Sinhgad Institutes

SINHGAD COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

SUBJECT CODE: 310257

LAB MANUAL

SYSTEM PROGRAMMING AND OPERATING SYSTEM LABORATORY

Semester - II

Academic Year : 2017-18

Compiled By:
Prof. C.A.Laulkar
Prof. H.A.Bhute
Prof. D.N.Patil
Prof. G.G.Chiddarwar
Prof. J.B.Kulkarni

Sinhgad College of Engineering
Department of Computer Engineering

310257: System Programming & Operating System Lab

Teaching Scheme:
Practical: 4 Hrs/Week

Examination Scheme:
Term work : 25 Marks
Practical : 50 Marks

List of Laboratory Assignments

Sr. No.	Group A	Page No.
1	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.	4
2	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.	7
3	Design suitable data structures and implement pass-I of a two-pass macro-processor using OOP features in Java	10
4	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.	12
	Group B	
1	Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).	15
2	Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of subset of 'Java' program.	18
3	Write a program using Lex specifications to implement lexical analysis phase of compiler to count no. of words, lines and characters of given input file.	20
4	Write a program using YACC specifications to implement syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.	21
5	Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file.	23
	Group C	
1	Write a Java program (using OOP features) to implement following scheduling algorithms: FCFS , SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive)	26
2	Write a Java program to implement Banker's Algorithm	28
3	Implement UNIX system calls like ps, fork, join, exec family, and wait for process management (use shell script/ Java/ C programming).	30
4	Study assignment on process scheduling algorithms in Android and Tizen.	31
	Group D	
1	Write a Java Program (using OOP features) to implement paging simulation using 1. Least Recently Used (LRU) 2. Optimal algorithm	34

GROUP – A

GROUP: A, ASSIGNMENT -1

Title: Implementation of Pass-1 of two pass assembler.

Objectives:

1. To study the design and implementation of 1st pass of two pass assembler.
2. To study the categorized instruction set of assembler.
3. To study the data structure used in assembler implementation.

Problem Statement:

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

Theory:

1. Explain various Data and Instruction formats of IBM-360/370.
2. Explain the design of Pass- I of assembler with the help of flowchart and example.
3. Discuss various Data structure used in Pass-I along with its format and significance of each field.
4. Discuss the various classes of java used during implementation.

Algorithm:

Flowchart:

Design:

1. Class Diagram
2. Use case Diagram
3. ER Diagram

Input: Source code of Assembly Language

```
SAMPLE    START     100
          USING     *, 15
          L         1, FOUR
          A         1, =F'3'
          ST        1, RESULT
          SR        1, 2
          LTORG
          L         2, FIVE
          A         2, =F'5'
          A         2, =F'7'
FIVE      DC        F'5'
FOUR      DC        F'4'
RESULT    DS        1F
          END
```

Output:

100	SAMPLE	START	100
100		USING	*, 15
100		L	1, FOUR
104		A	1, =F’3’
108		ST	1, RESULT
112		SR	1, 2
114		LTORG	
124		L	2, FIVE
128		A	2, =F’5’
132		A	2, =F’7’
136	FIVE	DC	F’5’
140	FOUR	DC	F’4’
144	RESULT	DS	1F
152		5	
156		7	
160		END	

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	PSTART
USING	PUSING
DC	PDC
DS	PDS
LTORG	PLTORG
END	PEND

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation
1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	__	4	R

Literal Table (LT)

Sr. No	Literal	Address	Length
1	3	120	4
2	5	152	4
3	7	156	4

Test cases:

1. Check syntax of instruction (Correct and wrong)
2. Symbol not found
3. Wrong instruction
4. Duplicate symbol declaration
5. Test the output of program by changing value of START pseudo opcode.
6. Test the output of program by changing position of LTORG pseudo-op.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. System Programming by John Donovan, TATA McGraw-Hill edition
2. Java: The Complete Reference by Herbert Schildt, 9th Edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: A, ASSIGNMENT -2

Title: Implementation of Pass-2 of two pass assembler.

Objectives:

1. To study the design and implementation of 2nd pass of two pass assembler.
2. To study the data structure used in Pass-2 of assembler implementation.

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.

Theory:

1. Explain the design of Pass- II of assembler with the help of flowchart and example.

Algorithm:

Flowchart:

Design:

1. Class Diagram
2. Use case Diagram
3. ER Diagram

Input: Intermediate code of pass-1.

LC	LABEL	INSTR.	OPERANDS
100	SAMPLE	START	100
100		USING	*, 15
100		L	1, FOUR
104		A	1, =F'3'
108		ST	1, RESULT
112		SR	1, 2
114		LTORG	
124		L	2, FIVE
128		A	2, =F'5'
132		A	2, =F'7'
136	FIVE	DC	F'5'
140	FOUR	DC	F'4'
144	RESULT	DS	1F
152		5	
156		7	
160		END	

Output: Object Code

LC	OPCODE	OPERAND
100	5A	1,40(0,15)
104	1B	1,20(0,15)
108	50	1,44(0,15)
112	18	1,2
124	5A	2,36(0,15)
128	1B	2,52(0,15)
132	1B	2,56(0,15)

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	PSTART
USING	PUSING
DC	PDC
DS	PDS
LTORG	PLTORG
END	PEND

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation
1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	__	4	R

Literal Table (LT)

Sr. No	Literal	Address	Length
1	3	120	4
2	5	152	4
3	7	156	4

Base Table (BT)

Register no	Availability	Value/ Contents
1	N	--
:	:	:
:	:	:
:	:	:
15	Y	100

Test cases:

1. Check syntax of instruction (Correct and wrong)
2. Symbol not found
3. Wrong instruction
4. Duplicate symbol declaration
5. Test the output of program by changing value of START & USING pseudo opcode.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. System Programming by John Donovan, TATA McGraw-Hill edition
2. Java: The Complete Reference by Herbert Schildt, 9th Edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: A, ASSIGNMENT -3

Title: Implementation of pass-1 of Two Pass Macro Processor

Objectives:

1. To study the data structure used in macro-processor implementation
2. To study design and implementation of two pass microprocessor.

Problem Statement:

Design suitable data structures and implement pass-I of a two-pass macro-processor.

Theory:

1. What is macro processor?
2. Differentiate Macro and Function?
3. Explain the design of Pass- I of macro-processor with the help of flowchart?
4. Explain the design of Data structure used in Pass-I?
5. Explain the data structures used in Pass-I?

Algorithm:

Flowchart:

Design:

1. Class diagram
2. Sequence Diagram

Input: Small assembly language program with macros written in file input.asm.

```
        MACRO
&lab    ADDS    &arg1,&arg2
&lab    L       1, &arg1
        A       1, &arg2
        MEND
PROG    START   0
        BALR    15,0
        USING   *,15
LAB     ADDS    DATA1, DATA2
        ST      4,1
DATA1   DC      F'3'
DATA2   DC      F'4'
        END
```

Output: Assembly language program without macro definition but with macro call.

Note: Follow the following templates during implementation

Macro Name Table (MNT) :

Index	Macro Name	MDT Index
1	ADDS	15

Macro Definition Table (MDT) :

Index	Macro Definition Card entry
15	&lab ADDS &arg1, &arg2
16	#0 L 1, #1
17	A 1, #2
18	MEND

Argument List Array (ALA) :

Index	Arguments
0	&lab
1	&agr1
2	&arg2

Test cases:

1. Check macro end not found.
2. Duplicate macro name found.
3. Check program output by changing macro name and parameter list.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. System Programming by John Donovan, TATA McGraw-Hill edition
2. Java: The Complete Reference by Herbert Schildt, 9th Edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: A, ASSIGNMENT -4

Title: Implementation of pass-2 of Two Pass Macro Processor

Objectives:

1. To study design and implementation of pass-2 of two pass microprocessor.

Problem Statement:

Design suitable data structures and implement pass-I of a two-pass macro-processor.

Theory:

1. Explain design steps of two pass microprocessor, types of statements, data structures required and flowcharts.

Algorithm:

Flowchart:

Design:

1. Class diagram
2. Sequence Diagram

Input: Output of pass-1 (Intermediate File) given as a input to pass-2.

```
PROG        START   0
            BALR    15,0
            USING   *,15
LAB         ADDS    DATA1, DATA2
            ST      4,1
DATA1       DC      F'3'
DATA2       DC      F'4'
            END
```

Output: Assembly language program without macro definition and macro call.

```
PROG        START   0
            BALR    15,0
            USING   *,15
LAB         L       1, DATA1
            A       1, DATA2
            ST      4,1
DATA1       DC      F'3'
DATA2       DC      F'4'
            END
```

Macro Name Table (MNT):

Index	Macro Definition Card entry		
15	&lab	ADDS	&arg1, &arg2
16	#0	L	1, #1
17		A	1, #2
18		MEND	

Macro Definition Table (MDT):

Index	Macro Name	MDT Index
1	ADDS	15

Argument List Array (ALA) :

Index	Arguments
0	LAB
1	DATA2
2	DATA3

Test cases:

1. Check macro definition not found.
2. Check program output by changing parameter list in macro call.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

3. System Programming by John Donovan, TATA McGraw-Hill edition
4. Java: The Complete Reference by Herbert Schildt, 9th Edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP – B

GROUP: B, ASSIGNMENT - 1

Title: Implementation of Dynamic Link Library

Objectives:

1. To study and understand concept of DLL.
2. To understand JNI
3. To be able to create and use DLL.

Problem Statement:

Write a program to create Dynamic Link Library in c/c++ for mathematical operations (add, sub, div, mul, sin, cos, log, fact, etc.) and write a menu driven Java application program to test it with the help of Java Native Interface (JNI).

Theory:

1. What is DLL? T are the types of DLL and explain them? (Hint: Static and dynamic)
2. Explain Need of DLL?
3. Explain advantages of DLL?
4. What is java native interface (JNI)? (with diagram)
5. What is shared object (.so)?
6. What is the meaning of static block in java program?
7. What is native method in java?

Algorithm:

1. **Write a Java Class that uses C Codes - TestJNI.java**

```
public class TestJNI {
    static {
                    System.loadLibrary(“cal");  // Load native library at runtime
                      // cal.dll (Windows) or libcal.so (Unix)
            }
// Declare a native method add() that receives nothing and returns void private native int add
(int n1,int n2);
 // Test Driver
public static void main(String[] args) {
// invoke the native method
System.out.println(“Addition is=”+new TestJNI().add(10,20);
}
}
```

Compile Java code:

```
javac TestJNI.java
```

2. **Create the C/C++ Header file - TestJNI.h**

javah -jni TestJNI

3. **C Implementation - TestJNI.c**

```
#include <jni.h>
#include <stdio.h>
#include "TestJNI.h"
// Implementation of native method add() of TestJNI class
JNIEXPORT jint JNICALL Java_TestJNI_add(JNIEnv *env, jobject thisObj,jint n1,jint n2)
        {
          jint res;
          res=n1+n2;
          return res;
        }
```

Compile c-program:

$gcc -I /usr/local/jdk1.8.0_91/include /usr/local/jdk1.8.0_91/include/linux -o libcal.so –shared TestJNI.c

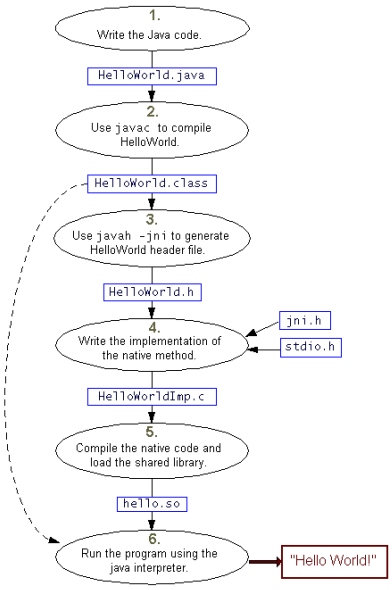
4. **Run the Java Program**

$java -Djava.library.path=. TestJNI

Addition is=30

5. **Repeat step 1-4 for all mathematical operations mentioned in problem statement.**

Flowchart:



Design:

1. Use case Diagram
2. Sequence Diagram

Input:

1. n1=10, n2=20

Output: Addition=30

Test cases:

1. Argument not found
2. Divide by zero for division operation

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: B, ASSIGNMENT – 2

Title: Lexical analyzer for subset of 'Java' program tokenization using LEX.

Objectives:

1. To understand working of LEX and lexical analyzer.
2. To understand token generation.
3. To understand file handling with command line arguments using LEX.

Problem Statement:

Implement lexical analysis phase of compiler to generate tokens of subset of 'Java' program.

Theory:

1. What is compiler ?
2. Phases of compiler.
3. Write short note on LEX.
4. Write and explain the rules used to generate tokens of JAVA program.

Algorithm:

Design:

1. Use case diagram
2. Transition diagram
3. Sequence diagram

Input: Sample input file containing java-language program (ip.java)

Output: Tokenization and lexemes (write output in output file)

Test cases:

1. Check regular expressions for every lexeme in input
2. Once regular expression (pattern) is matched, write it's token name to output file.
3. Can read input from multiple files using command line arguments.

Software requirements:

1. Fedora
2. LEX

Hardware requirements:

Conclusion:

References:

1. LEX & YACC by O'Reilly, 2nd edition.

Assessment Grade:

Signature of Subject Teacher:

Date of Completion:

GROUP: B, ASSIGNMENT - 3

Title: Implementation of lexical analysis phase of compiler to count no. of words, lines and characters of given input file.

Objectives:

1. To understand working of LEX.
2. To understand file handling with LEX.

Problem Statement:

Write a program using Lex specifications to implement lexical analysis phase of compiler to count no. of words, lines and characters of given input file.

Theory:

1. Write rules and action for counting words lines and characters.
2. Discuss various inbuilt variable of LEX.

Algorithm:

Flowchart:

Design:

1. Use case diagram
2. Transition diagram
3. Sequence diagram

Input:

1. Text file containing set of statements.

Output:

1. Total no. of words, lines and characters. (Write output in output file.)

Test cases: <Include successful and failure test cases>

Software requirements:

1. Fedora
2. LEX

Hardware requirements:

Conclusion:

1. **References:** LEX & YACC by O'Reilly, 2nd edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: B, ASSIGNMENT - 4

Title: Implementation of syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.

Objectives:

1. To understand working of YACC to recognize sentences.
2. To understand how LEX and YACC work together.
3. To understand implementation of grammar for recognition of variable declaration in Java.

Problem Statement:

Write a program using YACC specifications to implement syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.

Theory:

3. Different types of variable declaration.
4. Working of Syntax Analyzer.
5. Write short note on YACC.
6. Discuss the working of LEX and YACC program together to recognize sentence.

Algorithm:

Flowchart:

Design:

1. Use case Diagram
2. Sequence Diagram
3. Finite state diagram for rules designed to recognize tokens.

Input:

1. int a, b, c;

   int a, b=0;
   char c='z';
   float d=7.8;

Output:
Every declaration statement is parsed successfully, Otherwise display Error message.

Test cases:

1. Regular expressions should be matched for every lexeme in input which is a declaration statement. If not, test case fails.
2. The lexemes are keywords, operators, special symbols, numbers which are the parts of declaration statements.
3. Once regular expression (pattern) is matched, it's token name are returned to Yacc program.
4. For all tokens returned from Lex program, grammar is checked for all possible forms of declaration statements.
5. At the end, the output is displayed that declaration statement parsed is correct, else error message is displayed.

Software Requirement:

1. Fedora
2. LEX

Hardware Requirement:

Conclusion:

References:

1. LEX & YACC by O'Reilly, 2nd edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: B, ASSIGNMENT - 5

Title: Implementation of syntax analysis phase of compiler to recognize simple and compound sentences.

Objectives:

1. To understand working of YACC to recognize sentences.
2. To understand how LEX and YACC work together.
3. To understand implementation of grammar for sentence recognition.

Problem Statement:

Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file.

Theory:

1. Write short note on "Sentences in English".
2. Write and discuss rules used to generate tokens of sentence.
3. Write and explain grammar to recognize the sentence.
4. Discuss the working of LEX and YACC program together to recognize sentence.

Algorithm:

Flowchart:

Design:

1. Use case Diagram
2. Sequence Diagram
3. Finite state diagram for rules designed to recognize tokens.

Input:

1. List of verbs, nouns, pronouns…etc.
   a. NOUN Ana Fred Ram boy
   b. VERB is was go
2. Actual English sentence.

Output:

Each word with its type will be displayed if found in symbol table, Otherwise display error message.

e.g.

1) Ram is boy

output:

Ram is noun

is is verb

boy is noun

Simple statement

2) Ram is boy and he is intelligent

Ram is noun

is is verb

boy is noun

he is pronoun

is verb

and is conjunction

intelligent is adjective

compound statement

Test cases:

1. Regular expressions should be matched for every lexeme in input. If not, test case fails.
2. All lexeme should be English words.
3. Once regular expression (pattern) is matched, it's token name like noun, pronoun, verb, adjective, etc is returned to yacc program.
4. The entry of all words should be done in symbol table.
5. For all tokens returned from lex program, grammar is checked.
6. Test the program for simple sentence.
7. Test the program for one or multiple compound sentence.
8. Test the program for word which is not present in input list.

Software requirements:

1. Fedora
2. LEX
3. YACC

Hardware requirements:

Conclusion:

References:

1. LEX & YACC by O'Reilly, 2nd edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP – C

GROUP: C, ASSIGNMENT -1

Title: Implementation of CPU scheduling algorithms.

Objectives:

1. To study the process management and various scheduling policies viz.Preemptive and Non preemptive.
2. To study and analyze different scheduling algorithms.

Problem Statement:

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

Theory:

1. Define process. Explain need of process scheduling.
2. Explain different scheduling criteria and policies for scheduling processes.
3. Explain possible process states
4. Explain FCFS, SJF(Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive) and determine waiting time, turnaround time, throughput using each algorithm.

Algorithm

Flowchart:

Design:

1. Class Diagram
2. Use case Diagram
3. ER Diagram

Input:

1. Enter the number of processes:
2. Enter burst time and arrival time of each process

Output:

1. Compute Waiting time, turnaround time, average waiting time, average turnaround time and throughput.

For each algorithm display result as follows:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P1				
P2				
P3				
-				

Calculate

1. Average waiting time=
2. Average turnaround time=
3. Throughput=

Test cases:

1. Check arrival time of all process should not be same.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. Silberschatz, Galvin, Gagne, "Operating System Principles", 9th Edition, Wiley, ISBN 978-1-118-06333-0
2. Stallings W., "Operating Systems", 6th Edition, Prentice Hall, ISBN-978-81-317-2528-3.
3. Java: The Complete Reference by Herbert Schildt, 9th Edition.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: C, ASSIGNMENT - 2

Title: Banker's Algorithm

Objectives:

1. To understand safe and unsafe state to handle deadlock situation in the system.
2. To handle deadlock condition.
3. To implement banker's algorithm to avoid deadlock.

Problem Statement:

Write a Java program to implement Banker's Algorithm

Theory:

1. Write short note on Deadlock
2. Conditions for Deadlock
3. Discuss Banker's Algorithm for a Single Resource and Multiple Resources with example.
4. Dealing with deadlock

Algorithm: Banker's Algorithm

Flowchart:

Design:

1. E-R Diagram
2. Class Diagram
3. State Diagram

Input:

1. Accept claim and allocation matrix, available resource vector.
2. New request

Output:

1. Display need matrix.
2. Display safe sequence if safe else unsafe.

Test cases:

1. Test the program for safe state.
2. Test the program for unsafe state

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. Operating Systems: Internals and Design Principles, 6/E William Stallings. [Chapter 6]
2. Operating System concepts, Silberschatz, Galvin & Gagne, 7/E.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: C, ASSIGNMENT -3

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

Objectives:

1. To understand Unix system calls
2. To understand working and purpose of system calls

Problem Statement:

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

Theory:

1. What is System call? Explain the working of system call.
2. Explain User mode and kernel mode.
3. Explain the functions with syntax : fork, vfork, exec family, ps, join, kill, wait, waitpid, getpid ,getppid, setpid, Exit, nice.

Input: Unix commands discussed above.

Output: Output of Unix command discussed above.

Test cases: Test the commands with various options.

Software requirements:

1. Fedora
2. Shell script/ C or C++ /Java

Conclusion:

References:

1. Unix shell programming by Sumitabha Das, 4th edition, Mc.Graw Hill

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP: C, ASSIGNMENT -4

Title: Study assignment on process scheduling algorithms in Android and Tizen.

Objectives:

1. To understand purpose of Light weight operating system.
2. To understand process scheduling in Android and Tizen

Problem Statement:

To study process scheduling algorithms in Android and Tizen

Theory:

1. Android Framework/ Architecture, Feature, Advantages and Disadvantages.
2. Tizen Operating system, its framework/ Architecture, Features, Advantages and Disadvantages.

.

Algorithm: Process Scheduling algorithm in Android and Tizen

Flowchart:

Software requirements:

1. Fedora
2. Shell script/ C or C++ /Java

Hardware requirements:

Conclusion:

References:

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion:

GROUP - D

GROUP: D, ASSIGNMENT - 1

Title: Simulation of paging

Objectives:

1. To study page replacement policies to understand memory management.
2. To understand efficient frame management using replacement policies.

Problem Statement:

Write a Java Program (using OOP features) to implement paging simulation using
1. Least Recently Used (LRU)
2. Optimal algorithm

Theory:

1. Explain all page replacement policies.
2. Explain concept of frames, pages, page hit & page miss ratio.
3. Give one example.

Algorithm:

Flowchart:

Design:

1. Class Diagram

Input:

1. Number of frames.
2. Number of pages.
3. Page sequence.

Output:

1. Cache hit and cache miss ratio.

Test cases:

1. Test the page hit and miss ratio for different size of page frames.
2. Test the page hit and miss ratio for both algorithms with different page sequences.

Software requirements:

1. Fedora
2. Eclipse
3. JDK

Hardware requirements:

Conclusion:

References:

1. Operating Systems: Internals and Design Principles, 6/E William Stallings.
2. Operating System concepts, Silberschatz, Galvin & Gagne, 7/E.

Assessment Grade: **Signature of Subject Teacher:**

Date of Completion: