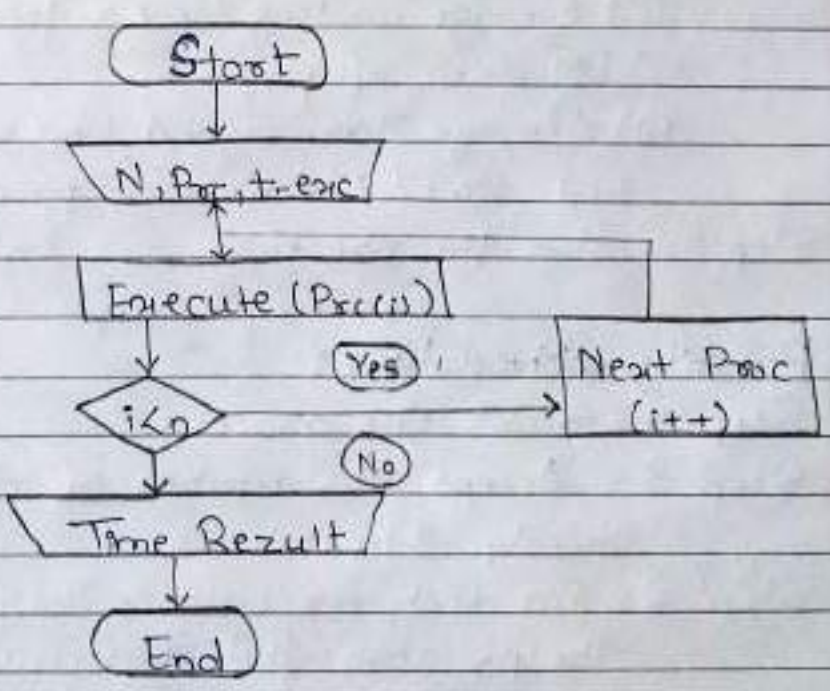
(g) Average waiting time = Total waiting Time / Number of process

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

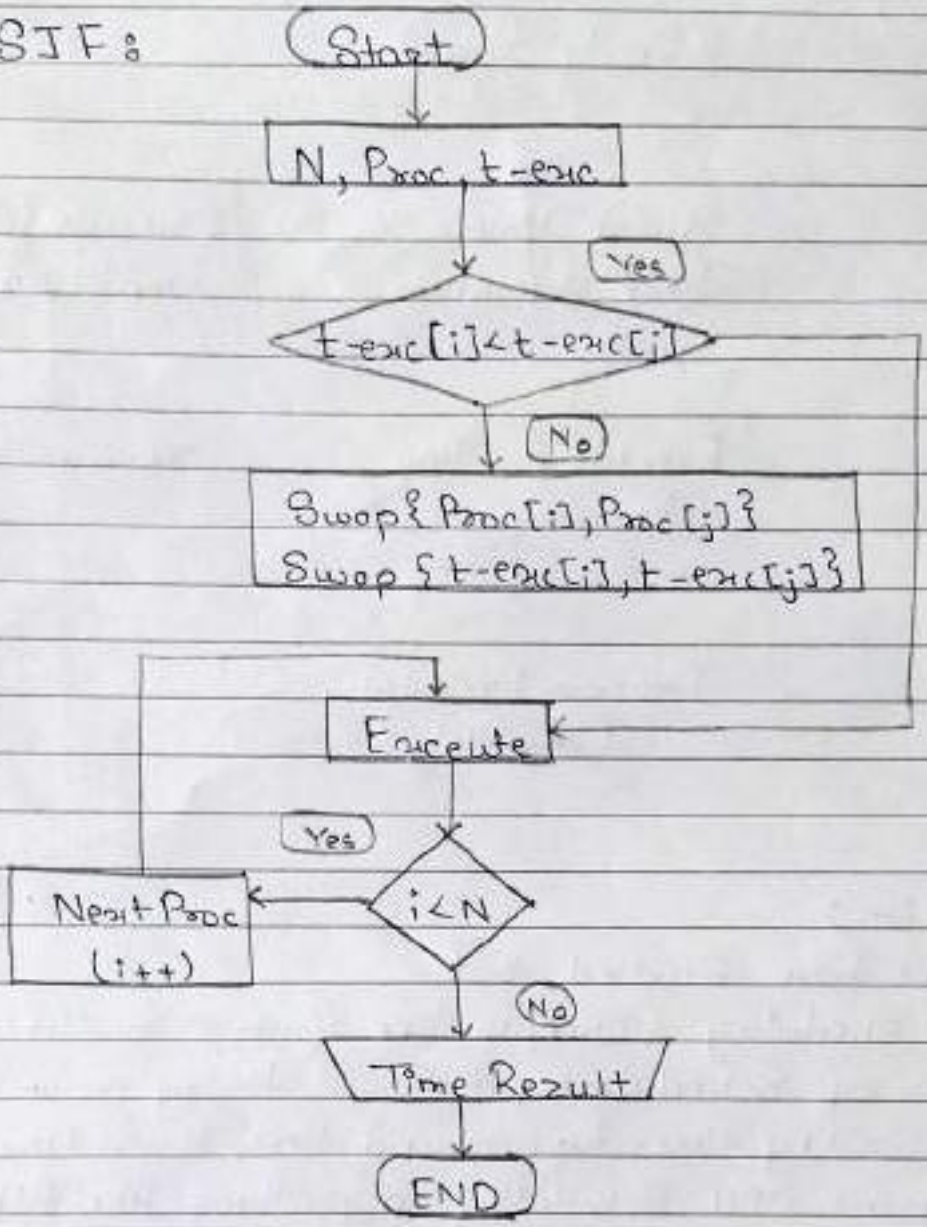
Step 7: Stop the process

Flowchart:

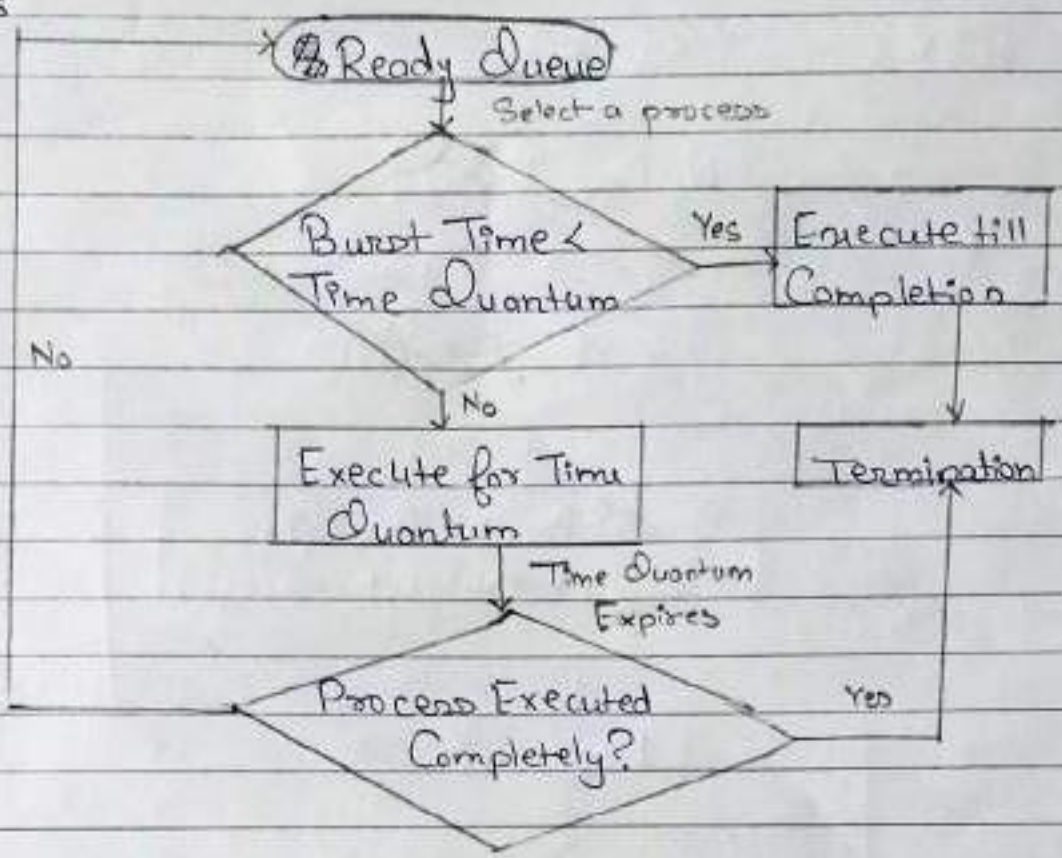
FCFS:



SJF:


Start
N, Proc, t-exec
Yes
t-exc[i]<t-exc[j]
No
Swap{Proc[i], Proc[j]}
Swap{t-exc[i], t-exc[j]}
Execute
Yes
i<N
Next Proc
(i++)
No
Time Rezult
END

RR:



Conclusion :

Hence we have Studied that -

- CPU Sheduling 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.

Assignment No. 1(C)

Name: Pratik Pingale
Class: TE Roll No. 19CO056
Subject: SPOS

****

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

FCFS

```
import java.io.*;
import
java.util.Scanner;
public class FCFS
{
    public static void main(String args[])
    {
        int
        i,no_p,burst_time[],TT[],WT[];
        float avg_wait=0,avg_TT=0;
        burst_time=new int[50];
        TT=new int[50];
        WT=new int[50]; WT[0]=0;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the number of
        process: "); no_p=s.nextInt();
        System.out.println("\nEnter Burst Time for processes:"); for(i=0;i<no_p;i++)
        {
            System.out.print("\tP"+(i+1)+": "); burst_time[i]=s.nextInt();
        }

        for(i=1;i<no_p;i++)
        {
            WT[i]=WT[i-1]+burst_time[i-1]; avg_wait+=WT[i];
        }
        avg_wait/=no_p;

        for(i=0;i<no_p;i++)
        {
            TT[i]=WT[i]+burst_time[i]; avg_TT+=TT[i];
        }
        avg_TT/=no_p;
```

```
System.out.println("\n************************************************************
****");
        System.out.println("\tProcesses:");


                                                        System.out.println("*******************************
                                                        ******************************
**");

                                                                System.out.println("
                                                                Process\tBurst Time\tWaiting
                                                                Time\tTurn Around Time");
                                                                for(i=0;i<no_p;i++)
                                                                {
                                        System.out.println("\tP"+(i+1)+"\t "+burst_time[i]+"\t\t "+WT[i]+"\t\t
                                        "+TT[i]);


                        }
                        System.out.println("\n_____________________________________________
                                                                                        ");
                        System.out.println("\nAverage waiting time : "+avg_wait);
                        System.out.println("\nAverage Turn Around time : "+avg_TT+"\n");
                }
}
/*Output:
Enter the number of process: 3

Enter Burst Time for
        processes: P1:
        24
        P2: 3
        P3: 3
************************************************************


        Processes:


************************************************************

  Process   Burst Time   Waiting Time   Turn Around Time
      P1      24    0       24
      P2      3     24      27
      P3      3     27      30


  ----------------------------------------
```

Average waiting time : 17.0
Average Turn Around time :
27.0 */

Round Robin(Preemptive)

```
import java.util.*;
import java.io.*; class
RoundR
{
        public static void main(String args[])
        {
                int Process[]=new int[10];
                int a[]=new int[10]; int
                Arrival_time[]=new int[10];
                int Burst_time[]=new
                int[10]; int WT[]=new
                int[10]; int TAT[]=new
                int[10]; int Pno,sum=0;;
                int TimeQuantum;

System.out.println("\nEnter the no. of Process::");
                Scanner sc=new
                Scanner(System.in);
                Pno=sc.nextInt();
                System.out.println("\nEnter each process::");
                for(int i=0;i<Pno;i++)
                {
                                Process[i]=sc.nextInt();
              }

System.out.println("\nEnter the Burst Time of each process::"); for(int
                i=0;i<Pno;i++)
                {
                        Burst_time[i]=sc.nextInt();
                }

System.out.println("\nEnter the Time
Quantum::"); TimeQuantum=sc.nextInt(); do{
                for(int i=0;i<Pno;i++)
                {
                        if(Burst_time[i]>TimeQuantum)
                        {
                                Burst_time[i]-=TimeQuantum; for(int
                                j=0;j<Pno;j++)
                                                {
        e
        l
        s
```

e

```
{                                                    } tum;

                }
                    for(int j=0;j<Pno;j++)

                    {
                        if((j!=i)&&(Burst_
                        time[j]!=0))
                        WT[j]+=Burst_ti
                        me[i];
                    }
                    Burst_time[i]=0;
                }
            }
                sum=0;
                for(int
                k=0;k<Pno;k
                ++)
                sum=sum+Burst_time[k];
        } while(sum!=0);

            for(int i=0;i<Pno;i++)
                    TAT[i]=WT[i]+a[i];
            System.out.println("process\t\tBT\tWT\tTAT"); for(int
            i=0;i<Pno;i++)
            {
               System.out.println("process"+(i+1)+"\t"+a[i]+"\t"+WT[i]+"\t"+TAT[i]);
            }
                float avg_wt=0;
            float avg_tat=0;
            for(int
            j=0;j<Pno;j++)
            {
                 avg_wt+=WT[j];
            }

            for(int j=0;j<Pno;j++)
            {
                 avg_tat+=TAT[j];
            }
              System.out.println("average waiting time "+(avg_wt/Pno)+"\n Average turn
              around
time"+(avg_tat/Pno));
        }
}
```

OUTPUT:

```
Enter the no. of Process::
5
Enter each process::
1
2
3
4
5

Enter the Burst Time of each process::
2
1
8
4
5
Enter the Time Quantum::
2
process BT WT TAT process1 0 0 0
process2 0 2 2 process3  0 12 12
process4 0 9 9 process5 0 13 13
average waiting time 7.2
Average turn around
time7.2 */
```

SJF(Non-Preemptive)

```
import java.util.Scanner; class
SJF1{ public static void
main(String args[]){
int burst_time[],process[],waiting_time[],tat[],i,j,n,total=0,pos,temp; float
wait_avg,TAT_avg;
Scanner s = new Scanner(System.in);

System.out.print("Enter number of process: "); n
= s.nextInt();
```

```
process = new int[n]; burst_time
= new int[n];
waiting_time = new
int[n]; tat = new int[n];

System.out.println("\nEnter Burst time:"); for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)
+"]: "); burst_time[i] = s.nextInt();;
process[i]=i+1;
//Process Number
}

//Sorting for(i=0;i<n;i++)
{
pos=i; for(j=i+1;j<n;j++)


{
if(burst_time[j]<burst_time[pos]) pos=j;
}

temp=burst_time[i];
burst_time[i]=burst_time[pos];
burst_time[pos]=temp;

temp=process[i]; process[i]=process[pos];
process[pos]=temp;
}
//First process has 0 waiting
time waiting_time[0]=0;
//calculate waiting time
for(i=1;i<n;i++)
{
waiting_time[i]=0;
for(j=0;j<i;j++)
waiting_time[i]+=burst_time[j
]; total+=waiting_time[i];
}

//Calculating Average
waiting time
wait_avg=(float)total/n;
total=0;

System.out.println("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)
{
```

```
tat[i]=burst_time[i]+waiting_time[i]; //Calculating Turnaround Time total+=tat[i];
System.out.println("\n p"+process[i]+"\t\t "+burst_time[i]+"\t\t "+waiting_time[i]+"\t\t "+tat[i]);
}

//Calculation of Average Turnaround Time
```

```
TAT_avg=(float)total/n;
System.out.println("\n\nAverage Waiting Time: "+wait_avg);
System.out.println("\nAverage Turnaround Time:
"+TAT_avg);

}
}
```

SJF(Preemptive)

```
import
java.util.Scanner;
class sjf_swap1{
public static void main(String args[])

{ int
burst_time[],process[],waiting_time[],tat[],arr_time[],completion_time[],i,j,n,total=0,total_comp=
0,pos, temp;
float wait_avg,TAT_avg;
Scanner s = new Scanner(System.in);
System.out.print("Enter number of
process: "); n
= s.nextInt();
process = new int[n];
burst_time = new
int[n]; waiting_time =
new int[n];
arr_time=new int[n];
tat = new int[n];
completion_time=new int[n];




//burst time
System.out.println("\nEnter Burst time:"); for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]: "); burst_time[i]
= s.nextInt();;
```

```
process[i]=i+1; //Process Number
}

//arrival time
System.out.println("\nEnter arrival time:"); for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)
+"]: "); arr_time[i] = s.nextInt();;
process[i]=i+1;
//Process Number
}

//Sorting for(i=0;i<n;i++)
{
pos=i; for(j=i+1;j<n;j++)
{
if(burst_time[j]<burst_time[pos]) pos=j;
}

temp=burst_time[i];
burst_time[i]=burst_time[pos];
burst_time[pos]=temp;

temp=process[i]; process[i]=process[pos];
process[pos]=temp;

System.out.println("process"+process[i]);
}
//completion time
new
for(i=1;i<n;i++)




{
completion_time[i]=0; for(j=0;j<i;j++)
completion_time[i]+=burst_time[j];
total_comp+=completion_time[i];
}
//First process has 0
waiting time
waiting_time[0]=0;
//calculate

waiting time for(i=1;i<n;i++)
{
```

```
  waiting_time[i]=0;
  for(j=0;j<i;j++)
  waiting_time[i]+=burst_time[j
  ]; total+=waiting_time[i];
  }

//Calculating Average waiting time
 wait_avg=(float)total/n;
  total=0;

  System.out.println("\nPro_number\t Burst Time \tcompletion_time\tWaiting Time\tTurnaround
  Time"); for(i=0;i<n;i++)
  {
  tat[i]=burst_time[i]+waiting_time[i];
   //Calculating Turnaround Time total+=tat[i];
  System.out.println("\n"+process[i]+"\t\t
  "+burst_time[i]+"\t\t
  "+completion_time[i]+"\t\t"+waiting_time[i]+"\t\t
  "+tat[i]);
  }

  //Calculation of Average Turnaround
  Time TAT_avg=(float)total/n;
  System.out.println("\n\nAWT:
  "+wait_avg); System.out.println("\nATAT:
  "+TAT_avg);
```

Assignment No :- 2

Title : Write a Java program to implement Banker's Algorithm

Objectives :
- To understand Safe and unsafe State of a System
- To understand deadlock
- To Implementation of banker's algorithm for deadlock detection and avoidance

Problem Statement :
Write a Java program to implement Banker's Algo

Outcomes :
- Knowledge Bankers Algorithms.
- Application of Bankers Algorithm.

Software and Hardware Requirements :
- Software : JDK / Eclipse
- Hardware : Dell i3, 6th Gen, 4GbRam 500GB HDD

Theory :
The Banker's algorithm, Sometimes referred to as the detection algorithm, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that test for safety by Simulating the allocation of predetermined maximum possible amounts of all resources,

and then makes an "S-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking system to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has enough money even if all the n account holders draw all their money at once Banker's algorithm works in a similar way in computers, whenever a new process is created, it must exactly specify the maximum instances of each resource type that it needs.

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

- Available: It is an array of length m. It represents the number of available resources of each type. If Available[j] = K, then are K instances available of resource type Rj.
- Max: It is an n x m matrix which represent the maximum number of instances of each resources that process can request. If Max[i][j] = K, then the process Pi can request atmost K instance of resource type Rj.
- Allocation: It is an n x m matrix which repre-
sent

the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P_i is currently allocated k instances of resource type R_j.

- Need: It is an n x m matrix which indicates the remaining resources needs of each process. If Need[i][j] = k, then process P_i may need k more instances of resource type R_j to complete its task.

Need[i][j] = Max[i][j] - Allocation[i][j]

Resource Request Algorithm:

This describes the behavior of the System when a process makes a resource request in the form of a request matrix. The steps are:

1. If number of requested instances of each resource is less than the need, go to step 2.
2. If number of requested instances of each resource type is less than the available resources of each type, go to step 3. If not, the process has to wait because sufficient resources are not available yet.
3. Now, assume that the resources have been allocated. Accordingly do,

Request$_i$ Allocation$_i$ = Allocation$_i$ + Request$_i$

Need$_i$ = Need$_i$ - Request$_i$

This step is done because the System needs to assume that resources have been allocated.

So there will be less resources available after allocation. The number of allocation instance will increase. The need of the resources by the process will reduce. That's what is represented by the above three operations.
After Completing the above three steps, check if the System is in Safe State by applying the Safety algorithm. If it is in Safe state, proceed to allocation the requested resources, Else, the process has to wait longer.

Safety Algorithm:
1. Let Work and finish be vectors of length m and n respectively, Initially,
2. Work = Available
3. Finish[i] = false for i = 0, 1, ..., n-1.
This means, initially, no process has finished and the number of available resources is represented by the Available array.
4. Find an index i such that both
5. Finish [i] == false
6. Need i <= Work
if there is no such i present, then proceed to Step 4 It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4
7. Perform the following:
8. Work = Work + Allocation
9. Finish[i] = true;
10. Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other process.

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

That means if all processes are finished, then the System is in safe state.

Advantages: Avoid deadlock and it is less restrictive than deadlock prevention.

Disadvantages: Only works with fixed number of resources and processes.
- Guarantees finite time - not reasonable response time
- Needs advantage knowledge of maximum needs
- Not suitable for multi-access systems
- Unnecessary delays in avoiding unsafe states which may not lead to deadlock.

Limitation:
Like the other algorithms, the Banker's algorithm has some limitations when implemented. Specifically, it needs to know how much of each resources a process could possibly request. In most system, this information is unavailable making it impossible to implement the Banker's algorithm. Also it is unrealistic to assume

that the number of process is static since in most systems the number of processes varies dynamically. Moreover, the requirement that a process will eventually release all its resources is sufficient for the correctness of the algorithm however it is not sufficient for a practical system. Waiting for hours for resources to be released is usually not acceptable.

Conclusion:

Thus, I have implemented how resource allocation is done with the bankers algorithm to avoid the deadlocks.

Assignment No. 2(C)

Name: Pratik Pingale

Class: TE **Roll No. 19CO056**

Subject: SPOS

****

Problem Statement : Write a Java program to implement Banker's Algorithm

Java:

```
import java.util.Scanner;
public class BankersImplementation {
   private int available[];
   private int maximum[][], allocation[][],
   need[][]; private boolean isCompleted[];
   private int sequence[];
   private int noOfresources, noOfprocesses;
   private static final Scanner scanner = new Scanner(System.in);

   BankersImplementation(int noOfprocesses, int noOfresources, int maximum[][], int
allocation[][], int available[]) {
      sequence = new int[noOfprocesses];
      isCompleted = new boolean[noOfprocesses];
      need = new
      int[noOfprocesses][noOfresources];
      this.maximum = maximum;
      this.allocation = allocation;
      this.noOfprocesses =
      noOfprocesses; this.noOfresources
      = noOfresources; this.available =
      available;
   }

   public void calculateNeedMatrix() {
      for(int i = 0 ; i < noOfprocesses;
```

```
i++) {
   for(int j = 0; j < noOfresources; j++) {
```

```
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}

public void displayNeedMatrix() {
    System.out.println("Resultant Need Matrix");
    for(int i = 0; i < noOfprocesses ; i++) {
        System.out.print("P"+i+"\t");
        for(int j = 0; j < noOfresources ;
            j++) {
            System.out.print(need[i][j]+" ");
        }
        System.out.println();
    }
}

public void calculateSafeSequence() {
    int count = 0; //for tracking how many processes completed their
    execution boolean execute = true;
    while(count < noOfprocesses) {
        boolean flag = false; //flag to break the while loop if none of the process can able to
execute (DEADLOCK)
        for(int i = 0 ; i < noOfprocesses ;
            i++) { execute = true;
            if(isCompleted[i] == false) {
                //check whether process can execute or
                not for(int j = 0 ; j < noOfresources ; j++) {
                    if(need[i][j] > available[j])
                        { execute = false;
                        break;
                    }
```

```
        }
        if(execute) {
          //execute the process
          //update available
          for(int j = 0 ; j < noOfresources ; j++) {
            available[j] = available[j] + allocation[i][j];
          }
          sequence[count++] =
          i; isCompleted[i] =
          true; flag = true;
        }
      }
    }
    if(flag ==
      false)
      break;
}
if(count < noOfprocesses) {
    System.out.println("SYSTEM IS
    UNSAFE");
}
else {
    //print the safe list
    System.out.println("SYSTEM IS
    SAFE"); System.out.print("SAFE
    SEQUENCE IS "); for(int i = 0; i <
    sequence.length; i++) {
      if(i == sequence.length-1)
        System.out.print("P"+
        sequence[i]);
      else
        System.out.print("P"+ sequence[i] + "=>");
```

```
    }
    System.out.println();
}
```

```
}

public static void main(String args[]) {
   System.out.println("Bankers Algorithm
   Implementation"); int noOfprocesses, noOfresources;
   int maximum[][], allocation[][];
   int available[];

   System.out.print("Enter No. of processes: ");
   noOfprocesses = scanner.nextInt();
   System.out.print("Enter No. of resources type:
   "); noOfresources = scanner.nextInt();

   available = new int[noOfresources];
   maximum = new int[noOfprocesses][noOfresources];
   allocation = new int[noOfprocesses][noOfresources];

   System.out.println("Available
   Resources"); for(int i = 0; i <
   noOfresources ; i++) {
      System.out.println("Enter the available count of "+ (i+1) +"
      Resource"); available[i] = scanner.nextInt();
   }

   System.out.println("Allocation Matrix");
   for(int i = 0; i < noOfprocesses; i++) {
      System.out.println("Enter the allocation matrix values for P"+i);
      for(int j = 0; j < noOfresources ; j++) {
         allocation[i][j] = scanner.nextInt();
      }
   }
```

```
    System.out.println("Maximum
    Matrix"); for(int i = 0; i <
    noOfprocesses; i++) {
      System.out.println("Enter the maximum matrix values for P"+i);
      for(int j = 0; j < noOfresources ; j++) {
        maximum[i][j] = scanner.nextInt();
      }
    }

    BankersImplementation bankersImplementation = new
BankersImplementation(noOfprocesses, noOfresources, maximum, allocation, available);
    bankersImplementation.calculateNeedMatrix();
    bankersImplementation.displayNeedMatrix();
    bankersImplementation.calculateSafeSequenc
    e();
  }
}
```

C:

```
//C program for Banker's Algorithm
#include <stdio.h>
int main()
{
        // P0, P1, P2, P3, P4 are the names of Process

        int n, r, i, j, k;
        n = 5; // Indicates the Number of
        processes r = 3; //Indicates the Number
        of resources
        int alloc[5][3] = { { 0, 0, 1 }, // P0 // This is Allocation Matrix
                                                { 3, 0, 0 }, // P1
                                                { 1, 0, 1 }, // P2
                                                { 2, 3, 2 }, // P3
```

{ 0, 0, 3 } }; // P4

```
int max[5][3] = { { 7, 6, 3 }, // P0 // MAX Matrix
                  { 3, 2, 2 }, // P1
                  { 8, 0, 2 }, // P2
                  { 2, 1, 2 }, // P3
                  { 5, 2, 3 } }; // P4

    int avail[3] = { 2, 3, 2 }; // These are Available Resources

     int f[n], ans[n], ind =
      0; for (k = 0; k < n;
          k++) { f[k] = 0;
    }
    int need[n][r];
    for (i = 0; i < n; i++) {
              for (j = 0; j < r; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
              for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                  int flag = 0;
                                  for (j = 0; j < r; j++) {
                                            if (need[i][j] >
                                                      avail[j]){ flag
                                                      = 1; break;
                                            }
                                  }

                                  if (flag == 0) {
```

```
                                        ans[ind++] = i;
                                        for (y = 0; y < r; y++)
                                                avail[y] += alloc[i][y];
                                        f[i] = 1;
                                }
                        }
                }
        }

        printf("Th SAFE Sequence is as
        follows\n"); for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);

        return (0);

}
```

Output:-

```
Th SAFE Sequence is as
 follows P1 -> P3 -> P4 ->
 P0 -> P2
```

Assignment No :- 3

Title : Implement UNIX System calls like ps, fork, join, exec family, and wait for process management

Objectives:
- To understand UNIX System call
- To understand Concept of process management
- To implement some of System call of OS

Problem Statement :
Implement UNIX System Calls like ps, fork, join, exec family, and wait for process management

Out Comes :
- Knowledge of System call
- Compare System call and System function
- Application of System Call.

Software Requirements :
- GCC or JDK/Eclipse

Hardware Requirements :
- M/C lenovo Think center M700 C13, 6100, 6th Gen. H81, 4GB RAM, 500GB HDD

Theory Concepts :

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 the resource. This is done via Something Called a System Call.
- When a program makes a System call, the mode is switched from user model 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 Changes of mode from Kernel mode back to user mode.

Generally, System Calls are made by the user level programs in the following Situations:

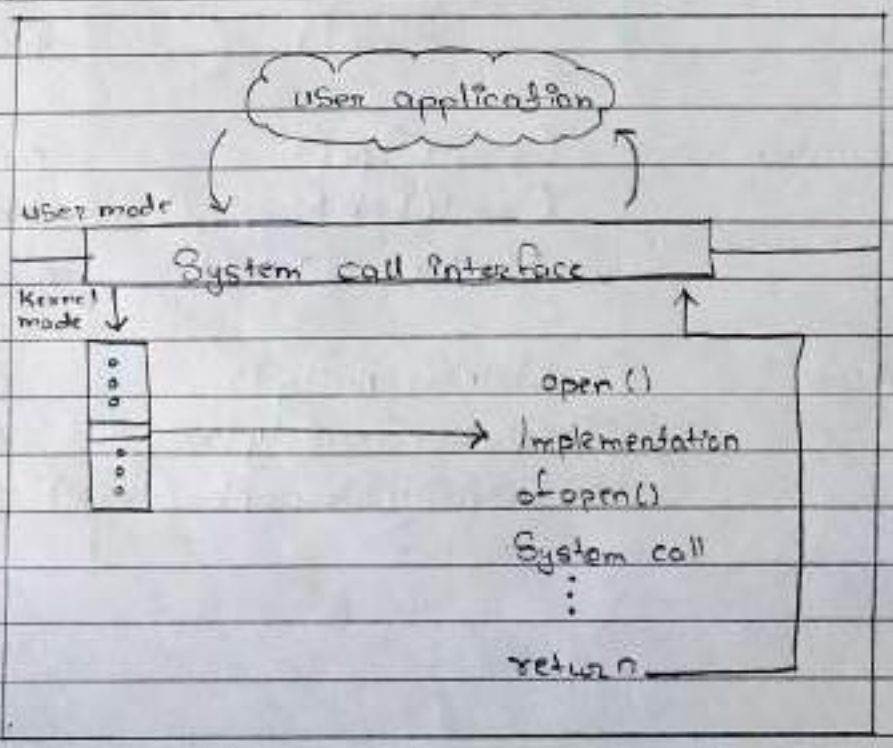
- Creating, opening, closing and deleting files in the file System.
- Creating and managing new processes.
- Creating a Connection in the network, Sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer
- To understand System calls, first one needs to understand 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.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire System will be halted.

User Mode
- When CPU is in user mode, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted
- That means that 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.



User mode and Kernel mode

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	Shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

System Call Basics:

- Since System calls are functions, we need to include the proper header files 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
  - To access errno you must declare it's
    - extern int errno;

Unix Processes

- Recall a process is a program in execution
- Processes create other processes with the fork() System call
- fork() creates an identical Copy of the parent process
- we say the parent has cloned itself to create a child
- we can tell the two process apart use the return value of fork()
  - In parent: fork() returns the PID of the new child - In child: fork
- Draw picture

```
main () {
  int id;
      id = fork();
      if (id == 0) {
         /* in child */
   } else {
      */ in parent */
     }
  }
```

Process PID = 10

```
main () {
  int id;
id = fork();
if (id == 0) {
   /* in child */
} else {
   /* in parent */
 }
}
```

Process PID 11

```
main () {
int id;
 id = fork();
 if (id == 0) {
    /* in child */
 } else {
    * in parent */
 }
}
```

Starting New Programs

- fork() only allows us to create a new process that is a duplicate of the parent
- The exec() System call is used to start a new pgm
- exec() replaces the memory image of the calling processes with the images of the new program.
- we use fork() and exec() together to start a new program.

```
main() {
int id;
    id = fork();
    if (id == 0) {
      /* in child */
     exec ("/bin/ls");
    } else {
      /* in parent */
       wait();
     }
   }
```

Process 10

```
main() {
 int id;
 id = fork();
 if (id == 0) {
 /* in child */
 exec("/bin/ls");
} else {
   /* in parent */
   wait();
  }
}
```

Process PID = 11

```
main() {
int id;
  id = fork();
  if (id == 0) {
    /* in child */
    exec("/bin/ls");
  } else {
  /* in parent */
    wait();
    }
}
```

Process PID = 11

```
main() {
/*Code for
  /bin/ls */


exit(0);
}
```

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 proces

- int execl (char *name, char *arg(), ..., char)
  - change program image of current process
  - Reset Stack and free memory.
- 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 Commands :

The Ps Command is used to provide information about the currently running processes, including their process identification numbers.

Syntax : ps [options]

- Fork() - The fork() System call is used to create processes. When a process 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.

- The process which called the fork() call is the parent process and the process which is created newly is Called the child process.

- Join Command: The joind 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.
Syntax: Join [option]...file1 file2

- Exec() - 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.

- It means, after an exec() call, only the new process exists. The process which made the System call wouldn't exist.

There are many flavors of exec() in UNIX, one being exed() which is shown below as an example:

```
#include
void main() {
exed("/bin/ls","ls",0);
printf("This text won't be printed unless an error
        occurs in exed().");  }
```

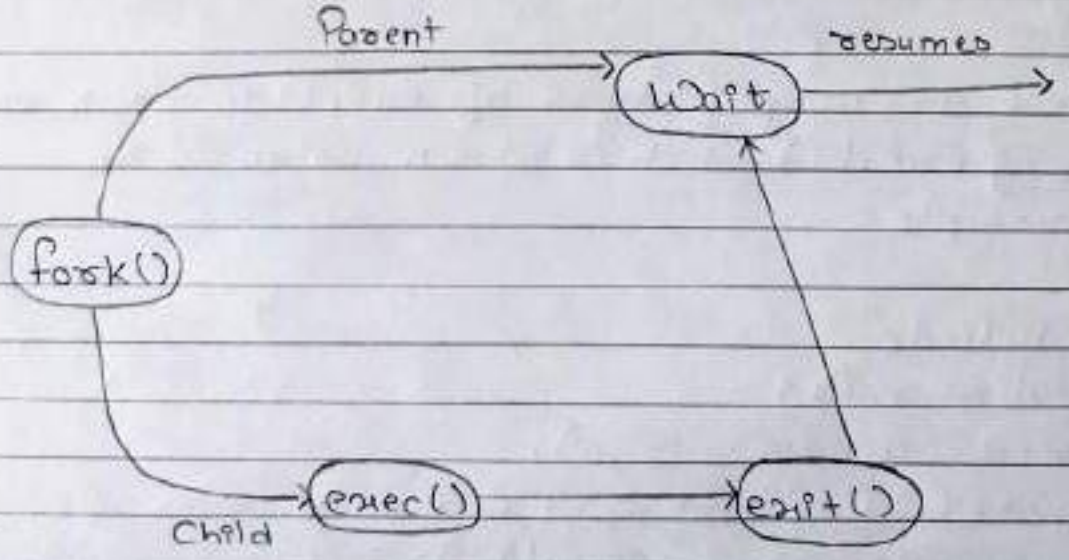
As shown above, the first parameter to the execlO function is the address of the program which needs to be executed, in the case, the address of the ls Untility in UNIX. Then it is followed by the name of the program which is ls in this case and followed by optional arguments. Then the list should be terminated by a NULL pointer (0).

-Wait()

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 from main
- It receives a Signal whose default action is to terminate.



```
#include <sys/type.h>
#include <sys/wait.h>
pid_t wait (int *status);
pid_t waitpid (pid_t pid, int *status, int options);
int waitid (idtype_t idtype, id_t id, Siginfo_t
            * infop, int options);
```

System Calls vs Library Functions

- A System Call is executed in the Kernel - p = getpid();
- A library function is executed in user Space - n = Strlen(s);
- Some library Calls are implemented with System calls - printf() really calls the write() System call
- Programs use both System Calls and library functions

Conclusion:

Thus, the process System call program is implemented and studied Various System call.

Assignment No. 3(C)

Name: Pratik Pingale

Class: TE **Roll No. 19CO056**

Subject: SPOS

****

Problem Statement : Implement UNIX system calls like ps, fork, join, exec family, and wait for process management (use shell script/ Java/ C programming).

```
#include <stdio.h>
#include
<unistd.h> int
main() {
  int choice;
  int pid;
  printf("1. ps \n2. fork \n3. exec \n4. join \n5. wait \n");
  printf("Enter your choice: ");
  scanf("%d",&choice);
  switch(choice) {
    case 1:
        execlp("ps","ps",NULL);
        break;
    case 2: pid =
        fork(); if(pid
        == 0) {
          //in child process
          printf("In child process \n");
          printf("Child process pid = %d \n",getpid());
        }
        else {
          //in parent process
          printf("Parent process pid = %d \n",getpid());
        }
        break;
```

```
    case 3: execlp("ls","ls","-l",NULL);
        printf("This will not printed because exec replace the parent process with current
        process
\n");
```

```
            break;
      case 4: execlp("join","join","file1.txt","file2.txt",NULL);
            break;
      case 5: pid =
            fork(); if(pid
            == 0) {
               printf("In child process
               \n"); sleep(2);
            }
            else {
               //wait for child process to
               complete wait();
               printf("In parent process \n");
            }
            break;
      default: printf("Invalid Input...");
            break;
   }
   return 0;
}
```

file1.txt

: Aditya

Dy patil

file2.txt

: jadhav

COE

OUTPU

T:

1. ps
2. fork
3. exec
4. join
5. wait

Enter your choice: 1

```
PID TTY      TIME CMD
16729 pts/37 00:00:00 dash
16736 pts/37 00:00:00 ps

Enter your choice: 2
Parent process pid =
11954 In child process
Child process pid = 11966

Enter your choice: 3
total 0 /* This will not printed because exec replace the parent process with current process */

Enter your choice:
4 Pratik Pingale
Dy patil COE

Enter your choice:
5 In child process
In parent process
```

Assignment No :- 4

Title : Study assignment on process Scheduling algorithm in Android and Tizen.

Objectives:
- To understand Android OS
- To understand Tizen OS
- To understand Concept of process management

Problem Statement:
Study assignment on process Scheduling algo in Android and Tizen.

Outcomes:
- Knowledge of Android and tizen OS
- Study of process management in android and tizen OS
- Application of android and tizen OS

Software Requirements:
- Android SDK

Hardware Requirements:
- M/c Lenovo Think center i3, 6th Gen, 4gb ram 500gb

Theory:
Android OS:
- Android is a mobile operating System developed by Google, based on a modified of the Linux

Kernel and other open Source Software and designed primarily for touchscreen mobile devices Such as Smartphones 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 Comers, PCs and other electronics.

- Initially developed by Android, 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 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 application in Smartphones. These application are more Comfortable and advanced for the users. The hardware that Supports android Software is based on ARAM 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.

Some android versions :

- Gingerbread
- Honey Comb
- Ice Cream Sandwich
- Jelly Beam
- Kit Kat
- Lollipop
- Marshmallow
- Nougat
- Oreo

Advantages:

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

Disadvantages :

1. Slow response
2. Heat
3. Advertisement etc

Assignment No:- 1

Title: Write a Java Program to implement paging Simulation using
1. Least Recently Used
2. Optimal algorithm

Objectives:
- To understand Concept of paging
- To Learn different page replacement algo.

Problem Statement:
Write a program to implement paging Simulation

Outcomes:
After Completion of this assignment students will be able to:
- Understand the programming language of Java
- Understand the Concept of Paging.

Software and Hardware requirenment:
- Software: Linux OS, JDK 1.7
- Hardware: Dell i3, 4gb Ram, 500gb HDD

Theory:
Paging - Paging is a memory-management scheme that permits the physical-address Space of a process to be noncontiguous. In paging, the physical memory is divided into fixed-sized blocks Called page frames and logical

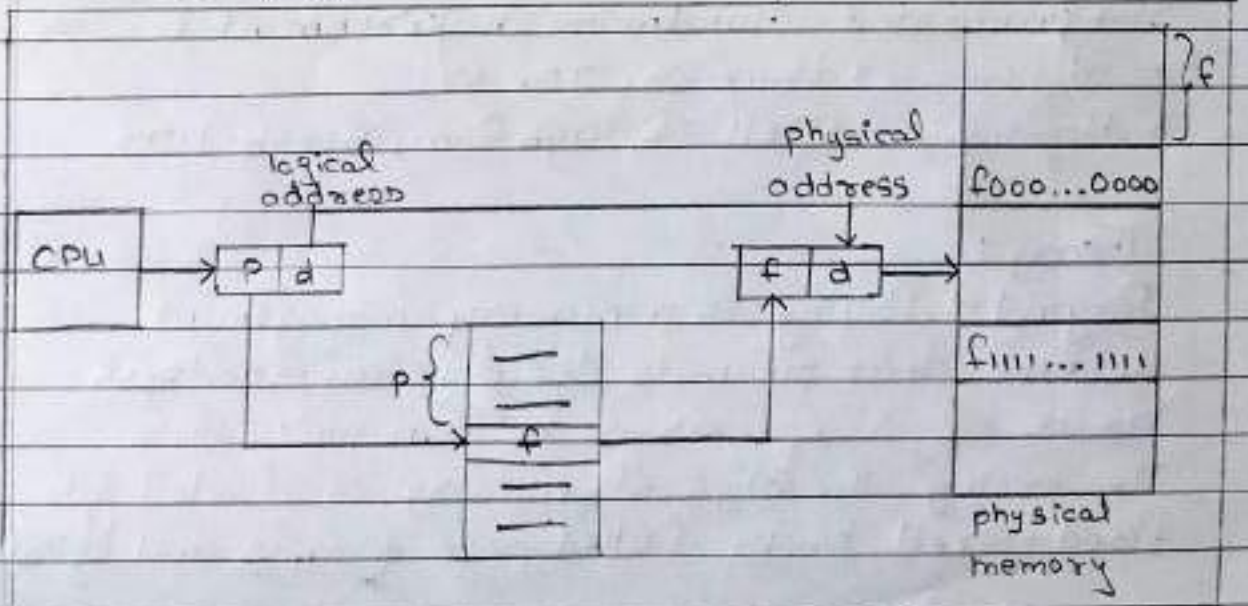
memory is also divided into fixed-size blocks called pages which are of same size as that of page frames.

When a process is to be executed, its pages can be loaded into any unallocated frames from the disk.

Consider the size of logical address space is 2^m. Now, if we choose a page size of 2^n, then n bits will specify the page offset and m-n bits will specify the page number.

How a logical address is translated into a physical address:

In paging, address translation is performed using a mapping table, called Page Table. The operating system maintains a page table for each process to keep track of which page frame is allocated to which page. It stores the frame number allocated to each page and the page number is used as index to the page table.



LRU

- Replaces the page that has not been referenced for the longest time;
- By the principle of locality, this should be the page least likely to be referenced in the near future.
- Performs nearly as well as the optimal policy.

Reference String

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

page frames

1
1
0
7

Optimal:

Optimal Page Replacement refers to the removal of the page that will not be used in the future, for the longest period of time.

reference String

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

page framed

Conclusion:

The various memory management page replacement algorithms were studied and Successfully implemented.

Assignment No. 1(D)

Name: Pratik Pingale
Class: TE Roll No. 19CO056
Subject: SPOS
****
**
**

Problem Statement : Write a Java Program (using OOP features) to simulate page replacement algorithms(FIFO,LRU& OPTIMAL).)

LRU

```
import java.io.*;
    class lru
     {
     public static void main(String args[])throws IOException
     {
                 BufferedReader obj=new BufferedReader(new
                 InputStreamReader(System.in)); int f,page=0,ch,pgf=0,n,chn=0;
                 boolean flag;
                 int pages[]; //pgf-page fault

               System.out.println("1.LRU
               "); int pt=0;
        System.out.println("enter no. of
               frames: ");
               f=Integer.parseInt(obj.readLine(
               )); int frame[]=new int[f];

               for(int i=0;i<f;i++)

               {
                        frame[i]=-1;
               }
               System.out.println("enter the no of pages ");
               n=Integer.parseInt(obj.readLine());

             pages=new int[n];
               System.out.println("enter the page
               no ");

               for(int j=0;j<n;j++)
               pages[j]=Integer.parseInt(obj.read
               Line());
```

```
int pg=0; for(pg=0;pg<n;pg++)
```

```
{
        page=page
        s[pg];
        flag=true;
        for(int
        j=0;j<f;j++)
        {
                if(page==frame[j])
            {
                flag=false; break;
            }
        }
        int temp,h=3,i; if(flag)
    {
        if( frame[1]!=-1 && frame[2]!=-1 && frame[0]!=-1)
            {
                temp=pages[pg-3];
                if(temp==pages[pg-2] ||
                temp==pages[pg-1])
                temp=pages[pg-4];

                    f
                    o
                    r
break;              (
                    i
                    =
                    0
                    ;
                    i
                    <
                    f
                    ;
                    i
                    +
                    +
                    )
                    i
                    f
                    (
                    t
                    e
                    m
                    p
```

=
=
f
r
a
m

e
[
i
]
)

frame[i]=pages[pg];

}
else
{

System.out.print("frame
:");

} for(int j=0;j<f;j++) else
System.out.print(frame[j]+" ");
{ System.out.println();

}

```
                }//for

        System.out.println("Page

        fault:"+pgf);

}//main
}//class


/*
```

OUTPUT:

```
    1
    0
    1
    2
    3
    7
    8
    1
    5 2 frame :1 -1 -1
    -1 frame :1 0 -1 -
    1 frame :1 0 -1 -1
    frame :1 0 2 -1
    frame :1 3 2 -1
    frame :7 3 2 -1
    frame :7 3 8 -1
    frame :7 1 8 -1
    frame :5 1 8 -1
       1.       LRU frame :5
    1 2 -1      e
    nter no. of
    frames: 4
    enter the no
    of pages 10
    enter the
    page no
```

Optimal

```
import java.util.*;
import java.io.*;


class Optimal
{
        public static void main(String args[])throws IOException
        {
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in)); int numberOfFrames, numberOfPages,
                flag1, flag2, flag3, i, j, k, pos = 0, max;


                int faults = 0;
                int temp[] = new int[10];

                System.out.println("Enter number of Frames: ");
                numberOfFrames = Integer.parseInt(br.readLine());
                int frame[] = new int[numberOfFrames];



                System.out.println("Enter number of Pages: "); numberOfPages
                = Integer.parseInt(br.readLine());

                int pages[] = new int[numberOfPages];
                System.out.println("Enter the pages: ");
                for(i=0; i<numberOfPages; i++) pages[i]
                = Integer.parseInt(br.readLine());


                for(i = 0; i < numberOfFrames; i++) frame[i]
              = -1;


                for(i = 0; i < numberOfPages; ++i){ flag1
                    = flag2 = 0;


                     for(j = 0; j <
                       numberOfFrames; ++j){
                       if(frame[j] == pages[i]){
                       flag1 = flag2 = 1; break;
                         }
                     }
```

```
                    if(flag1 == 0){
                      for(j = 0; j < numberOfFrames; ++j){
                                        if(frame[j] == -1){
                          faults++; frame[j] =
                          pages[i]; flag2 = 1;
                          break;
                        }
                      }
                    }
if(flag2 == 0){
flag3 =0;

                  for(j = 0; j < numberOfFrames; ++j){
                                        temp[j] = -1;

                      for(k = i + 1; k <
                        numberOfPages; ++k){
                        if(frame[j] == pages[k]){
                          temp[j] = k; break;
                                              }
                        }
                      }

                for(j = 0; j < numberOfFrames; ++j){
                                    if(temp[j] == -1){
                      pos = j; flag3
                      = 1; break;
                    }
                  }

                if(flag3 ==0){ max
                  = temp[0]; pos
                    = 0;

                      for(j = 1; j < numberOfFrames; ++j){
                                      if(temp[j] > max){
```

```
                    max = temp[j]; pos
                    = j;
                }
            }
        }

        frame[pos] = pages[i]; faults++;
    }

//      System.out.print();

        for(j = 0; j < numberOfFrames; ++j){
            System.out.print("\t"+ frame[j]);
        }
    }

    System.out.println("\n\nTotal Page Faults: "+ faults);



    }


}

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