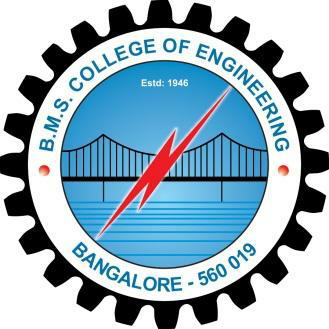
**B.M.S College of Engineering**

**P.O. Box No.: 1908 Bull Temple Road,**

**Bangalore-560 019**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**



**Course – Unix System Programming**

**Course Code – 19IS4PWUSP**

**AY 2019-20**

**Initial Report on USP Project**

**Multiple Arithmetic Operations and**

**Two Lock Queue using Multi-Threading**

Submitted by

**TANMAY SRINATH, 1BM18IS116**

**TEJASVI SHRIVASTAVA, 1BM18IS117**

**GUIDED BY :Sindhu K Ma’am**

|  |
| --- |
| **CERTIFICATE**  Certified that the Project has been successfully presented at **B.M.S College Of Engineering** by **Tanmay Srinath and Tejasvi Shrivastava** bearing USN: **1BM18IS116 and 1BM18IS117** in partial fulfillment of the requirements for the IV Semester degree in **Bachelor of Engineering in Information Science & Engineering** of **Visvesvaraya Technological University, Belgaum** as a part of project for the course **UNIX System Programming (19IS4PWUSP)** during academic year 2019-2020.  **Faculty Name – Sindhu K**  **Designation – Assistant Professor**  **Department of ISE, BMSCE**  **B.M.S College of Engineering**  **P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019**  **DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING** |

**CONTENTS**

|  |  |
| --- | --- |
| **Topic** | **Page No.** |
| Abstract | 3 |
| Introduction | 4 |
| Problem Statement | 6 |
| APIs Used | 6 |
| Explanation of APIs | 6 |
| Implementation and Results |  |

**Abstract:**

Drawing ideas from previous authors, we present a new non-blocking concurrent queue algorithm and a new twolock queue algorithm in which one enqueue and one dequeue can proceed concurrently. Both algorithms are simple, fast, and practical; we were surprised not to find them in the literature. Experiments on a 12-node SGI Challenge multiprocessor indicate that the new non-blocking queue consistently outperforms the best known alternatives; it is the clear algorithm of choice for machines that provide a universal atomic primitive (e.g. compare and swap or load linked/store conditional). The two-lock concurrent queue outperforms a single lock when several processes are competing simultaneously for access; it appears to be the algorithm of choice for busy queues on machines with non-universal atomic primitives (e.g. test and set). Since much of the motivation for non-blocking algorithms is rooted in their immunity to large, unpredictable delays in process execution,we report experimental results both for systems with dedicated processors and for systems with several processes multiprogrammed on each processor.

**Introduction**

In regular menu based programs, it is possible for the user to use all the operations, but this is an inefficient model, as the programmer has to either include a loop, when a single method is executed multiple times and the memory space is wasted, or he excludes a loop, where the user has to repeatedly run the program in order to get all possible outputs. Multi-threading hopes to resolve this issue. The fundamental advantages of multi-threading are:

1. Enhanced performance by decreased development time
2. Simplified and streamlined program coding
3. Improvised GUI responsiveness
4. Simultaneous and parallelized occurrence of tasks
5. Better use of cache storage by utilization of resources
6. Decreased cost of maintenance
7. Better use of CPU resource

One of the simplest and efficient ways of demonstrating the advantages of multi threading are through more than one arithmetic operations being executed on the same set of numbers. Through our project, we aim to understand the uses of Multi threading and aim to demonstrate its effectiveness even in the simplest of cases.

For our project, we will be using the UNIX API PThread. Pthread is available as a header file in the programming languages c and c++. Using the functions available in pthread, we plan to take a few numbers as input and then ask the user how many, if not all, arithmetic operations he wishes to execute. Once this is done, we ask him to input in what order he wants these operations to be executed. Once this is taken from the user, each thread will then begin execution, and the user will be shown when the thread starts, functions and stops. This will provide him/her useful insight about multi threaded programming. With regards to the specific details, we will list them in our upcoming reviews.

**Two lock queue:**

Queues are ubiquitous in parallel programs, and their performance is a matter of major concern. We have presented a concurrent queue algorithm that is simple, nonblocking, practical, and fast. We were surprised not to find it in the literature. It seems to be the algorithm of choice for any queue-based application on a multiprocessor with universal atomic primitives (e.g. compare and swap or load linked/store conditional). We have also presented a queue with separate head and tail pointer locks. Its structure is similar to that of the nonblocking queue, but it allows only one enqueue and one dequeue to proceed at a given time. Because it is based on locks, however, it will work on machines with such simple atomic primitives as test and set. We recommend it for heavily-utilized queues on such machines (For a queue that is usually accessed by only one or two processors, a single lock will run a little faster.) This work is part of a larger project that seeks to evaluate the trade offs among alternative mechanisms for atomic update of common data structures. Structures under consideration include stacks, queues, heaps, search trees, and hash tables. Mechanisms include single locks, data structure-specific multi-lock algorithms, general-purpose and special-purpose non-blocking algorithms, and function shipping to a centralized manager (a valid technique for situations in which remote access latencies dominate computation time).

**PROBLEM STATEMENT**

**ARITHMETIC OPERATIONS USING THREADS**

* ADD,SUBTRACT,DIVISION ,MULTIPICATION OF TWO NUMBERS
* POWER , Nth ROOT
* LOGARITHMIC OPERATION WITH GIVEN BASE

**TWO LOCK QUEUE USING THREADS**

* ENQUEUE IN A LIST
* DEQUEUE IN A LIST
* TRAVERSE
* ENHANCEMENTS
* DRAWBACKS OF BIG KERNEL LOCK

**APIs USED**

* MUTEX LOCK
* POSIX
* BLOCKING AND NON BLOCKING ALGORITHM
* PTHREAD

**EXPLANATION OF APIs USED**

* **Read:** Used to read data from the user.
* **Sudo:** Used to get the superuser privilege. Sudo is a program for unix-like computer operating systems that allows users to run programs with the security privileges of another user, by default the superuser.
* **Open:** Used to open a file. An open API (often referred to as a public API) is a publicly available API that provides developers with programmatic access to a propriety software application or web service.
* **POSIX**stands for Portable Operating System Interface, and is an IEEE standard designed to facilitate application portability. **POSIX** is an attempt by a consortium of vendors to create a single standard version of UNIX. If they are successful, it will make it easier to port applications between hardware platforms

Code:

Arithmetic Operations Using Multi Threading:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

typedef struct tn

{

int a;

int b;

int oprtn;

}tn;

int counter;

pthread\_mutex\_t lock;

void \* mtmcloprtns(void\* arg);

int main(int argc, char\* argv)

{

int a1=0,b1=0,i,i1=0,error;

if (pthread\_mutex\_init(&lock, NULL) != 0)

{

printf("\n Mutex Initialisation has failed\n");

return 1;

}

printf("Enter the 2 Numbers on which arithmetic operations need to be performed");

scanf("%d%d",&a1,&b1);

tn obj[5];

int oprtns[5];

printf("Enter the order of Operations to be performed, with each symbol as such:\n 1)+\n2)-\n3)\*\n4)/\n 5)Modulus");

for(i=0;i<5;i++)

{

scanf("%d",&oprtns[i]);

obj[i].a=a1;

obj[i].b=b1;

obj[i].oprtn=oprtns[i];

}

pthread\_t t[5];

while(i1<5)

{

error = pthread\_create(&(t[i1]),NULL,&mtmcloprtns,(void \*)&obj[i1]);

if (error != 0)

printf("\nThread can't be created :[%s]",strerror(error));

i1++;

}

pthread\_join(t[0], NULL);

pthread\_join(t[1], NULL);

pthread\_join(t[2], NULL);

pthread\_join(t[3], NULL);

pthread\_join(t[4], NULL);

pthread\_mutex\_destroy(&lock);

return 0;

}

void \*mtmcloprtns(void\* arg)

{

pthread\_mutex\_lock(&lock);

counter += 1;

tn\* args=(tn\*)arg;

int x1=args->oprtn;

int xa=args->a,xb=args->b;

printf("\n Operation %d has started\n", counter);

switch(x1)

{

case 1:printf("Sum is %d",(xa+xb));

break;

case 2:printf("Diff is %d",(xa-xb));

break;

case 3:printf("Product is %d",(xa\*xb));

break;

case 4:printf("Quotient is %d",(xa/xb));

break;

case 5:printf("Remainder is %d",(xa%xb));

break;

default:printf("None satisfy. Try again");

}

printf("\n Operation %d has finished\n", counter);

pthread\_mutex\_unlock(&lock);

return NULL;

}

Two Lock Queue:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

typedef struct node

{

int val;

struct node \*next;

}node;

typedef struct queue

{

node \*head, \*tail;

pthread\_mutex\_t hl,tl;

}queue;

typedef struct inp

{

queue \*q;

int v;

}inp;

node\* get\_node()

{

node\* q = (node\*)malloc(sizeof(node));

return q;

}

queue\* get\_queue()

{

queue\* q = (queue\*)malloc(sizeof(queue));

return q;

}

node\* initialise(queue\* q)

{

node \*n = get\_node();

n->next=NULL;

q->tail=n;

q->head=q->tail;

return n;

}

void \*enqueue(void\* arg)

{

node \*n = get\_node();

inp \*ip = (inp\*)arg;

queue \*q1 = ip->q;

n->val = ip->v;

n->next = NULL;

pthread\_mutex\_lock(&q1->tl);

q1->tail->next=n;

q1->tail=n;

pthread\_mutex\_unlock(&q1->tl);

printf("Enqueue successful\n");

}

void \*dequeue (void\* arg)

{

inp \*ip = (inp\*)arg;

queue \*q1 = ip->q;

pthread\_mutex\_lock(&q1->hl);

node \*n = q1->head;

if(n==NULL||n->next==NULL)

{

pthread\_mutex\_unlock(&q1->hl);

printf("Impossible to deque as there is not enough nodes\n");

return;

}

node \*new\_head = n->next;

printf("Value before release is %d\n",n->val);

q1->head=new\_head;

pthread\_mutex\_unlock(&q1->hl);

free(n);

printf("Dequeue successful\n");

}

void traverse(queue \*q)

{

node \*n=q->head;

if(n==NULL)

return;

while(n->next!=NULL)

{

printf("%d\n",n->val);

n=n->next;

}

}

int main()

{

queue \*q1 = get\_queue();

node \*n = initialise(q1);

pthread\_t t[10];

inp i1[10];

int i=0;

for(i=0;i<10;i++)

{

i1[i].v=i+1;

i1[i].q=q1;

}

pthread\_create(&(t[0]),NULL,&enqueue,(void \*)&i1[0]);

pthread\_create(&(t[1]),NULL,&enqueue,(void \*)&i1[1]);

traverse(q1);

pthread\_create(&(t[2]),NULL,&dequeue,(void \*)&i1[2]);

traverse(q1);

pthread\_create(&(t[3]),NULL,&enqueue,(void \*)&i1[3]);

pthread\_create(&(t[4]),NULL,&dequeue,(void \*)&i1[4]);

pthread\_create(&(t[5]),NULL,&enqueue,(void \*)&i1[5]);

traverse(q1);

pthread\_create(&(t[6]),NULL,&dequeue,(void \*)&i1[6]);

pthread\_create(&(t[7]),NULL,&enqueue,(void \*)&i1[7]);

traverse(q1);

pthread\_create(&(t[8]),NULL,&dequeue,(void \*)&i1[8]);

traverse(q1);

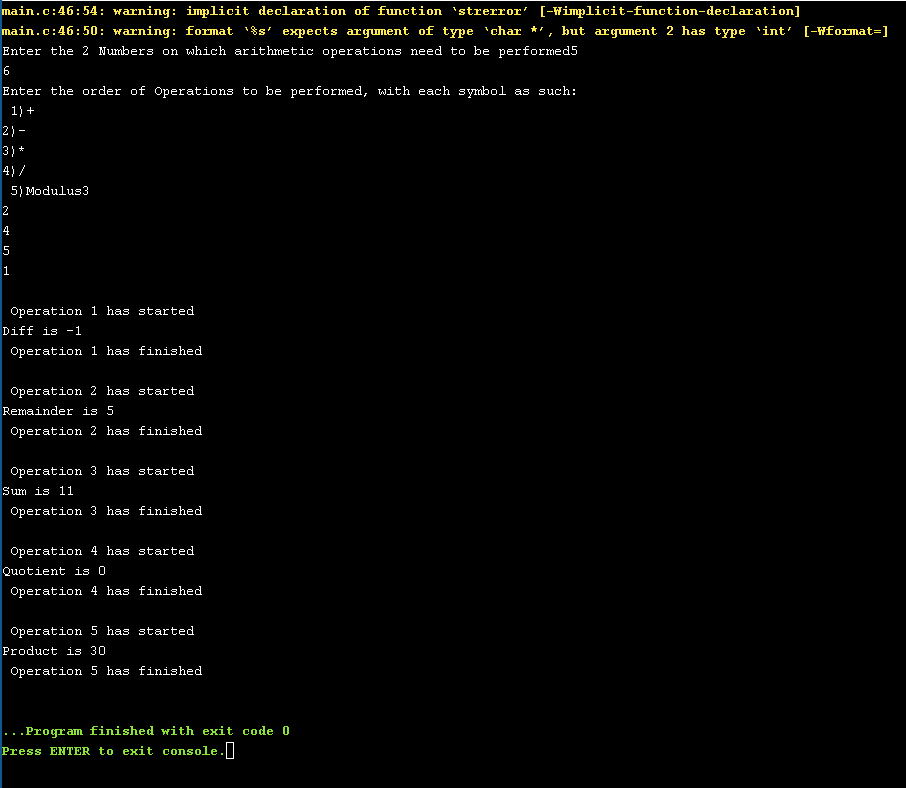
pthread\_create(&(t[9]),NULL,&dequeue,(void \*)&i1[9]);

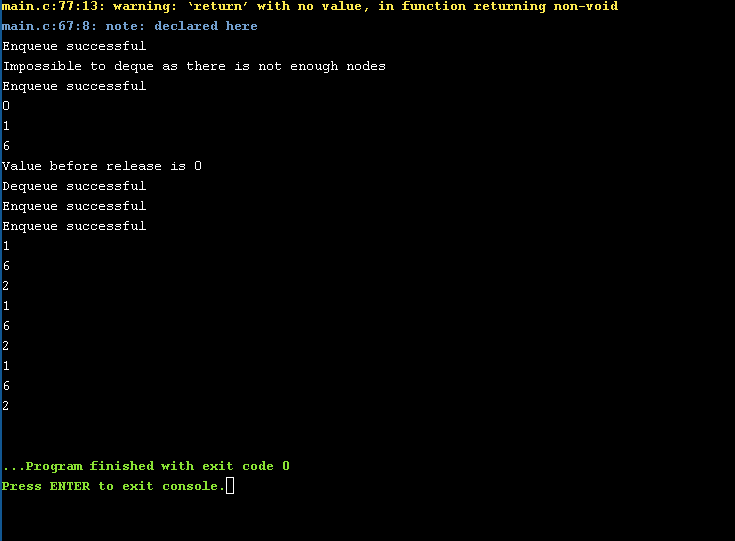
traverse(q1);

return 0;

}

Output:

Arithmetic Operations Using Multi Threading

Two Lock Queue: