# *Lab Manual For Programming Lab –III*

**T. E.**

**Dept Of Computer Engineering**

**2014-15**

MIT ACADEMY OF ENGINEERING, ALANDI (D)

Department of Computer Engineering

Lab Manual For Programming Lab -III

WE LEARN MOST

WHEN WE HAVE TO INVENT…

- PIAGET

# List of Assignments

| S N | Title of Assignments |
|---|---|
| 1 | Develop an application using Beeglebone Black/ARM Cortex A5 development board to simulate the operations of LIFT. |
| 2 | Develop an application using Beeglebone Black/ARM Cortex A5 development board to simulate the working of signal lights. |
| 3 | Implement an calculator (Binary Multiplication) application using concurrent lisp |
| 4 | Apply the Following Software Engineering to all assignments (No 1, 2, 3 of Group A and B). Mathematical Modeling must result into UML Requirements. Apply Assignment No 4a to 4d for all Group A and Group B assignments of Embedded Operating system and Concurrent and Distributed Programming. Use tools Open source tools like ArgoUML, UMLLet, StarUML or equivalent tools for UML models) |
| 4a | Design mathematical model of the Application/system using set theory, algebraic system, relations and functions, Deterministic and Non-Deterministic entities. |
| 4b | Analyze requirements from the Problem statement, mathematical model, Domain requirements and identify Functional, Non functional, Actors, Usecases for the Application/system. Create usecase diagram, activity diagram/swimlane diagram for each usecase. |
| 4c | Design the architecture for the system/application using package diagram, deployment diagram. Design classes using class diagram. |
| 4d | Design the behavior of the system/application using state machine diagram and sequence diagram. |
| 5 | Create Project plan, SRS, Design document and Test Plan for one group-C assignment from embedded operating system or Concurrent and Distributed Programming. |
| 6 | Write an application to parse input text file concurrently and compare the result of concurrent parsing with serial parsing ( Use concurrent YACC parser) |
| 7 | Vedic Mathematics method to find square of 2-digit number is used in a distributed programming.Use shared memory and distributed (multi-CPU) programming to complete the task. |
| 8 | Implement n-ary search algorithm using OPENMP |
| 9 | Implement concurrent prims algorithm using OPENMP |
| 10 | Develop a network based application by setting IP address on BeagleBoard/ ARM Cortex A5. |
| 11 | Implement a Multi-threading application for echo server using socket programming in JAVA. |
| 12 | Implement Reader-Writer problem using OPENMP. |
| 13 | Perform Assignment No 4 of Group A for Assignment No 12 of Group-B using UMLLet |
| 14 | Perform Assignment No 4 of Group A for Assignment No 13 of Group-B using concurrent UML. |
| 15 | Develop Robotics (stepper motor) Application using Beagle Board. |

# Assignment No 3

**Title:** Implement an calculator (Binary Multiplication) application using concurrent lisp

**Requirements:** Any Linux Distro. with sbcl lisp compiler

## Theory:

Lisp (historically, LISP) is a family of computer programming languages with a long history and a distinctive, fully parenthesized Polish prefix notation. Originally specified in 1958, Lisp is the second-oldest high-level programming language in widespread use today; only Fortran is older (by one year).Like Fortran, Lisp has changed a great deal since its early days, and a number of dialects have existed over its history. Today, the most widely known general-purpose Lisp dialects are Clojure, Common Lisp and Scheme.

Lisp was originally created as a practical mathematical notation for computer programs, influenced by the notation of Alonzo Church's lambda calculus. It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, Lisp pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, and the self-hosting compiler.

The name LISP derives from "LISt Processing". Linked lists are one of Lisp language's major data structures, and Lisp source code is itself made up of lists. As a result, Lisp programs can manipulate source code as a data structure, giving rise to the macro systems that allow programmers to create new syntax or new domain-specific languages embedded in Lisp.

Traditionally Lisp implements multiprogramming by emulating multi-tasking. When it comes down to it, only one task is being performed at a time. Forthcoming releases of both ACL and LW will support symmetric multiprocessing: concurrent threads running in parallel on multiple cores. This will allow for considerable performance gains but will have implications for the enforcement of atomicity. Both implementations will introduce new operators for controlling concurrency as well as a number of primitives for atomic read-modify-write operations.

Concurrent LISP is a concurrent programming language based on LISP.  It is designed without changing the original language features of LISP. For example, literal atoms are unique, functional notation is preserved; dynamic binding strategy is used and so on.

In addition to the ordinary LISP functions, Concurrent LISP has three primitive concurrent functions and special functions to manipulate data attached to processes. The three primitive functions are STARTEVAL for process activation, CR (Critical Region function) and CCR (Conditional Critical Region function) for mutual exclusion. A multi-process system written in Concurrent LISP is a set of many cooperating sequential processes, each of which evaluates its given form.

### Features of Common LISP

- It is machine-independent
- It uses iterative design methodology, and easy extensibility.
- It allows updating the programs dynamically.
- It provides high level debugging.
- It provides advanced object-oriented programming.
- It provides convenient macro system.
- It provides wide-ranging data types like, objects, structures, lists, vectors, adjustable arrays, hash-tables, and symbols.
- It is expression-based.
- It provides an object-oriented condition system.
- It provides complete I/O library.
- It provides extensive control structures.

### Applications Built in LISP

Large successful applications built in Lisp.

- Emacs
- G2
- AutoCad
- Igor Engraver
- Yahoo Store

### LISP functions

This section describes a number of simple operations on lists, i.e., chains of cons cells.

- **cl-caddr** *x*

  This function is equivalent to (car (cdr (cdr*x*))). Likewise, this package defines all 24 c*xxx*r functions where *xxx* is up to four ‘a’s and/or ‘d’s. All of these functions are setf-able, and calls to them are expanded inline by the byte-compiler for maximum efficiency.

- **cl-first** *x*

  This function is a synonym for (car *x*). Likewise, the functions cl-second, cl-third, ..., through cl-tenth return the given element of the list *x*.

- **cl-rest** *x*

  This function is a synonym for (cdr*x*).

- **cl-endp** *x*

  Common Lisp defines this function to act like null, but signaling an error if x is neither a nil nor a cons cell. This package simply defines cl-endp as a synonym for null.

- **cl-list-length** *x*

  This function returns the length of list *x*, exactly like (length *x*), except that if *x* is a circular list (where the cdr-chain forms a loop rather than terminating with nil), this function returns nil. (The regular length function would get stuck if given a circular list. See also the safe-length function.)

- **cl-list***_arg&rest others_

  This function constructs a list of its arguments. The final argument becomes the cdr of the last cell constructed. Thus, (cl-list* *abc*) is equivalent to (cons *a* (cons *bc*)), and (cl-list* *ab* nil) is equivalent to (list *ab*).

- **cl-ldiff** *list sublist*

  If *sublist* is a sublist of *list*, i.e., is eq to one of the cons cells of *list*, then this function returns a copy of the part of *list* up to but not including *sublist*. For example, (cl-ldiff x (cddr x)) returns the first two elements of the list x. The result is a copy; the original *list* is not modified. If *sublist* is not a sublist of *list*, a copy of the entire *list* is returned.

- **cl-copy-list** *list*

  This function returns a copy of the list *list*. It copies dotted lists like (1 2 . 3) correctly.

- **cl-tree-equal** *x y* &key :test :test-not :key

  This function compares two trees of cons cells. If *x* and *y* are both cons cells, their cars and cdrs are compared recursively. If neither *x* nor *y* is a cons cell, they are compared by eql, or according to the specified test. The :key function, if specified, is applied to the elements of both trees.

## Program:

```
; To install lisp compiler on linux you can this blog for more information:
; http://www.mohiji.org/2011/01/31/modern-common-lisp-on-linux/

(defvar a)
(defvar b)
(defvar c)
(defvar d)

(write-line " Enter two numbers : ")

        (setf a(read))
        (setf b(read))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                        (setf c(+ a b))
                                        (print "ADDITION : ")
                                        (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                        (setf c(- a b))
                                        (print "SUBTRACTION : ")
                                        (print c))))
        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(* a b))
                                              (print "MULTIPLICATION : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(* a a))
                                              (print "SQUARE OF 1st NUMBER  : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(* b b b))
                                              (print "CUBE OF 2ND NUMBER : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(sin a))
                                              (print "SINE OF 1ST NUMBER : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(tan a))
                                              (print "TAN OF 1ST NUMBER : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(cos a))
                                              (print "COSINE OF 1ST NUMBER : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(min a b))
                                              (print "MINIMUM NUMBER : ")
                                              (print c))))

        (sb-thread:make-thread(lambda()(progn(sleep 0)
                                              (setf c(max a b))
                                              (print "MAXIMUM NUMBER : ")
                                              (print c))))
(exit)
```

## Compiling mechanism:

Go to directory where your program resides and at that location type:

```
[abc@localhost ~]$ sbcl

This is SBCL 1.1.12-1.fc20, an implementation of ANSI Common Lisp.
More information about SBCL is available at <http://www.sbcl.org/>.
SBCL is free software, provided as is, with absolutely no warranty.
It is mostly in the public domain; some portions are provided under
BSD-style licenses.  See the CREDITS and COPYING files in the
distribution for more information.

* (load "cal111.lisp")
Enter two numbers:
9876543212345678
1234567898765432
```

## Output:

```
"ADDITION : "
11111111111111110
"SUBTRACTION : "
8641975313580246
"MULTIPLICATION : "
12193263200731593561957021002896
"SQUARE OF 1st NUMBER  : "
97546105825331484352994965279684
"CUBE OF 2ND NUMBER : "
1881676411868861777694985806514774434038701568
"SINE OF 1ST NUMBER : "
-0.20660499
"TAN OF 1ST NUMBER : "
0.21116091
"COSINE OF 1ST NUMBER : "
-0.97842443
"MINIMUM NUMBER : "
1234567898765432
"MAXIMUM NUMBER : "
9876543212345678
```

**Conclusion:** In this way, we have implemented the calculator application using concurrent lisp.

# Assignment No 7

**Title:** Vedic Mathematics method to find square of 2-digit number is used in a distributed programming. Use shared memory and distributed (multi-CPU) programming to complete the task.

**Requirements:** Any Linux Distro.with GCC 4.2 or above compiler

## Theory:

Vedic Mathematics is the name given to the ancient system of Indian Mathematics which was rediscovered from the Vedas between 1911 and 1918 by Sri BharatiKrsnaTirthaji (1884-1960). According to his research all of mathematics is based on sixteen Sutras, or word-formulae. For example, 'Vertically and Crosswise` is one of these Sutras. These formulae describe the way the mind naturally works and are therefore a great help in directing the student to the appropriate method of solution.

Perhaps the most striking feature of the Vedic system is its coherence. Instead of a hotch-potch of unrelated techniques the whole system is beautifully interrelated and unified: the general multiplication method, for example, is easily reversed to allow one-line divisions and the simple squaring method can be reversed to give one-line square roots. And these are all easily understood. This unifying quality is very satisfying; it makes mathematics easy and enjoyable and encourages innovation.

In the Vedic system 'difficult' problems or huge sums can often be solved immediately by the Vedic method. These striking and beautiful methods are just a part of a complete system of mathematics which is far more systematic than the modern 'system'. Vedic Mathematics manifests the coherent and unified structure of mathematics and the methods are complementary, direct and easy.

The simplicity of Vedic Mathematics means that calculations can be carried out mentally (though the methods can also be written down). There are many advantages in using a flexible, mental system. Pupils can invent their own methods, they are not limited to the one 'correct' method. This leads to more creative, interested and intelligent pupils.

Interest in the Vedic system is growing in education where mathematics teachers are looking for something better and finding the Vedic system is the answer. Research is being carried out in many areas including the effects of learning Vedic Maths on children; developing new, powerful but easy applications of the Vedic Sutras in geometry, calculus, computing etc.

Vedic Maths is methodology to solve complicated Mathematics through Easy theorems and corollaries known as **‘Sutras’**and **‘Upsutras’**.Some methods which are present in vedas to solve our most familiar mathematics calculations like Calculate factorial of 3 Digit number, Calculate Cube of 6 digit number which are time consuming and require calculators to solve problem.

In this practical, we are going to see two simple sutras:

1. **Ekadhiken-Purnen**

2. **Urdhva-tiryagbhyam**

Let’s see how we are able to achieve this:

Suppose I want to make square of 25. I will use the **Ekadhiken-Purnen.** In this case, the method works as follows:

1. Take a number at 10th place
2. Add +1 to 10th place digit
3. Now, multiply original no in 10th place and modified number,generated by adding +1 in original number.
4. After multiplying them, append square of number at Unit’s place.

For ex. $25^2$ = 2 * (2+1)/25 = 625

Second method is mainly used to make multiplications of two digit different numbers. We can also use it. The second method works as follows:

1. Multiply the digits at Unit’s place of 2 numbers.
2. Leaving the unit’s place, cross multiply and add together, producing the middle digit and put it before we have left place for Unit’s digit number.
3. For the 10th place digit, multiply 10th place digits of two numbers.
4. Put all of the digits together to produce your answer.

```
   25
X  25
-----------
  4 2 5
+ 2 0 x
-----------
  6 2 5
-----------
```

So considering easiness of the first method, we will use Ekadhiken-Purnen.This program is going to execute in following ways on multicores.

1. Multiplication of Unit place digits will occur on the 1st core.
2. Crosswise multiplication and then addition on 2nd core.
3. Multiplication of 10th place digit on 3rd core.
4. Addition of these two results on 4th core and print result on the same.

This program deals with calculating the operation of multiplying two digit numbers for multiple threads

In the same way,

$35^2$ = 3 x (3+1) /25 = 3 X 4/ 25 = 1225;

$65^2$ = 6 x 7 / 25 = 4225;

$105^2$ = 10 x 11/25 = 11025;

$135^2$ = 13 x 14/25 = 18225;

Apply the formula to find the squares of the numbers 15, 45, 85, 125, 175 and verify the answers.

Algebraic proof:

Consider (ax + b)2 = a

This identity for x = 10 and b = 5 becomes

$$(10a + 5)^2 = a^2 . 10^2 + 2.10a.5+5^2$$

$$= a^2.10^2 + a.10^2 + 5^2$$

$$= (a^2+a) .10^2+5^2$$

$$= a (a + 1) . 10^2 + 25.$$

Clearly 10a + 5 represents two-digit numbers 15, 25, 35, -------,95 for the values a = 1, 2, 3, --,9 respectively. In such a case the number (10a + 5)2 is of the form whose L.H.S is a (a + 1) and R.H.S is 25, that is, a (a + 1) / 25.

We are able to implement the distributed programming concept using the OpenMP sections directive which is able to make the changes according to different sections of the program. The SECTIONS clause in OpenMP is having capability to execute portion of program separately on separate processors.

Distributed systems are groups of networked computers, which have the same goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly coupled form of distributed computing and distributed computing may be seen as a loosely coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors may have access to a shared memory to exchange information between processors.
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.

High-performance parallel computation in a shared-memory multiprocessor uses parallel algorithms while the coordination of a large-scale distributed system uses distributed algorithms.

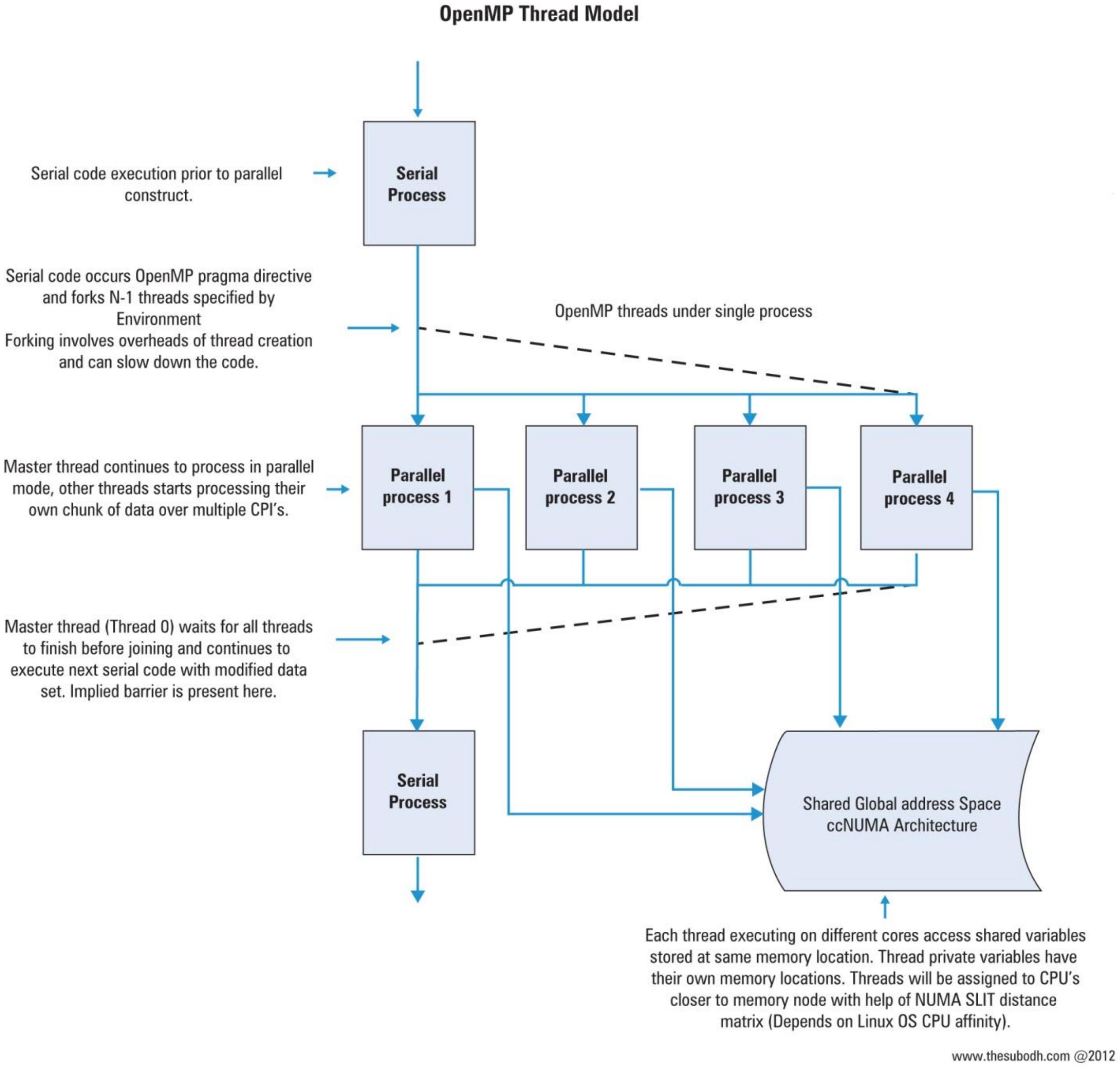

OpenMP Thread Model
Serial code execution prior to parallel construct.
Serial Process
Serial code occurs OpenMP pragma directive and forks N-1 threads specified by Environment
Forking involves overheads of thread creation and can slow down the code.
OpenMP threads under single process
Master thread continues to process in parallel mode, other threads starts processing their own chunk of data over multiple CPI's.
Parallel process 1
Parallel process 2
Parallel process 3
Parallel process 4
Master thread (Thread 0) waits for all threads to finish before joining and continues to execute next serial code with modified data set. Implied barrier is present here.
Serial Process
Shared Global address Space ccNUMA Architecture
Each thread executing on different cores access shared variables stored at same memory location. Thread private variables have their own memory locations. Threads will be assigned to CPU's closer to memory node with help of NUMA SLIT distance matrix (Depends on Linux OS CPU affinity).
www.thesubodh.com @2012

**Figure: Diagram of OpenMP processing**

## Program:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>

int main()
{  int a,b,c;     // defining the three variables

    int tid = omp_get_num_threads();

    #pragma omp parallel sections shared(a,b,c,tid)      // to execute the threads in parallel section

   {

    #pragma omp section                              // to execute threads as per the for loop defined

     for(tid=0;tid<6;tid++)

  {
    printf("Thread %d starting\n",tid);
    printf("Enter a as 10th place digit: ");
    printf("\n");
    scanf("%d",&a);

    printf("Thread %d starting\n",tid);
    printf("Enter b as Units place digit: ");
    printf("\n");
    scanf("%d",&b);

    printf("On thread %d => ",tid);

    c = a + 1;                                        // For 25, example c = 2 + 1 = 3
    a = c * a;                                         // a = 3 * 2 = 6
    b = b * b;                                         // b = 5 * 5 = 25

    printf("The result is %d%d\n",a,b);          // appending the values of the two digits

    }

    }

    return 0;

   }
```

## Compilation Mechanism:

Program can be compiled as:

```
$ gcc –fopenmp  vedic_maths.c

$ ./a.out
```

## Output:

```
Thread 0 starting
Enter a as 10th place digit:
5

Thread 0 starting
Enter b as Unit’s place digit:
5

On thread 0 => The result is 3025

Thread 1 starting
Enter a as 10th place digit:
6

Thread 1 starting
Enter b as Unit’s place digit:
5

On thread 1 => The result is 4225


Thread 2 starting
Enter a as 10th place digit:
7

Thread 2 starting
Enter b as Unit’s place digit:
5

On thread 2 => The result is 5625

Thread 3 starting
Enter a as 10th place digit:
8

Thread 3 starting
Enter b as Units place digit:
5
```

```
On thread 3 => The result is 7225

Thread 4 starting
Enter a as 10th place digit:
3

Thread 4 starting
Enter b as Units place digit:
5

On thread 4 => The result is 1225

Thread 5 starting
Enter a as 10th place digit:
2

Thread 5 starting
Enter b as Units place digit:
5

On thread 5 => The result is 625
```

**Conclusion:** In this way, we have implemented Vedic Mathematics method to find square of 2-digit number is used in a distributed programming.

# Assignment No 8

**Title:** Implement n-ary search algorithm using OPENMP.

**Requirements:** Any Linux Distros with GCC 4.2 or above compiler

## Theory:

A n-ary tree is simply is number of nodes associated with each parent either zero or n numbers depends on search criteria. It traverses according to parent to child fashion rather than each level of nodes present in n-ary tree. It makes searching time upto O ($\log_2 n$) . The height of a complete K-ary tree with n nodes is ceiling ($\mathbf{log_k n}$). In parallel programming, the simple thing is that we are comparing all calculated values of the each level at once so that we are able to find number of element we desire at particular location in very small amount time.

N-ary algorithm is extension of Binary Search algorithm. Searching a list for a particular item is a common task. In real applications, the list items often are records (e.g. student records), and the list is implemented as an array ofobjects. The goal is to find a particular record, identified by name or an ID number such as a student number. Finding the matching list element provides access to target information in the record the student's address, for example. The following discussion of search algorithms adopts a simpler model of the search problem the lists are just arrays ofintegers. Clearly, the search techniques could generalize to more realistic data. The concept of efficiency (or complexity) is important when comparing algorithms. For long lists and tasks, like searching, that are repeated frequently, the choice among alternative algorithms becomes important because they may differ in efficiency. To illustrate the concept of algorithm efficiency (or complexity), we consider two common algorithms for searching lists: linear search and binary search.

### Linear Search :

What is the simplest method for searching a list of elements? Check the first item, then the second item, and so on until you find the target item or reach the end of the list. That's **linear** (or **sequential**) search. Assume an array called list, filled with positive integers. The task is to search for the location of the number that is the value of a variable target. Here's how to search for the location of target in list:

```
int location = -1;
int i = 0;
while ((list[i] != target) && (i <list.length))
i++;
if (list[i] == target)
location = i;
```

A while loop is appropriate since the target could be anywhere in the array, or perhaps not in it at all. The program must be able to exit the search loop as soon the target is located, and to iterate over all the items in the array, if necessary.

### Binary search :

The strategy of binary search is to check the middle (approximately) item in the list. If it is not the target and the target is smaller than the middle item, the target must be in the first half of the list. If the target is larger than the middle item, the target must be in the last half of the list. Thus, one unsuccessful comparison reduces the number of items left to check by half!

The search continues by checking the middle item in the remaining half of the list. If it's not the target, the search narrows to the half of the remaining part of the list that the target could be in. The splitting process continues until the target is located or the remaining list consists of only one item. If that item is not the target, then it's not in the list.

Here are the main steps of the binary search algorithm, expressed in ***pseudocode*** (a mixture of English and programming language syntax).

Main **steps** of Binary Search

```
1. location = -1;
2. while ((more than one item in list) and (haven't yet found target))
     a)  look at the middle item
     b)  if (middle item is target)
             have found target
         else
     c)  if (target < middle item)
             list = first half of list
     d)  else (target > middle item)
             list = last half of list
   end while
3. if (have found target)
   location = position of target in original list
4. return location as the result.
```

It might appear that this algorithm changes the list. Each time through the while loop half of the list is "discarded", so that at the end the list may contain only one item. There are two reasons to avoid actually changing the list. First, it may be needed again, to search for other targets, for example. Second, it creates extra work to delete half of the items in a list, or to make a copy of a list. This could offset any gains in efficiency.

## Program:

```
/*
	N-ary search
	To compile:-
		gcc -fopenmp N-ary_search.c
*/
```

```
#include<stdlib.h>
#include<stdio.h>
#include<omp.h>
#define SIZE 100000000  //size of total number of nodes in N-ary tree
enum locate_t                  // define loacate_t for position of traverser in N-ary Search tree
{
        EQUAL,
        LEFT,
        RIGHT
};
#define N 250                  // no of nodes associated with each edge

nary_search(int *a, long int size, int key)
{
        long int mid[N+1];          // in the middle of list with N+1 elements
        int i;                      // temporary variable for shifting elements
        enum locate_t locate[N+2];
        locate[0] = RIGHT;          // locate[0] indicates to move traverser in right direction
        locate[N + 1] = LEFT;       // locate[N+1] iindicates to move traverser in leftward directions
        long int lo = 0;            // location of elements in lower level of N-ary tree
        long int hi = size - 1;     // location of elements in highest level of N-ary tree
        long int pos = -1;
        double step, offset;        // "step" for carryforwarding values and "offset" to define locally
                                    // for calculating / lower middle value for fast searching
        #pragma omp parallel
        {
                while(lo <= hi && pos == -1)
                {
                        #pragma omp single    // only this part of code is shared with one processor
                        {
                                mid[0] = lo - 1;
                                step = (hi - lo + 1) / (N + 1);
                        }
                        #pragma omp for private(offset) firstprivate(step)
                        for(i=1;i<=N;i++)
                        {
                                offset = step * i + (i - 1);
                                int lmid = mid[i] = lo + offset;

                               // To check whether the lower middle value is less or equal to that of hi
                                if(lmid <= hi)
                                {
                                        if(a[lmid] > key)
                                        {
                                                locate[i] = LEFT;
```

```
                    }
                    else if(a[lmid] < key)
                    {
                        locate[i] = RIGHT;
                    }
                    else
                    {
                        locate[i] = EQUAL;
                        pos = lmid;
                    }
                }
                else
                {
                    mid[i] = hi + 1;
                    locate[i] = LEFT;
                }
            }
            #pragma omp single
            {
                for(i=1;i<=N;i++)
                {
                    if(locate[i] != locate[i-1])
                    {
                        lo = mid[i - 1] + 1;
                        hi = mid[i] - 1;
                    }
                }
                if(locate[N] != locate[N+1])
                {
                    lo = mid[N] + 1;
                }
            } // end of single
        }
    } // end of parallel
    return pos;
}

int main()
{
    int *array = (int *) malloc(sizeof(int)*SIZE);  // to allocate memory for array of total number
                                                    //of nodes in tree
    long  int pos, i;                               // For calculating position and temp variable
    double start_time;

    int key = 10;                                   //It is a key to search in the record
    for(i=0;i<SIZE;i++)                     //Putting some values in the array
    {
        array[i] = i-SIZE/2;
    }
    printf("\nSearching...");
    // Start timer
    start_time = omp_get_wtime();

    pos = nary_search(array, SIZE, key);  // array is memory allocation parameter in n_ary
                                          //search
                                         // SIZE is total no of elements present in n_ary search
                                        // key is the element to find in tree
    if(pos!=-1)
    {
        printf("\nKey: %d is found in the record at location: %ld", key, pos);
    }
    else
    {
        printf("\n Key is not found");
    }

    printf("\nExecution time = %lf seconds\n", omp_get_wtime() - start_time);
    free(array);
    return 0;
}
```

## Compilation Mechanism:

Program can be compiled as:

```
$ gcc –fopenmp  N-ary_search.c

$ ./a.out
```

## Output:

```
Searching...

Key: 40 is found in the record at location: 90

Execution time = 0.003000 seconds
```

**Conclusion:** In this way, we have implemented n-ary search algorithm using OpenMP.

# Assignment No 9

**Title:** Implement concurrent prims algorithm using OPENMP

**Requirements:** Any Linux Distro. with GCC 4.2 or above

## Theory:

Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.
A group of edges that connects two set of vertices in a graph is called cut in graph theory. So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

***How does Prim's Algorithm Work?***
The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *Minimum* Spanning Tree.

**Algorithm**
**1)** Create a set *mstSet* that keeps track of vertices already included in MST.
**2)** Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
**3)** While mstSet doesn't include all vertices
=> **a)** Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
=> **b)** Include *u* to mstSet.
=> **c)** Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

For concurrent prims, approach only differs as the number of nodes we are going to calculate have to be processed parallely. Simply by getting values of all minimal cost edges associated with each node.

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define DIM 1000         // for defining the total dimension of elements in form of array
void initialization(void);        // to define initialization method
void delete_elements(int);     // to define delete_elements method
struct prim_data {
int edges_weight[DIM][DIM];        // For edges cost
int dimension;                                  // Declaration in form of array
int U[DIM];                                      // User defined input
int total_min_distance;                   // Sum of minimum MST cost
int count_nodes_in_mst;                 // Total number of nodes traversed for MST
};
struct prim_data prim;                  //defining structure of different variables required for program
int main()
{
int ch,j,t,p_c,p_j,k,serial=1 ;
int i;
//variable that holds the current maximum distance
int min_distance;
//variable that holds the next node in MST
int new_next_element;
prim.total_min_distance = 0; // for total of minimal edges costs
prim.count_nodes_in_mst = 0; // for nodes involved in traversal of MST
//declaring the structs the are used by gettimeofday
//struct timeval tb1 ;
//struct timeval tb2;
//setting the minimum distance
min_distance = 1000;
//opterr = 0;
//parsing the arguments given
printf("Enter the number of nodes:\n");
scanf( "%d", &prim.dimension); // for storing the dimensions of Prims
printf("Enter the cost of edges: \n");
for (i = 0; i < prim.dimension; ++i) { // cost of edges in form of rows
for (j = 0; j < prim.dimension; j++) { // cost of edges in form of columns
scanf("%d",&(prim.edges_weight[i][j])); // Store into the edges
weight
```

```
printf("Cost: %d ",prim.edges_weight[i][j]); // print the edges cost
printf("From %d To %d\n",i,j); //
}//
printf("\n");
}

printf("\nPrinting the weight array...\n\n");
#pragma omp parallel num_threads(4),default(none),private(i,j),shared(prim)
// Used to share structure "Prim" to execute and
// so that all threads able to execute and calculate
// cost of edges
{

int tid = omp_get_thread_num();
printf("thread %d starting\n",tid);
#pragma omp for
for(i=0; i<prim.dimension; i++) {                    // dimension in row format
  for(j=0; j<prim.dimension; j++) {                  // dimension in column format
      printf("%d\t\n",prim.edges_weight[i][j]);    // print each edge cost associated with each node
     }
}//
printf("\n");
}
//initializing the data
initialization();
//calculating for all the nodes
for(k = 0; k < prim.dimension 1 ; k++)
{
min_distance = 1 000;
//for every node in minimum spanning tree
for(i = 0; i < prim.count_nodes_in_mst; i++)
{
//declaring OpenMP's directive with the appropriate scheduling...
#pragma omp parallel for
for(j = 0; j < prim.dimension; j++)
{
//find the minimum weight
// by calculating edges weight either greater than cost of MST or equal to 0
if(prim.edges_weight[prim.U[i]][j] > min_distance ||  prim.edges_weight[prim.U[i]][j]==0)
{
continue;
} else {
#pragma omp critical
  {
   min_distance = prim.edges_weight[prim.U[i]][j];
   new_next_element = j;
  }
 }
}

//Adding the local min_distance to the total_min_distance
prim.total_min_distance += min_distance;

//Adding the next node in the U set
prim.U[i] = new_next_element;


//Substructing the elements of the column in which the new node is assosiated with
delete_elements( new_next_element );


//Increasing the nodes that they are in the MST
prim.count_nodes_in_mst++;
}
printf("\n");
//Print all the nodes in MST in the way that they stored in the U set
for(i = 0 ; i < prim.dimension; i++) {
printf("%d ",prim.U[i] + 1 );
if ( i < prim.dimension  1 ) printf("> ");
}
printf("\n\n");
printf("Total minimun distance: %d\n\n", prim.total_min_distance);
printf("\nProgram terminates now..\n");
return 0;
}

void initialization(void) {
int i,j;
prim.total_min_distance = 0;
prim.count_nodes_in_mst = 0;
```

```
//initializing the U set i.e. User set
for(i = 0; i < prim.dimension; i++) prim.U[i] = 1 ;

//storing the first node into the U set
prim.U[0] = 0;

//deleting the first node
delete_elements( prim.U[0] );
//incrementing by one the number of node that are inside the U set
prim.count_nodes_in_mst++;
}

void delete_elements(int next_element) {
int k;
for(k = 0; k < prim.dimension; k++) {
prim.edges_weight[k][next_element] = 0;  // k is defined to delete the Prims Edges Cost
}
}
```

## Compiling mechanism:

```
$ gcc –fopenmp concurrent_prims.c

$ ./a.out
```

## Output:

```
Enter the number of nodes:
4

Enter the cost of edges:

0 3 4 5

Cost: 0 From 0 To 0
Cost: 3 From 0 To 1
Cost: 4 From 0 To 2
Cost: 5 From 0 To 3

3 0 6 1

Cost: 3 From 1 To 0
Cost: 0 From 1 To 1
Cost: 6 From 1 To 2
Cost: 1 From 1 To 3
```

```
4 6 0 2

Cost: 4 From 2 To 0
Cost: 6 From 2 To 1
Cost: 0 From 2 To 2
Cost: 2 From 2 To 3

5 1 2 0

Cost: 5 From 3 To 0
Cost: 1 From 3 To 1
Cost: 2 From 3 To 2
Cost: 0 From 3 To 3

Printing the weight array...

thread 2 starting

4
6
0
2

thread 0 starting

0
3
4
5

thread 3 starting
5
1
2
0

thread 1 starting
3
0
6
1

1 -> 2 -> 4 -> 3

Total minimun distance: 6
```

**Conclusion:** In this way, we have implemented the concurrent prims using OpenMP.

# Assignment No 11

**Title:** Implement a Multi-threading application for echo server using socket programming in JAVA

**Requirements:** Any Windows or Linux Distro. with JDK 6.x or above.

**Socket Programming :**

Socket programming is useful for building client-server applications.

**The server :**

Creates a socket with some port number (>1023):

*ServerSocketechoServer = new ServerSocket(6789);*

Waits for client connection :

*Socket clientSocket = echoServer.accept();*

Gets input/output streams :

*BufferedReader is = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));*

*PrintStreamos = new PrintStream(clientSocket.getOutputStream());*

Exchanges information with the client :

*String line = is.readLine();*

*os.println( "Echo: " + line );*

Clean up :

*is.close();   os.close();*

*clientSocket.close();   echoServer.close();*

**The client :**

Creates a socket with the same port number :

*Socket clientSocket = new Socket( "localhost", 6789 );*

Gets input/output streams.

Exchanges information with the server.

Clean up.

**ECHO Server :**

**What is it and what is it for ?**

An "echo server" is a server that does nothing more than sending back whatever is sent to it. Hence the name : echo What can you use it for ? Whatever you feel like. Practical applications could be network and connectivity testing and troubleshooting. Assume you've build a rather complex network with VLANs and subnets, and really strick firewalls between those subnets, and you're beginning to wonder if a client on one segment of the network will still be able to connect to your web server, database server, ... on some other segment. A ping or a traceroute will establish if the server (IP address) can be reached but does not tell you if an application will be able to connect to the desired port on the server and whether a reply from the server will be able to reach the client again.

This "echo server" can be set up to listen on any desired (tcp) port to simulate whatever application you want to run (eg web server = port 80, Microsoft SQL Server = port 1433, etc). From the client machine, you can then telnet to this port. When a telnet connection has been established, everything you type will be echoed back to your screen, indicating that the telnet client and the echo server can talk to each other : you've established connectivity at the application level.
In a similar way, you can use this echo server to troubleshoot networks, test a firewall (eg "if I have a server listening on port 123, wil my firewall allow connections to it ?) and so on.

**Echo server :**

1. The client reads a line from its standard input and writes that line to the server.
2. The server reads a line from its network input and echoes the line back to the client over the network.
3. The client reads the echoed line from the network and prints it on its standard output.

**Multithreaded Server Advantages :**

The advantages of a multithreaded server compared to a single threaded server are summed up below:
Less time is spent outside the accept() call.
Long running client requests do not block the whole server.
In a single threaded server long running requests may make the server unresponsive for a long period. This is not true for a multithreaded server, unless the long-running request takes up all CPU time and/or network bandwidth.

## Program:

***Server.java***

```
import java.net.*;
import java.io.*;
class Server {
        public static void main(String args[]) throws IOException {
        ServerSocket ss = null;
 try {
   ss = new ServerSocket(95);
}  catch (IOException ioe)  {
   System.out.println("Error finding port");
   System.exit(1);
}
  Socket soc = null;

try {
 soc = ss.accept();
 System.out.println("Connection accepted at :" + soc);
} catch (IOException ioe) {
 System.out.println("Server failed to accept");
 System.exit(1);
}
```

```
  DataOutputStream dos = new DataOutputStream(soc.getOutputStream());

  BufferedReader br = new BufferedReader(new InputStreamReader(soc.getInputStream()));

  String s;

  System.out.println("Server waiting for message from the client");

  boolean quit = false;

 do {
 String msg = "";
 s = br.readLine();
 int len = s.length();

 if (s.equals("exit")) {
   quit = true;
 }

 for (int i = 0; i < len; i++) {
   msg = msg + s.charAt(i);
  dos.write((byte) s.charAt(i));
 }
 System.out.println("From client :" + msg);
 dos.write(13);
 dos.write(10);
 dos.flush();
} while (!quit);

  dos.close();
  soc.close();
   }
}
```

***Client.java***

```
 import java.net.*;
 import java.io.*;

  class Client {
    public static void main(String args[]) throws IOException {
       Socket soc = null;
       String str = null;
       BufferedReader br = null;
       DataOutputStream dos = null;
       BufferedReader kyrd = new BufferedReader(new InputStreamReader(System.in));
   try {
    soc = new Socket(InetAddress.getLocalHost(), 95);
    br = new BufferedReader(new InputStreamReader(soc.getInputStream()));
    dos = new DataOutputStream(soc.getOutputStream());
    } catch (UnknownHostException uhe) {
    System.out.println("Unknown Host");
    System.exit(0);
    }
    System.out.println("To start the dialog type the message in this client window \n Type exit to end");
    boolean more = true;
    while (more) {
    str = kyrd.readLine();
    dos.writeBytes(str);
    dos.write(13);
    dos.write(10);
    dos.flush();
    String s, s1;
    s = br.readLine();
    System.out.println("From server :" + s);
    if (s.equals("exit")) {
    break;
    }
    }
    br.close();
    dos.close();
    soc.close();
    }
 }
```

## Compiling mechanism:

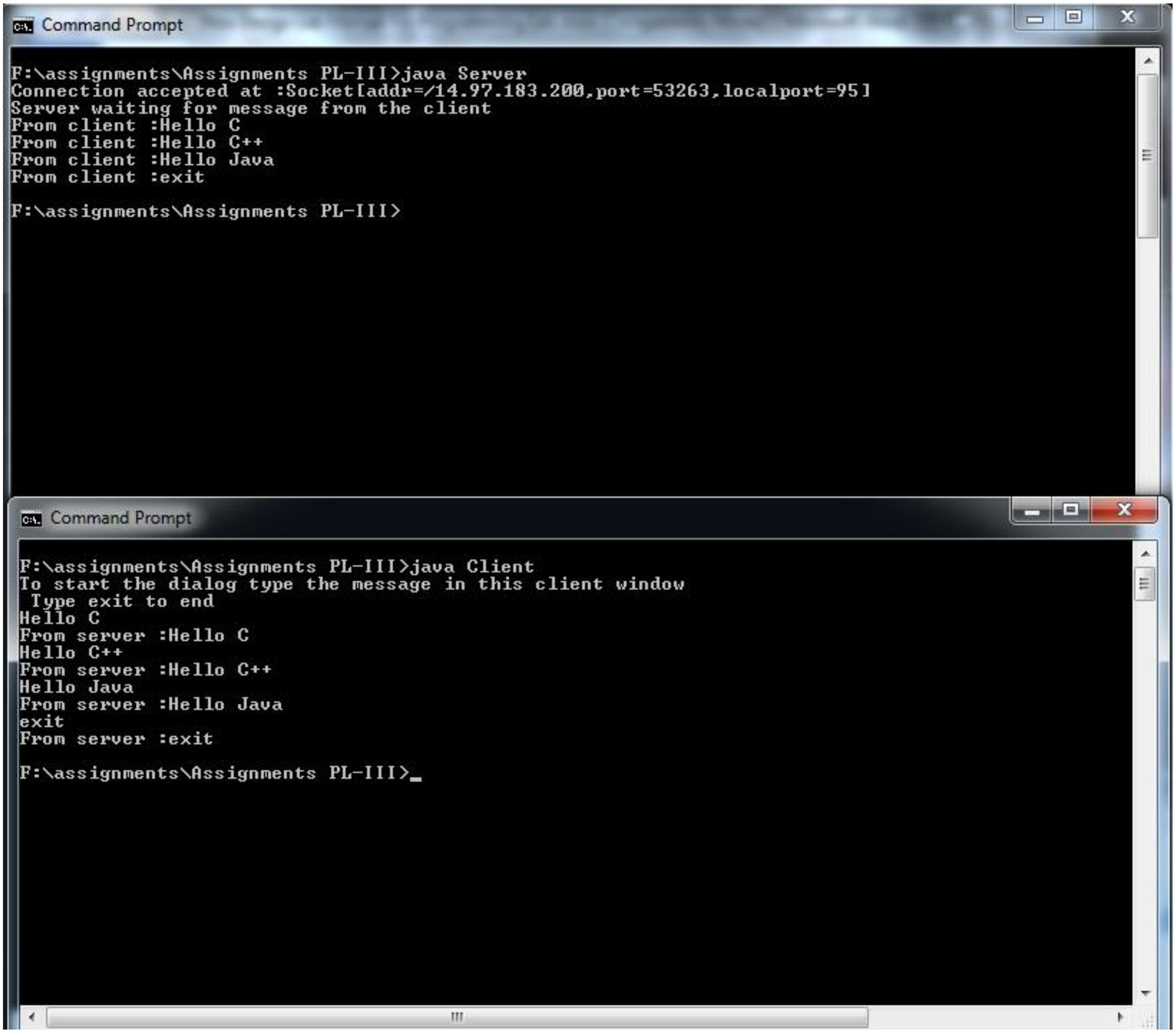
```
$ javac Client.java

$ java Client

$ javac Server.java

$ java Server
```

## Output:



```
F:\assignments\Assignments PL-III>java Server
Connection accepted at :Socket[addr=/14.97.183.200,port=53263,localport=95]
Server waiting for message from the client
From client :Hello C
From client :Hello C++
From client :Hello Java
From client :exit

F:\assignments\Assignments PL-III>
```

```
F:\assignments\Assignments PL-III>java Client
To start the dialog type the message in this client window
 Type exit to end
Hello C
From server :Hello C
Hello C++
From server :Hello C++
Hello Java
From server :Hello Java
exit
From server :exit

F:\assignments\Assignments PL-III>_
```

**Conclusion:** In this way, we have implemented a Multi-threading application for echo server using socket programming in JAVA.

# Assignment No 12

**Title:** Implement Reader-Writer problem using OPENMP.

**Requirements:** Any Linux Distro. with GCC 4.2 or above

## Theory:

The R-W problem is another classic problem for which design of synchronization and concurrency mechanisms can be tested. The producer/consumer is another such problem; the dining philosophers is another.

**Definition:**

- There is a data area that is shared among a number of processes.
- Any number of readers may simultaneously write to the data area.
- Only one writer at a time may write to the data area.
- If a writer is writing to the data area, no reader may read it.
- If there is at least one reader reading the data area, no writer may write to it.
- Readers only read and writers only write
- A process that reads and writes to a data area must be considered a writer (consider producer or consumer)

The readers/writers problem is one of the classic synchronization problems. Like the dining philosophers, it is often used to compare and contrast synchronization mechanisms. It is also an eminently practical problem.

**Readers/Writers Problem - Classic definition**

Two kinds of processes -- readers and writers -- share a database. Readers execute transactions that examine database records; writer transactions both examine and update the database. The database is assumed initially to be in a consistent state (i.e., one in which relations between data are meaningful). Each transaction, if executed in isolation, transforms the database from one consistent state to another. To preclude interference between transactions, a writer process must have exclusive access to the database. Assuming no writer is accessing the database, any number of readers may concurrently execute transactions.

A data object is to be shared among several concurrent processes. Some of these processes may want to only to read the content of the shared object, whereas others may want to update the shared object. We distinguish between these two types of processes by referring to those processes that are interested in only reading as readers, and to the rest as writers. Obviously, if two readers access the shared data object simultaneously, no adverse effects will result.

The readers-writers problem has several variations, all involving priorities. The simplest one, referred to as the first readers-writers problem, requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. The second readers-writers problem requires that, once a writer is ready, that writer performs its write as soon as possible. Here, I adopt the first readers-writers problem. At a given time, there is only one writer and any number of readers. When a writer is writing, the readers can not enter into the database. The readers need to wait until the writer finishes to write on the database. Once a reader succeed in reading the database, subsequent readers can enter into the critical section (in this case, say, database) without waiting for the precedent reader finish to read. On the other hand, a writer who arrives later than the reader who is reading currently is required to wait the last reader finish to read. Only when the last reader finish reading, the writer can enter into the critical section and is able to write on the database.

To implement this algorithm, we need two kinds of semaphore, that is wrt and mutex. The former plays a role of lock for the database itself. Readers or writers can enter into the database only if they have this semaphore. The later semaphore is used to lock the critical section prier and posterior to reading: we need them to increase and decrease the number of readers in database.

Suppose we have a shared memory area with the constraints detailed above. It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. However, this solution is suboptimal, because it is possible that a reader $R_1$ might have the lock, and then another reader $R_2$ requests access. It would be foolish for $R_2$ to wait until $R_1$ was done before starting its own read operation; instead, $R_2$ should start right away. This is the motivation for the first readers-writers problem, in which the constraint is added that *no reader shall be kept waiting if the share is currently opened for reading.* This is also called readers-preference,

In this solution of the readers/writers problem, the first reader must lock the resource (shared file) if such is available. Once the file is locked from writers, it may be used by many subsequent readers without having them to re-lock it again.
Before entering the Critical Section, every new reader must go through the entry section. However, there may only be a single reader in the entry section at a time. This is done to avoid race conditions on the readers. To accomplish this, every reader which enters the <ENTRY Section> will lock the <ENTRY Section> for themselves until they are done with it. Please note that at this point the readers are not locking the resource. They are only locking the entry section so no other reader can enter it while they are in it. Once the reader is done executing the entry section, it will unlock it by signalling the mutex semaphore. Signalling it is equivalent to: mutex.V() in the above code. Same is valid for the <EXIT Section>. There can be no more than a single reader in the exit section at a time, therefore, every reader must claim and lock the Exit section for themselves before using it.

Once the first reader is in the entry section, it will lock the resource. Doing this will prevent any writers from accessing it. Subsequent readers can just utilize the locked (from writers) resource. The very last reader (indicated by the readcount variable) must unlock the resource, thus making it available to writers.

In this solution, every writer must claim the resource individually. This means that a stream of readers can subsequently lock all potential writers out and starve them. This is so, because after the first reader locks the resource, no writer can lock it, before it gets released. And it will only be released by the very last reader. Hence, this solution does not satisfy fairness.

**Program:**

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>  // used for process control according to POSIX standard
#include <omp.h>

int main()
{
int i=0,NumberofReaderThread=0,NumberofWriterThread;

omp_lock_t writelock;    // for defining a lock of one transaction session
omp_init_lock(&writelock); // to initialize a lock

int readCount=0;          // no of times a particular read by processes

printf("\nEnter number of Readers thread(MAX 10):");
scanf("%d",&NumberofReaderThread);
printf("\nEnter number of Writers thread(MAX 10):");
scanf("%d",&NumberofWriterThread);

int tid=0;
#pragma omp parallel
#pragma omp for
for(i=0;i<NumberofReaderThread;i++)
{
 time_t rawtime;                  // for defining
 struct tm * timeinfo;            // the time of process invokation

 time ( &rawtime );
 timeinfo = localtime ( &rawtime );
 ////printf ( "Current local time and date: %s", asctime (timeinfo) );
 //  sleep(2);

 printf("\nReader %d is trying to enter into the Database for reading the data",i);

  omp_set_lock(&writelock);             // to set lock for reading
  readCount++;                // increase the read count so the status of process is going to change
  if(readCount==1)
  {
    printf("\nReader %d is reading the database",i);
  }

  omp_unset_lock(&writelock);
  readCount--;
  if(readCount==0)
  {
    printf("\nReader %d is leaving the database",i);
  }
}

#pragma omp parallel shared(tid)
#pragma omp for nowait                 // continue execution of a parallel region without waiting for the
other threads
for(i=0;i<NumberofWriterThread;i++)
{
  printf("\nWriter %d is trying to enter into database for modifying the data",i);
  omp_set_lock(&writelock);
  printf("\nWriter %d is writing into the database",i);
  printf("\nWriter %d is leaving the database",i);
  omp_unset_lock(&writelock);
}

 omp_destroy_lock(&writelock);        // destroy to end the session
return 0;
}
```

## Compiling mechanism:

```
$ gcc –fopenmp reader-writer.c

$ ./a.out
```

## Output:

```
Enter number of Readers thread(MAX 10):2

Enter number of Writers thread(MAX 10):2


Reader 1 is trying to enter into the Database for reading the data

Reader 1 is reading the database

Reader 1 is leaving the database

Reader 0 is trying to enter into the Database for reading the data

Reader 0 is reading the database

Reader 0 is leaving the database
```

```
Writer 1 is trying to enter into database for modifying the data

Writer 1 is writing into the database

Writer 1 is leaving the database

Writer 0 is trying to enter into database for modifying the data

Writer 0 is writing into the database

Writer 0 is leaving the database
```

**Conclusion:** In this way, we have implemented Implement Reader-Writer problem using OPENMP.