* Tejas vijay khairnar
* Roll No-13 T.E * Sub - SPOS

CLASSMATE
Date :
Page :

INDEX

Group C

(10)	write a Java program [using oop features] to implement following scheduling algorithms: FCFS, SJF (premptive), priority (non-prepmptive) and round robin (preemptive)	22
(11)	write a Java program to implement Banker's Algorithm.	29
(12)	Implement UNIX system calls like ps, fork, join, exec family, and wait for process management [use shell script / Java / c programming]	34
(13)	study assignment on process scheduling algorithm in android and Tizen.	43

Group D

(14)	write a Java program [using oop features] to implement paging simulation using ① Least recently used [LRU] ② optimal algorithm	45

NAME - Tejas Vijay Khairnar

Roll No - 13

Class - TE

Sub - SPOS

Name - tejas Vijay khairnar
Roll No - 13 T.E
Sub - SPOS

Group - A

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 instructions 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 program to create pass-I Assembler.

outcomes -

After completion of this assignment students will be also to:

- understandard the concept of pass-I Assembler
- understand the programming language of java

Software requirements -

Linux OS. JDK 1.7

Hardware requirements -

- 4GB RAM, 500GB HDD

Theory Concepts -

A language translator bridges an execution gap to machine language of computer system. an assembler is a language translator whose source language is assembly language.

An Assembler is a program that accepts as input an assembly language program and converts it into machine language.

Language processing activity consists of two phase, Analysis phase & synthetic phase. analysis of source program consists of three components, Lexical rules, syntax rules & semantic rules. lexical rules govern the formation of valid statements in source language. semantic rules associate the formation meaning with valid statements of language. synthesis phase is concerned with construction of target language statements, which have the same meaning as source language statements. This consist of memory allocation and code generation.

Two Pass translation scheme -

In a 2-pass assembler, the first pass constructs an ~~int~~ intermediate representation of the source program for use by the second pass. This representation consists of two main components - data structures like symbol ~~ti~~ table. literal table & processed form of the source program called as intermediate code (IC). This intermediate code is represented by the syntax of varient - I

Analysis of source program statements may not be immediately followed by synthetic of equivalent target statements. this is due to forward reference issue concerning memory requirements & organization of language processor [LP]

forward reference of a program entity is a reference to the entity, which preceds its defination in the program. This problem can be solved by postponing the generation of targetion of target code until more information concerning the entity is available. This also reduces memory requirements of LP & simplifies its organization. This leads to multi-pass model of language processing.

Language processor pass -
It is the processing of every statement in a source program or its equivalent representation to perform language-processing function.

Assembly Language statements - there are three types of statements
Impreatives, declarative, Assembly directives, an imprative statement indicates an action to be performed during the execution of assembled program. Assembler directives instruct the assembler to perform certain actions during assembly of a program.
e.g- START <constant> directive indicates that the first word of the target program generated by assembler should be placed at memory word with address <constant>.

Function of Analysis and Synthesis phase-

Analysis phase-

Isolate the label operation code & operand fields of a statement.

Enter the symbol found in label field [if any] & 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 & operand field of the statement.

calculate the address of the address of the first machine word following the target code generated for this statement [Location counter processing]

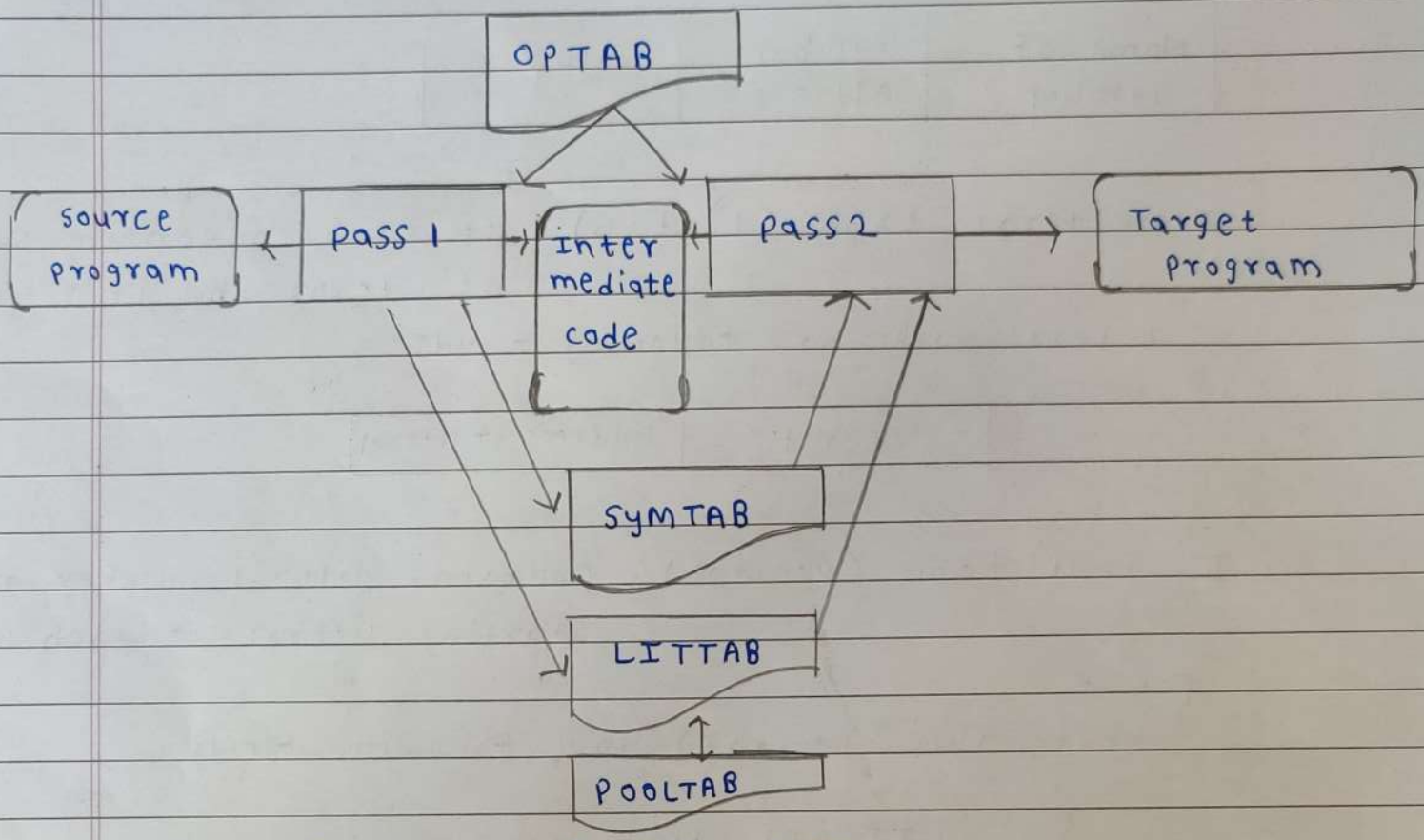
synthesis phase-

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 from 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 as follows.

b] Data structure updated during translation: Also called as translation time data structure. they are

I. Symbol table (symtab) : Ti contains entires such symbol, it"s address and value.

Symbol table have following fields -

Name of Symbol	Symbol Address	value

II Literal table (Littab): It contains entires such as literal and it"s value.

Literal table has following fields:

literal	Address of literal

III Pool table (pooltab): contains literal number of the starting literal of each literal pool.

Pool table (pooltab) have following fields.

LITERAL - No

IV - Location counter which contains address of next instruction by calculating length of each instruction.

Design of a Two pass Assembler -

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

pass I - separate the symbol, mnemonic opcode & operand fields.

Determine the storage - required for every assembly language statement & update the location counter.

Build the symbol table & 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]

Declaration statements		Assembler directives	
DC	01	START	01
DS	02	END	02
		~~ORG~~ ORIGIN	03
		EQU	04
		LTORG	05

Algorithms (procedure): Pass I

- Initalize locations counter, entires of all tables as zero
- Read statements from input file one by one
- while next statement is not END statement.

I. Tokenize or separate out input statement as label, numonic, operand 1, operand 2

II. If label is present insert label into symbol table

III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table & allocate memory.

IV. If statement is START or ORIGEN process location counter accordingly.

~~V. If statement is START~~

V. If an EQU statement, assign value to symbol by correcting entry is symbol table.

VI. for declarative statement update code, size & location counter.

VII. generate intermediate code.

VIII. pass this intermediate code to pass-2.

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

EXPT 01 INPUT

```
START    100
         READ     A
LABLE    MOVER    A,B
         LTORG
                  ='5'
                  ='1'
                  ='6'
                  ='7'
         MOVEM    A,B
         LTORG
                  ='2'
LOOP     READ     B
A        DS       1
B        DC       '1'
                  ='1'
         END
```

Expt 1 output

```
    SYMBOL TABLE
--------------------------
SYMBOL  ADDRESS LENGTH
--------------------------

LABLE   101     1

A       112     1
B-------113-------1-------


    OPCODE TABLE
----------------------------

MNEMONIC----CLASS---INFO----
READ        IS  (04,1)

MOVER       IS  (04,1)
LTORG       AD  R11
MOVEM       IS  (04,1)
READ        IS  (04,1)
DS      DL  R7
DC      DL  R7
END     AD  R11
----------------------------


   LITERAL TABLE
-----------------
LITERAL ADDRESS
-----------------
='5'    104
='1'    105
='6'    106
='7'    107
='2'    110
='1'    114
-----------------


---POOL-TABLE----
LITERAL NUMBER
----------------
1
5
6
-----------------
```

```java
//SPOS EXPT1
import java.io.*;
class SymTab
{
    public static void main(String args[])throws Exception
    {
        FileReader FP=new FileReader(args[0]);
        BufferedReader bufferedReader = new BufferedReader(FP);

        String line=null;
        int line_count=0,LC=0,symTabLine=0,opTabLine=0,litTabLine=0,poolTabLine=0;

         //Data Structures
         final int MAX=100;
         String SymbolTab[][]=new String[MAX][3];
         String OpTab[][]=new String[MAX][3];
         String LitTab[][]=new String[MAX][2];
         int PoolTab[]=new int[MAX];
         int litTabAddress=0;
/*---------------------------------------------------------------------------------------------------*/


System.out.println("___________________________________________________");
            while((line = bufferedReader.readLine()) != null)
             {

                 String[] tokens = line.split("\t");
                if(line_count==0)
                {
//set LC to operand LC=Integer.parseInt(tokens[2]);

                    for(int i=0;i<tokens.length;i++)        //for printing the input program
                        System.out.print(tokens[i]+"\t");
                    System.out.println("");
                }
                else
                {
                     for(int i=0;i<tokens.length;i++) //for printing the input program
                        System.out.print(tokens[i]+"\t");
                     System.out.println("");
                    if(!tokens[0].equals(""))
                    {

                        //Inserting into Symbol Table
                        SymbolTab[symTabLine][0]=tokens[0];
                        SymbolTab[symTabLine][1]=Integer.toString(LC);
                        SymbolTab[symTabLine][2]=Integer.toString(1);
                        symTabLine++;
                }
                else if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                {
                    //Entry into symbol table for declarative statements
                    SymbolTab[symTabLine][0]=tokens[0];

                        SymbolTab[symTabLine][1]=Integer.toString(LC);
                        SymbolTab[symTabLine][2]=Integer.toString(1);
                }       symTabLine++;

                if(tokens.length==3 && tokens[2].charAt(0)=='=')
                {
                    //Entry of literals into literal table
                    LitTab[litTabLine][0]=tokens[2];

                        LitTab[litTabLine][1]=Integer.toString(LC);
                }       litTabLine++;

                else if(tokens[1]!=null)
                {
                        //Entry of Mnemonic in opcode table
                    OpTab[opTabLine][0]=tokens[1];


if(tokens[1].equalsIgnoreCase("START")||tokens[1].equalsIgnoreCase("END")||tokens[1].equalsIgnoreCase("ORIGIN")||tokens[1].equalsIgnoreCase("EQU")||tokens[1].equalsIgnoreCase("LTORG"))        //if Assembler Directive
                    {

                        OpTab[opTabLine][1]="AD";

                    else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                    {
                        OpTab[opTabLine][1]="DL";

                    }   OpTab[opTabLine][2]="R7";
                    else
                    {
                        OpTab[opTabLine][1]="IS";
                        OpTab[opTabLine][2]="(04,1)";
                    }

                }   opTabLine++;
                }
                line_count++;
                LC++;
            }


System.out.println("___________________________________________________");

            //print symbol table
            System.out.println("\n\n    SYMBOL TABLE        ");

            System.out.println("SYMBOL\tADDRESS\tLENGTH");

            for(int i=0;i<symTabLine;i++)
                System.out.println(SymbolTab[i][0]+"\t"+SymbolTab[i][1]+"\t"+SymbolTab[i][2]);
            System.out.println("------------------------------");


            //print opcode table
            System.out.println("\n\n    OPCODE TABLE        ");
            System.out.println("--------------------------------");
            System.out.println("MNEMONIC\tCLASS\tINFO");
            System.out.println("--------------------------------");
            for(int i=0;i<opTabLine;i++)
                System.out.println(OpTab[i][0]+"\t\t"+OpTab[i][1]+"\t"+OpTab[i][2]);
            System.out.println("--------------------------------");

            //print literal table
            System.out.println("\n\n   LITERAL TABLE        ");
            System.out.println("-----------------");
            System.out.println("LITERAL\tADDRESS");
            System.out.println("-----------------");
            for(int i=0;i<litTabLine;i++)
                System.out.println(LitTab[i][0]+"\t"+LitTab[i][1]);
            System.out.println("-----------------");


            //intialization of POOLTAB
            for(int i=0;i<litTabLine;i++)
            {
                if(LitTab[i][0]!=null && LitTab[i+1][0]!=null ) //if literals are present
                {
                    if(i==0)
                    {
                        PoolTab[poolTabLine]=i+1;
                        poolTabLine++;
                    }
                    else if(Integer.parseInt(LitTab[i][1])<(Integer.parseInt(LitTab[i+1][1]))-1)

                    {   PoolTab[poolTabLine]=i+2;
                        poolTabLine++;
                    }
                }
            }
            //print pool table

            System.out.println("\n\n---POOL TABLE");    ");
            System.out.println("LITERAL NUMBER");
            System.out.println("-----------------");
            for(int i=0;i<poolTabLine;i++)
                System.out.println(PoolTab[i]);
            System.out.println("-----------------");


            // Always close files.
            bufferedReader.close();
    }
}
```

Name - Tejas Vijay Khairnar
Roll No - 13 T.E
Sub - SPOS

CLASSMATE
Date :
Page : 10

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 [intermediate file and symbol table] 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 requirements -

Linux OS, Jdk 1.7

Hardware requirement -

- 4GB RAM, 500 GB HDD

Theory Concept -

Design of a two pass Assembler -

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

pass I -

separate 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 & 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 (procedure) -

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 [POOLTAB [pooltab_ptr]] ---
LITTAB [POOLTAB [tab_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 stament 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 adddress from SYMTAB or LITTAB

II] Assemble instruction in machine code buffer.

III] size = size of instruction;

f] if size #0

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 & implemented dynamic link library application for arithmatic operation.

EXPT 02 INPUT

```
intermediate code -
(AD,01)(C,200)
(IS,04)(1)(L,1)
(IS,05)(1)(S,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 --

AOOP            201         1

NEXT            208         1
BACK            202         1
LAST            210         1

literal table --

5       206
1       207
1       213
```

EXPT 02 OUTPUT

```
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
```

```java
//SPOS Expt 2
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");
            }
            else{
                f1.write("\n");
            }
        }
        f1.close();
        b1.close();
        b2.close();
        b3.close();
    }
}
```

Name- Tejas Vijay Khairnar
Roll No- 13 T.E
Sub- SPOS

CLASSMATE
Date :
Page: 14

Assignment No-3

Tittle - Design suitable data structures & implement pass-I of a two-pass macro-processor. using oop features in java.

objectives-

- To identify & create the data structure 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-1 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 requirements -

Linux OS, JDK 1.7

Hardware requirement -

- 4GB RAM, 500 GB HDD.

Theory concepts -

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 occurs is a program the sequence of codes is substituted at that point.

A macro consist of

1. Name of the macro
2. set of paremeters
3. Body of macro

macro are typically defined at the start of program. macro definition consist of

1. MACRO Pseudo
2. MACRO name.
3. Sequence of statements
4. MEND pseudo opcode terminating

A macro is called by writting the macro name with actual parameter is an assembly program.

The macro call has following syntax

< macro name >.

MACRO PROCESSOR -

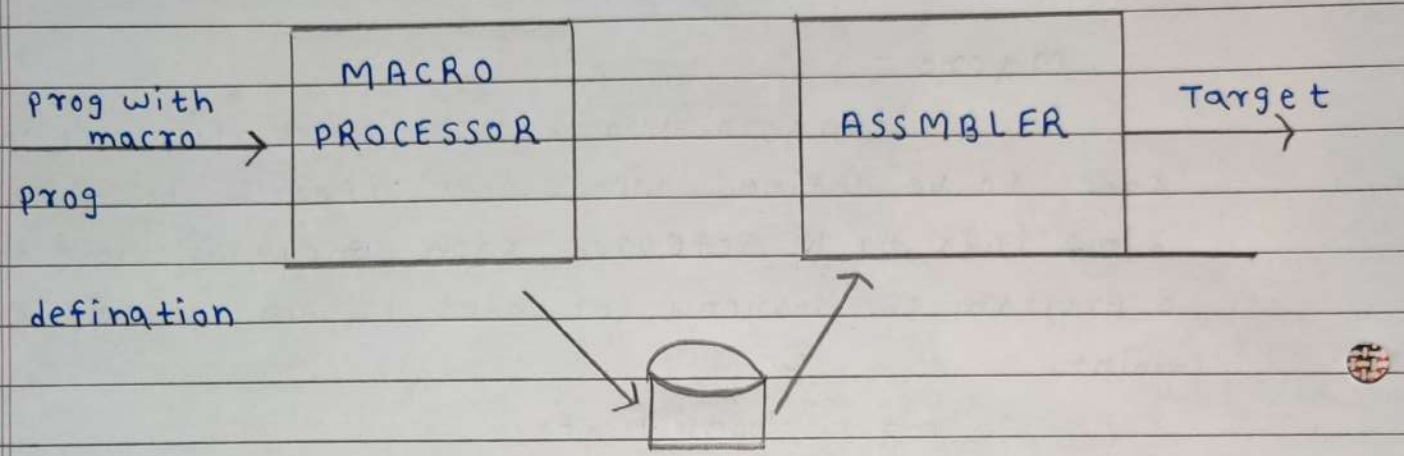


fig1 - Assembly language Program without macro.

macro processor takes a source program containing macro defination & macro calls & translates into an assembly language program without any macro defination or calls. This program can now be handled over to a conventional assembler to obtain the target language

MACRO Deinition -

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 definition

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 preceeded by the character, , & " 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 definition table pointer to the corresponding macro definition started in MDT. The initial value of MDT is obtained from MDT index.

Design of macro processor -

pass - I -

generate Macro Name Table (MNT)
generate macro Definition Table (MDT)
generate IC ie- a copy of source code without macro definitions.

~~MNT~~ : specification of Data Bases. -

Pass I data bases -

① The input macro source desk.
② The output macro source desk copy for use by passes 2.
③ The macro definition table [MDT] used to store the names of defined macros.
④ macro name table [MDT] used to stare the name of defined macros.
⑤ The macro definition 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 definition.

conclusion - Thus we have successfully implemented pass - I of a two - pass macro - processor.

```
MACRO
INCR1       &FIRST,&SECOND=DATA9
A           1,&FIRST
L           2,&SECOND
MEND
MACRO
INCR2       &ARG1,&ARG2=DATA5
L           3,&ARG1
ST          4,&ARG2
MEND
PRG2        START
            USING                   *,BASE
            INCR1                   DATA1
            INCR2                   DATA3,DATA4
FOUR        DC                      F'4'
FIVE        DC                      F'5'
BASE        EQU                     8
TEMP        DS                      1F
            DROP                    8
            END
```

EXPT 03 INPUT

```
PRG2        START
            USING                   *,BASE
            INCR1                   DATA1
            INCR2                   DATA3,DATA4
FOUR        DC                      F'4'
FIVE        DC                      F'5'
BASE        EQU                     8
TEMP        DS                      1F
            DROP                    8
            END
```

EXPT 03 OUTPUT

```java
import java.util.*;
import java.io.*;
class MACRO
{
static String mnt[][]=new String[5][3]; //assuming 5 macros in 1 program
static String ala[][]=new String[10][2]; //assuming 2 arguments in each macro
static String mdt[][]=new String[20][1]; //assuming 4 LOC for each macro
static int mntc=0,mdtc=0,alac=0;
public static void main(String args[])
{
pass1();
System.out.println("\n********PASS-1 MACROPROCESSOR**********\n");
System.out.println("MACRO NAME TABLE (MNT)\n");
System.out.println("i macro loc\n");
display(mnt,mntc,3);
System.out.println("\n");
System.out.println("ARGUMENT LIST ARRAY(ALA) for Pass1\n");
display(ala,alac,2);
System.out.println("\n");
System.out.println("MACRO DEFINITION TABLE (MDT)\n");
display(mdt,mdtc,1);
System.out.println("\n");
}
static void pass1()
{
int index=0,i;
String s,prev="",substring;
try
{
BufferedReader inp = new BufferedReader(new FileReader("input.txt"));
File op = new File("pass1_output.txt");
if (!op.exists())
op.createNewFile();
BufferedWriter output = new BufferedWriter(new FileWriter(op.getAbsoluteFile()));
while((s=inp.readLine())!=null)
{
if(s.equalsIgnoreCase("MACRO"))
{
prev=s;
for(;!(s=inp.readLine()).equalsIgnoreCase("MEND");mdtc++,prev=s)
{
if(prev.equalsIgnoreCase("MACRO"))
{
StringTokenizer st=new StringTokenizer(s);
String str[]=new String[st.countTokens()];
for(i=0;i<str.length;i++)
str[i]=st.nextToken();
mnt[mntc][0]=(mntc+1)+""; //mnt formation
mnt[mntc][1]=str[0];
mnt[mntc++][2]=(++mdtc)+"";
st=new StringTokenizer(str[1],","); //tokenizing the arguments
String string[]=new String[st.countTokens()];
for(i=0;i<string.length;i++)
{
string[i]=st.nextToken();
ala[alac][0]=alac+""; //ala table formation
index=string[i].indexOf("=");
if(index!=-1)
ala[alac++][1]=string[i].substring(0,index);
else
ala[alac++][1]=string[i];
}
}
else //automatically eliminates tagging of arguments in definition
{ //mdt formation
index=s.indexOf("&");
substring=s.substring(index);
for(i=0;i<alac;i++)
if(ala[i][1].equals(substring))
s=s.replaceAll(substring,"#"+ala[i][0]);
}
mdt[mdtc-1][0]=s;
}
mdt[mdtc-1][0]=s;
}
else
{
output.write(s);
output.newLine();
}
}
output.close();
}
catch(FileNotFoundException ex)
{
System.out.println("UNABLE TO END FILE ");
}
catch(IOException e)
{
e.printStackTrace();
}
}
static void display(String a[][],int n,int m)
{
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
System.out.print(a[i][j]+" ");
System.out.println();
}
}
}
```

Name- Tejas vijay khairnar
Roll No- 13 T.E
Sub- SPOS

CLASSMATE
Date :
Page : 19

Assignment No- 4

Title- 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.

objectives-

- To Identify & create the data structure required in the design of macro processor.
- To learn paramming language ~~of Java~~ 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 requirements-

Linux OS, JDK 1.7

Hardware requirements-

- 4GB RAM, 500 GB HDD

Theory concept -

Pass II -

replace every occurance of macro call with macro definition [Expanded code]

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

~~Pass II~~ ① Recognize macro definition -

A macro instruction processor must recognize macro definition identified by the MACRO & MEND pseudo-OPS. This tasks can be complicated when macro definition appers within macros. when MACRO's and MENDS are nested, as the macro processor must recognize the nesting & correctly match the last or outer MEND with first MACRO. All intervning text, including nested MACRO'S & MEND defines a single macro instruction.

② Save the definition. - 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 appers as operation mnemonics. This suggests that macro ~~name~~ names be handled as a type of op-code.

④ Expand calls & substitute arguments -

The processor must substitute for dummy or macro definition arguments the corresponding arguments from a macro call; the resulting symbolic text is then substitute for macro call. This txt may contain additional macro definition or call.

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

EXPT 04 INPUT

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

Kpdt–


a    AREG
b    -
u    CREG
v    DREG

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
```

EXPT 04 OUTPUT

```
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'
```

```java
//SPOS Expt4
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(word[3]));
            kpdtpHash.put(s1,Integer.parseInt(word[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],paramNo);
                    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],paramNo);
                    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].length()))
,initializedParams[1].substring(0,initializedParams[1].length()));
                    i++;
                }
                i=mdtp-1;
                while(mdt.get(i).compareToIgnoreCase("MEND")!=0){
                    f1.write("+ ");
                    for(int j=0;j<mdt.get(i).length();j++){
                        if(mdt.get(i).charAt(j)=='#')
                            f1.write(aptab.get(Integer.parseInt("" +
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();
    }
}

```

Name - Tejas vijay khairnar
Roll No - 13 T.E
Sub - SPOS

Group - C

Assignment No - 1

Title -

write a java program [using oop features] to implement following scheduling algorithms : FCFS, SJF (preemptive), Priority [Non-premptive] & round robin [preemptive].

objectives -

- To understand os & scheduling concepts.
- To implement scheduling FCFS, SJF, RR & priority algorithm.
- To study about scheduling & scheduler.

problem statement -

write a java program (oop features) to implement following scheduling algorithms : FCFS, SJF, priority & Round Robin

outcomes -

- knowledge scheduling policies.
- compare different scheduling algorithms.

Software requirements -

JDK / Eclipse

Hardware requirement -

- M/c Lenovo think center M700 / Ci 3, 6100, 6th Gen. H81, 4GB RAM, 500GB HDD

Theory concepts -

CPU scheduling -

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

• 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?~~

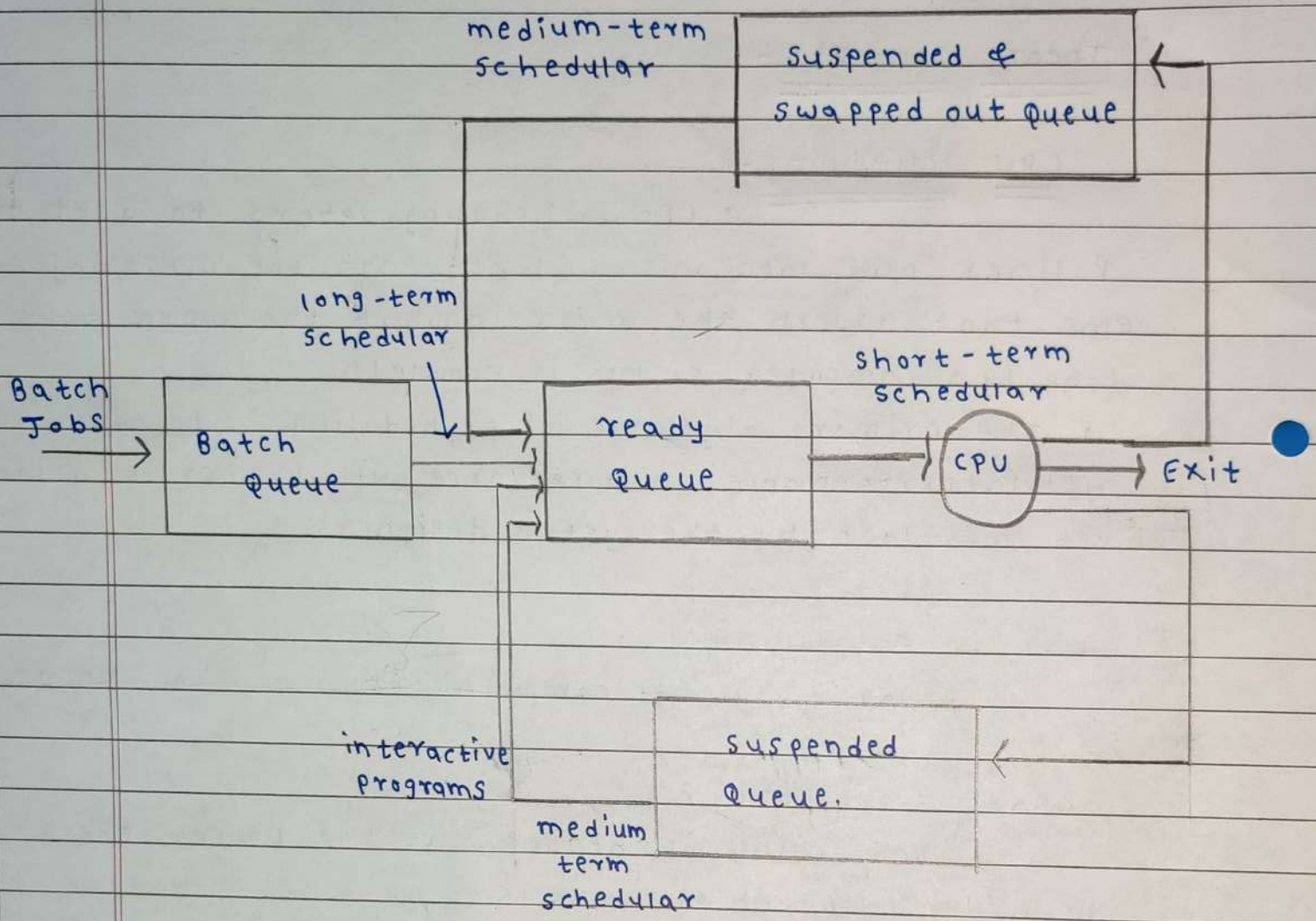
~~Department of computer Engineering, MMIT,~~

what is scheduling ?

Scheduling is defined as the process the govern 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 ie- scheduling that means how the jobs are to be performed ie- order CPU sechedulling is best example of scheduling

Types of schedulers -

① Long term scheduler
② medium term schedular
③ short term schedular.


medium-term scheduler
Suspended & swapped out Queue
long-term scheduler
short-term scheduler
Batch Jobs
Batch Queue
ready Queue
CPU
Exit
interactive programs
Suspended Queue.
medium term scheduler

Scheduling Criteria -

- CPU utilization -

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

- Throughput -

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

- Turnaround time -

This is the how long a process taken to execute a process. It is calculated as the time gap between the submission of a process & it's completion.

- waiting time -

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

- Types of scheduling Algorithms -

- In general, scheduling disciplines may be pre-~~or~~ emptive or non-pre-emptive.

- In batch, non-pre-emptive implies that once scheduled, a selected job turns to completion.

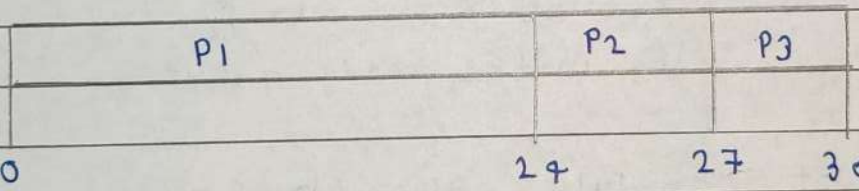
there are different types of scheduling algorithm such as :

① FCFS [first come first serve]
② SJF [short Job first]
③ Priority scheduling
④ Round Robin scheduling algorithm

① first come, first served. ⟶

process	Brust time
P1	24
P2	3
P3	3

- suppose that the process arrive in the order- P1, P2, P3
- the Gantt chatt for the schedule is

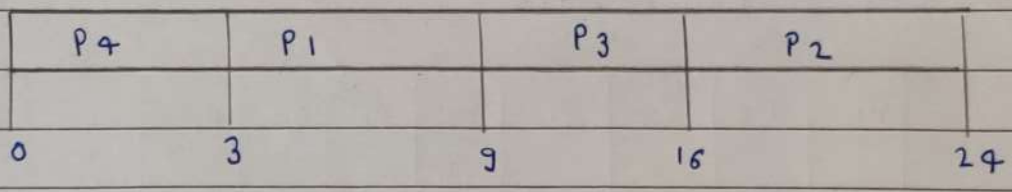


- waiting time for P1 = 0 ; P2 = 24 ; P3 = 27
- Average waiting time = (0 + 24 + 27)/3 = 17

② short Job first.

process	Brust time
P1	6
P2	8
P3	7
P4	3

- SJF scheduling chart

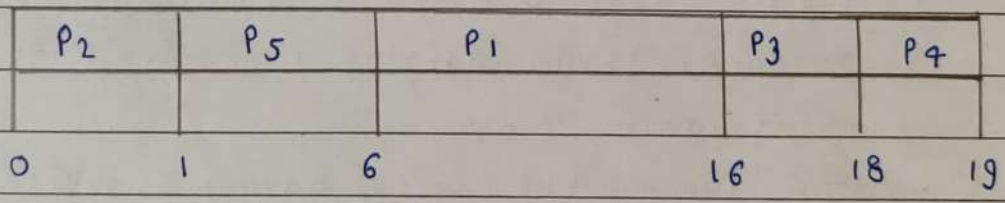


- Average waiting time = (3 + 16 + 9 + 0)/4 = 7

③ Priority scheduling -

Process	Brust time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Gantt chatt.

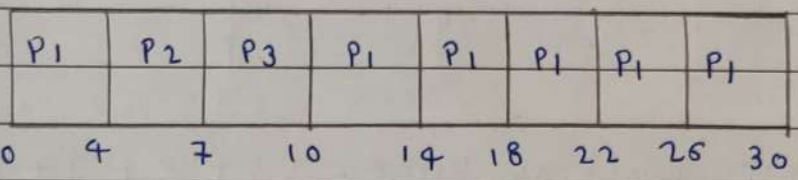


- Average waiting time = (6 + 0 + 16 + 18 + 1)/5 = 8.2

④ round robin scheduling -

process	Brust time
P1	24
P2	3
P3	3

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



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

~~flow chart~~

Conclusion - Hence we have studied that-

- CPU scheduling concept like context switching, types of schedulers, different timing parameter like waiting time, turnaround time, burst time etc.
- Different CPU Scheduling algorithm 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

```java
//1.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");
    }
}
```

```
/* FCFS 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  */
```

```java
/*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();
            for(int i=0;i<Pno;i++)
            {

                if(Burst_time[i]>TimeQuantum)
                {   Burst_time[i]-=TimeQuantum;
                    for(int j=0;j<Pno;j++)
                    {
                        if((j!=i)&&(Burst_time[j]!=0))
                        WT[j]+=TimeQuantum;

                }   }
                else
                {
                    for(int j=0;j<Pno;j++)
                    {
                            if((j!=i)&&(Burst_time[j]!=0))

                        }   WT[j]+=Burst_time[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));

}   }
```

```
/* RR OUTPUT::
unix@unix-HP-280-G1-
MT:~/TEA33$ java RoundR
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   11  13
Average waiting time7.2
Average turn around time7.2       */
```

```java
/*          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 (NON PREEMPTIVE) OUPUT

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Abhishek\java> cd "d:\Abhishek\CPP\c++ course\" ; if ($?) { javac SJF1.java } ; if ($?) { java SJF1 }
Enter number of process: 5

Enter Burst time:

Process[1]: 6

Process[2]: 2

Process[3]: 8

Process[4]: 3

Process[5]: 4

Process  Burst Time     Waiting Time    Turnaround Time

 p2              2               0               2

 p4              3               2               5

 p5              4               5               9

 p1              6               9               15

 p3              8               15              23


Average Waiting Time: 6.2

Average Turnaround Time: 10.8
```

```
/*  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,to
tal_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);

}
}
```

SJF (PREEMPTIVE) OUTPUT

```
PS D:\Abhishek\java course> cd "d:\Abhishek\CPP\c++ course\" ; if ($?) { javac
sjf_swap1.java } ; if ($?) { java sjf_swap1 }
Enter number of process: 5

Enter Burst time:

Process[1]: 6

Process[2]: 2

Process[3]: 8

Process[4]: 3

Process[5]: 4

Enter arrival time:

Process[1]: 2

Process[2]: 5

Process[3]: 1

Process[4]: 0

Process[5]: 4
process2
process4
process5
process1
process3

Pro_number      Burst Time     completion_time Waiting Time    Turnaround Time

2               2               0              0               2

4               3               2              2               5

5               4               5              5               9

1               6               9              9               15

3               8               15             15              23


AWT: 6.2

ATAT: 10.8
```

Name- Tejas Vijay Khairnar
Roll No- 13 T.E
Sub- SPOS

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
- Implementation of banker's algorithm for deadlock detection and advoidance.

problem statement -

write a java program to implement Banker's Algorithm.

outcomes -

- knowledge Bankers Algorithms
- Application of Bankers Algorithms

software -

requirements:

~~ents~~:

IDK / Eclipse

Hardware requirement -

- M/C Lenovo think center M700 Ci3, 6000, 6th gen. H81, 4GB RAM, 500GB HDD

Theory concepts -

The Banker's algorithm, sometimes referred to as the detection algorithm. is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijksta that tests 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 systems to determine whether a loan can be granted or not.

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 there are K instances available, of resource type Rj

- Max - It is an nxm matrix which represents the maximum number of instances of each resource that a 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 nxm matrix which represents the number of resources of each type currently allocated to each process. If Allocation [i][j] = K, then process Pi is currently allocated K instances of resource type Rj

- **Need** - It is an nxm matrix which indicates the remaining resource needs of each process. If Need [i][j] = K, then process Pi may need K more instances of resource type Rj to complete its task.

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

Resource request Algorithm -

① If number of requested instances of each resource is less than the need (which was declared previously by the process), 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,

Requesti Allocationi = Allocationi + requesti

Needi = Needi - requesti

After completing the above three steps, check if the system is in safe by applying the safety algorithm. If it is in safe state, proceed to allocate the requested resources. else, the Process has to wait longer.

Safety Algorithm -

① let work and finish be vectors of length m & n, respectively. Initially.

② work = available

③ finish[i] = false for i = 0, 1, ----, n-1

④ find an index i such that both

⑤ finish[i] = = false

⑥ Needi C = 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 the then the resources, are allocated & the process is marked finished. and then, the loop is repeated to check the same for all other processes.

⑪ If finish[i] = = true for all i, then the system is in a safe state.

Advantages- Avoids deadlock and it is less restrictive than deadlock prevention.

Disadvantages- only works with fixed number of resources & processes.

- gurantees finite time - not reasonable response time
- needs advanced knowledge of maximum needs
- Not suitable for multi-access systems
- unnecessary delays in avoiding unsafe states which may not lead to deadlock.

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

EXPT 11 OUTPUT

```
Enter no. of processes: 5
Enter no. of Resources : 3
Enter allocation matrix -->
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix -->
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix -->
3 3 2
Allocated process : 1
Allocated process : 3
Allocated process : 4
Allocated process : 0
Allocated process : 2

Safely allocated
```

```java
//SPOS expt 11 BAnkers algorithm
import java.util.Scanner;
public class Bankers{
    private int need[][],allocate[][],max[][],avail[][],np,nr;

    private void input(){
     Scanner sc=new Scanner(System.in);
     System.out.print("Enter no. of processes: ");
     np=sc.nextInt();  //no. of process
     System.out.print("Enter no. of Resources : ");
     nr=sc.nextInt();  //no. of resources
     need=new int[np][nr];  //initializing arrays
     max=new int[np][nr];
     allocate=new int[np][nr];
     avail=new int[1][nr];

     System.out.println("Enter allocation matrix -->");
     for(int i=0;i<np;i++)
          for(int j=0;j<nr;j++)
         allocate[i][j]=sc.nextInt();  //allocation matrix

     System.out.println("Enter max matrix -->");
     for(int i=0;i<np;i++)
          for(int j=0;j<nr;j++)
         max[i][j]=sc.nextInt();  //max matrix

        System.out.println("Enter available matrix -->");
        for(int j=0;j<nr;j++)
         avail[0][j]=sc.nextInt();  //available matrix

        sc.close();
    }

    private int[][] calc_need(){
       for(int i=0;i<np;i++)
         for(int j=0;j<nr;j++)  //calculating need matrix
          need[i][j]=max[i][j]-allocate[i][j];

       return need;
    }

    private boolean check(int i){
       //checking if all resources for ith process can be allocated
       for(int j=0;j<nr;j++)
       if(avail[0][j]<need[i][j])
          return false;

    return true;
    }

    public void isSafe(){
       input();
       calc_need();
       boolean done[]=new boolean[np];
       int j=0;

       while(j<np)
      {
//until all process allocated
       boolean allocated=false;
       for(int i=0;i<np;i++)
        if(!done[i] && check(i)){
 //trying to allocate
            for(int k=0;k<nr;k++)
            avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
         System.out.println("Allocated process : "+i);
         allocated=done[i]=true;
               j++;
             }
          if(!allocated) break;  //if no allocation
       }
       if(j==np)  //if all processes are allocated
        System.out.println("\nSafely allocated");
       else
        System.out.println("All proceess cant be allocated safely");
    }

    public static void main(String[] args) {
       new Bankers().isSafe();
    }
}
```

Name - Tejas Vijay Khairnar
Roll No - 13 T.E
Sub - SPOS

CLASSMATE
Date :
Page : 34

Assignment - 3

Title - Implement UNIX system calls like PS, fork, join, exec family, and wait for process management [use shell script / Java / C programing]

objectives -

- To understand UNIX system call
- To understand concept of process management
- Implementation of some system call of OS

problem statement -

Implement UNIX system calls like PS, fork, join, exec family, and wait for process management [use shell script / Java / C programming]

outcomes -

- Knowledge of system call
- compare system call and system function.
- Application of system call.

Software requirements -

GCC or JDK / Eclipse

Hardware requirement -

- M/C lenovo Think center M700 Ci3, 6100, 6th Gen, H81, 4GB RAM, 500 GB 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 that resource. This is done via something called a system call.
- when a program makes a system call, the mode is switched from user mode to kernel mode. This is called a context switch.
- Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

generally, system calls are made by the user level programs in the following situations -

- creating, opening, closing and deleting files, in the files 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 pointer.

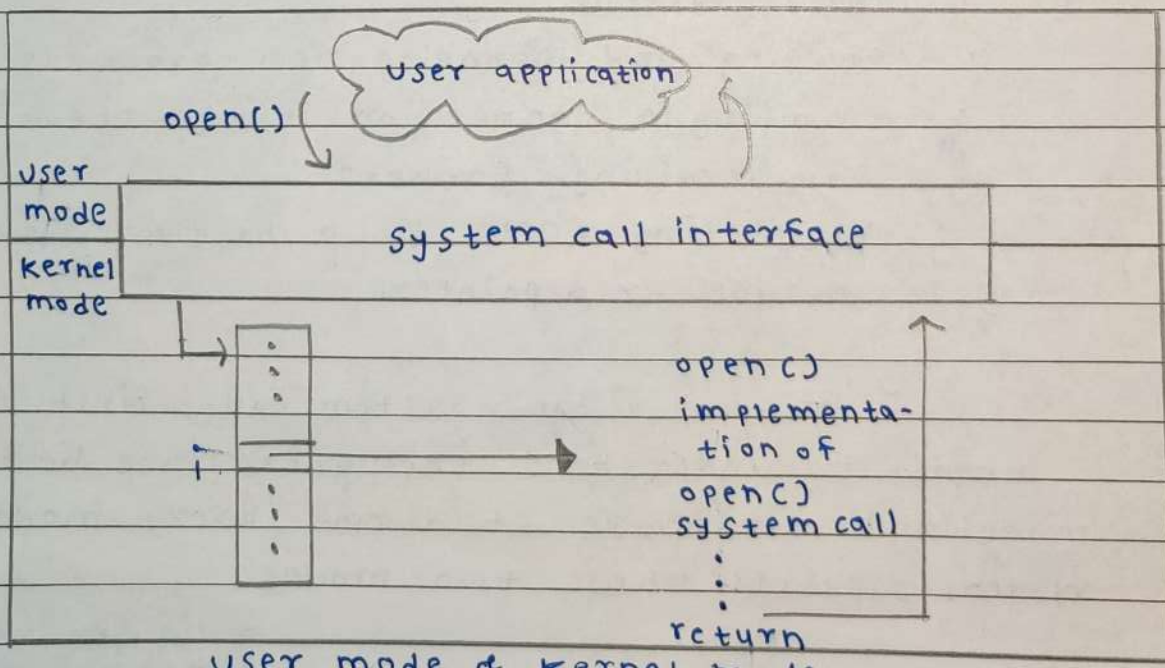
- To understand system calls, first one needs one needs to understand the difference between kernal mode and user mode of a CPU. Every modern operating system supports these two modes.

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 priviledged 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 the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode



User mode & Kernel mode

examples of window & unix system calls-

	windows	unix
process control	create process () Exit process () wait for single object ()	fork () exit () wait ()
file manipulation	create file (), read file () write file () close Handle ()	open () read () write () close ()
Device manipulation	set console mode () read console () write console ()	ioctl () read () write ()
Information maintenance	get current process ID () set timer () sleeo ()	getpid () alarm () sleep ()
communication	create pipe () create file mapping () map view o file ()	pipe shmget () mmap ()
protection	set file security () Initialize security Descrip-tor () set security Descriptor grap()	chmod () umask () chown ()

System call Basics -

- since system calls are functions, we need to include the proper header file.

eg - for getpid() we need.

- #include <sys/types.h>
- #include <unistd.h>

UNIX processes -

- recall a process is a program in execution
- process 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() returns 0
- fork() may seem strange at first, that's because it is a bit strange!
- 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-

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

```
    exec("/bin/ls");
  } else {
    /* in parent */
    wait();
  }

}
```

process PID = 10 | process PID = 11 | process PID = 11

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

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

```
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
- child return value is 0.

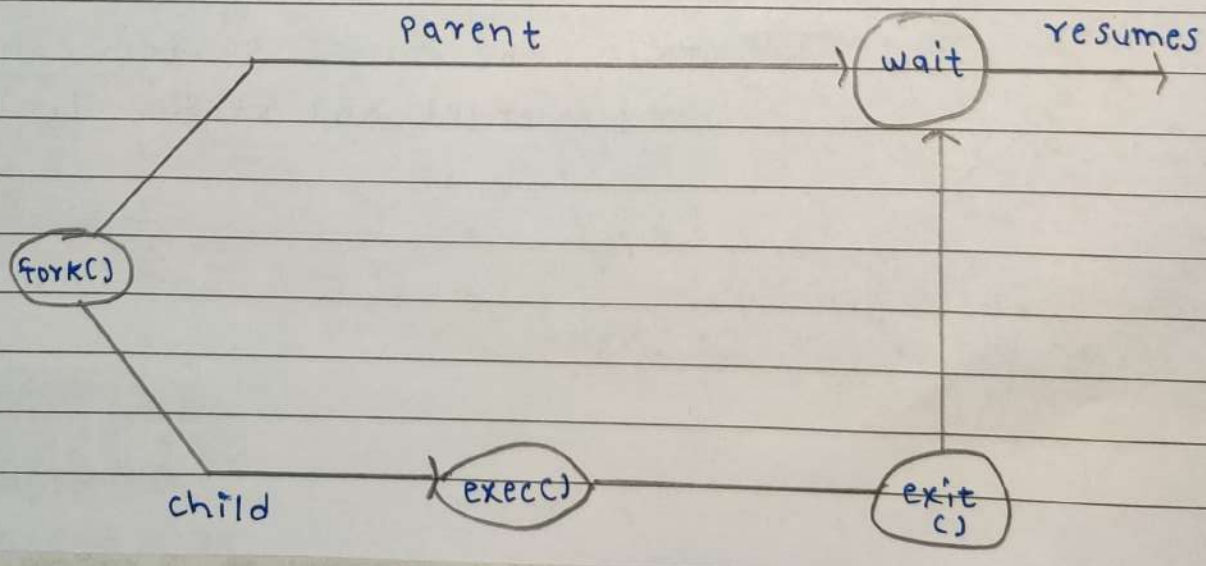
- int kill (pid_t, pid, int sig)
  - send a signal to a process
  - send SIGKILL to force termination.

- wait() -

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

child process may terminate due to any of these:

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



```
# include <sys/types.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.

EXPT 12 OUTPUT

```
parent process created
child created succesfull
child process id : 0
child after sleep
  PID TTY          TIME CMD
 4244 pts/1    00:00:00 bash
 4265 pts/1    00:00:00 a.out
 4266 pts/1    00:00:00 ps
parent still executingstatus : 0
p_status :4266
parent after sleep
parent process id : 4244
parent terminating
```

```c
/*SPOS expt12 */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>


int main()
{
    pid_t pid , ppid , p_status ;
    int status ;
    printf("parent process created \n");
    pid = fork();
    if(pid ==0)
    {
        printf("child created succesfull\n");
        printf("child process id : %d \n", pid);
        sleep(10);
        printf("child after sleep \n");
        execlp("/bin/ps","ps",NULL);

        printf("child terminating\n");
        exit(0);
    }
    else
    {
        printf("parent still executing");
        p_status = wait(&status);
        printf("status : %d \n",status);
        printf("p_status :%d \n",p_status);
        sleep(10);
        printf("parent after sleep\n");
        ppid = getppid();
        printf("parent process id : %d\n",ppid);
        printf("parent terminating\n");
        exit(0);
    }

    return 0;
}
```

Name- Tejas Vijay Khairnar
Roll No- 13 T.E
Sub- SPOS

Assignment No- 4

Title- Study assignment on process scheduling algorithms 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 algorithms in android & 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 requirement-

- M/C Lenovo think center M700 Ci3, ~~600~~ 6100, 6th gen. 4GB RAM, 500 GB HDD.

Theory concepts :

① Android is a mobile operating system developed by google, based on a modified version of the linux kernel and other open source software & designed primarily for touchscreen mobile devices such as smarthphones 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. varients of android are also used on game consoles, digital cameras, PCs and other electronics.

② Intially developed by android INC., which ~~google~~ google bought in 2005, Android was unvelid in 2007, with the first commerical android device launched in september 2008. The operating system has since gone through multiple major releases, with the current version being 8.1 "oreo" released in December 2017.

③ The android development supports with the full java programming language. even other Packages that are API and JSE are not supported. The first version 1.0 of android development kit (SDK) was released in 2008 and latest updated version is jelly bean.

some android version -

- Gingerbread (2.3)
- Honeycomb (3.0)
- Ice Cream sandwich (4.0)
- Jelly Bean (4.3 / 4.2 / 4.1)

Name - Tejas vijay khairnar
Roll No - 13 T.E
Sub - SPOS

CLASSMATE
Date :
Page : 45

Group - D

Assignment - 1

Title- write a java program [using oop features] to implement paging simultion using
① least recently used [LRU]
② optimal algorithm.

objectives -
- To understand concept of paging
- To learn different page replacement algorithms.

problem statement -
write a program to implement paging ~~simultion~~ simulation.

outcomes -
After completion of this assignment students will be able to:
- understand the programming language of Java
- understand the concept of paging

software requirements -
Linux OS, JDK 1.7

Hardware requirements
4 GB RAM, 500 GB HDD

- Kit Kat (4.4)
- Lollipop (5.0)
- Marshmallow (6.0)
- Nougat (7.0)
- oreo (8.0)

Advantages -

① support 2D & 3D graphics
② support multiple language
③ Java support
④ faster web browser
⑤ support audio, video etc

Disadvantages -

① slow response
② Heat
③ Advertisement etc

Theory concepts -

Paging - Paging is a memory-management scheme that permits the physical-address space of a process to be non contiguous.

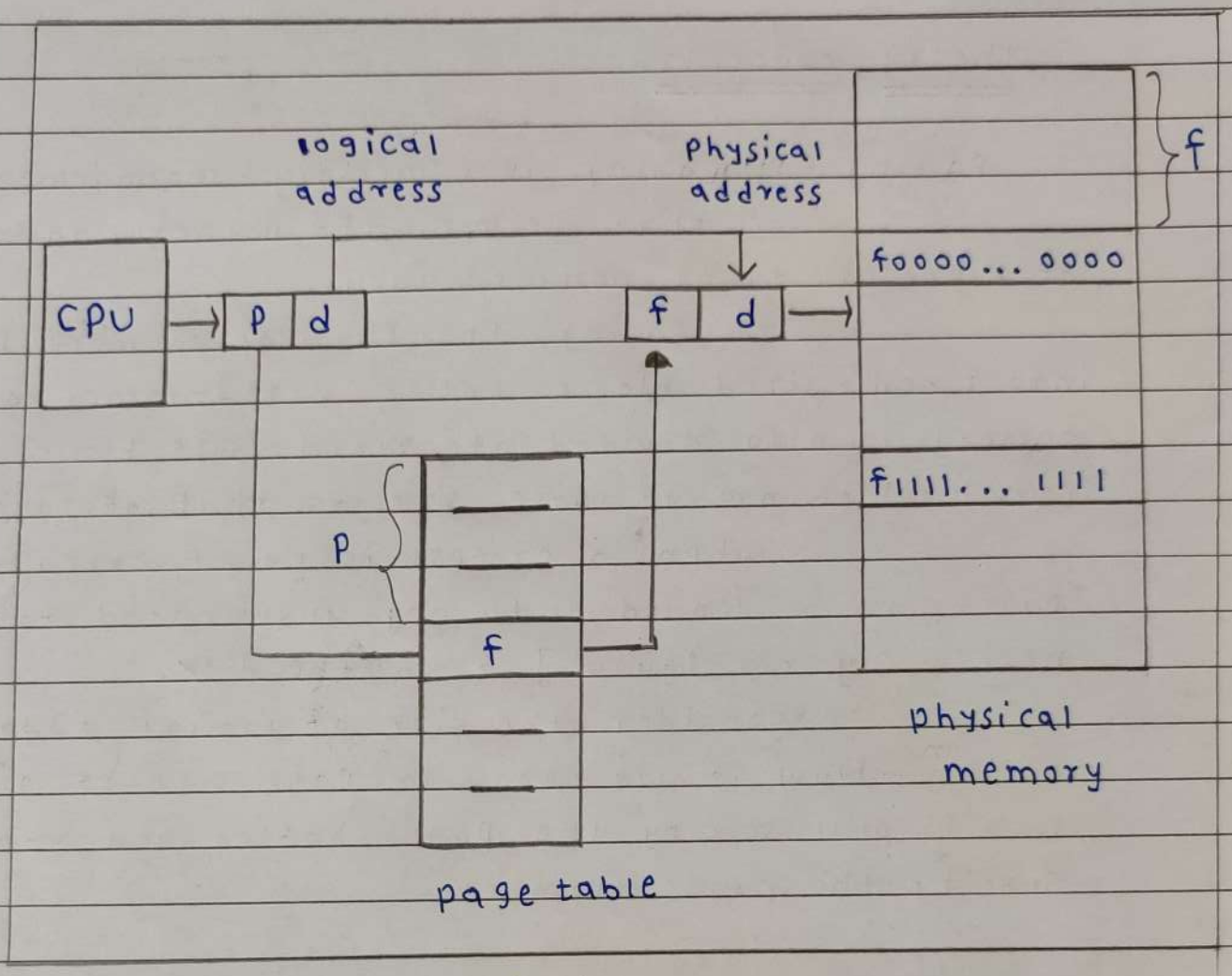
In paging, the Physical memory is divided into fixed-sized blocks called page frames and logical memory is also divided into fixed-size blocks called pages which are of same size as that of pages frames.

when a process is to be executed, its pages can be loaded into any unallocated frames [not necessarily contiguous] 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 pages offset and m-n bits will specify the pages 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 pages table.



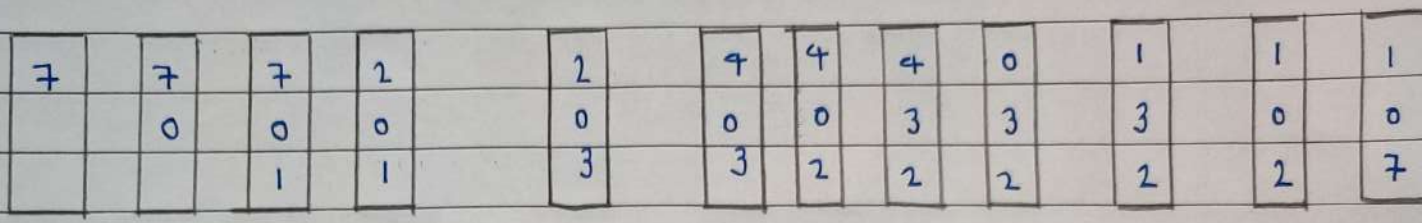
Paging.

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 1



page frames.

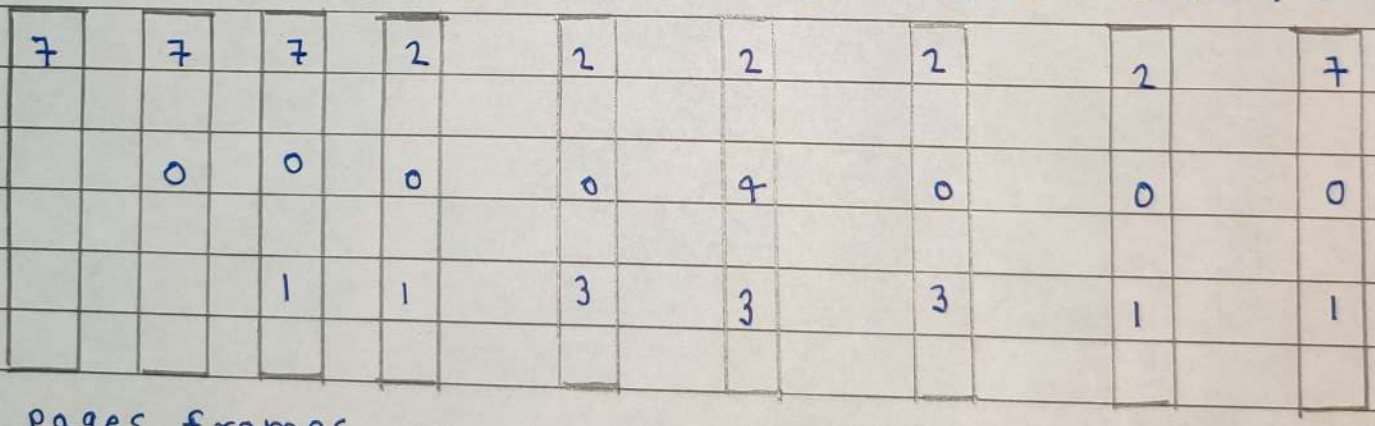
Example of LRU

optimal-

optimal page replacement refers to the removal of the page that will not be used in the future, for 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



pages frames.

conclusion- The various memory management pages replacement algorithms were studied and successfully implemented.

```java
// SPOS expt 14 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.readLine());

          int pg=0;
          for(pg=0;pg<n;pg++)
              {
              page=pages[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];

                  for(i=0;i<f;i++)
                      if(temp==frame[i])
                          break;
                  frame[i]=pages[pg];
              }
              else
              {
                  if(frame[0]==-1)
                      frame[0]=pages[pg];
                  else if(frame[1]==-1)
                      frame[1]=pages[pg];
                  else if(frame[2]==-1)
                      frame[2]=pages[pg];
              }


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

          }//for

          System.out.println("Page fault:"+pgf);

}//main
}//class
```

LRU OUTPUT

```
enter no. of frames:
4
enter the no of pages
10
enter the page no
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
frame :5   1   2   -1
Page fault:9
```

```java
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);



    }


}
```

OUTPUT OF OPTIMAL ALGO

```
Enter number of Frames:
3
Enter number of Pages:
13
Enter the pages:
1
7
8
3
0
2
0
3
5
4
0
6
1
        1       -1      -1      1       7       -1      1       7       8       1
3       8       1       3       0       2       3
0       2       3       0       2       3       0       5       3       0       4
3       0       4       3       0       6       3
0       1       3       0

Total Page Faults: 10
```