**OPERATING SYSTEMS PROJECT**

**ON**

**THREADS AND MODIFIED CPU** SCHEDULING ALGORITHMS

Submitted by-

Dileep Kumar(12BIF0044)

**PROBLEM STATEMENTS**

1. Using threads for solve Sudoku validator (Also involves a model sum solved using threads)

(Including the code implementation carried out in C /C++ language)

1. Modified fuzzy based CPU scheduling algorithms.

(Including the code implementation carried out in C/C++)

THREAD PROGRAMMING AND SUDOKU SOLVING

MODEL SUM - A multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

Pseudocode-

Header <pthread.h>

Void \*saul(char ch[])

{

declare length, temp, a[50] ,k as integer variables

initialise k=0

Length=strlen(ch)

While temp!=0

a[k++]=temp%10

temp=temp/10

}

Declare max,min,I,sum as integer variables

Initialise max=min=a[0] and sum=0

For i=0 to i=k

If a[i]>max

Then max=a[i]

If a[i]<min

Then min=a[i]

Sum+=a[i]

End for

Print max,min,sum/k

Main()

{

Set Pthread\_t as t1

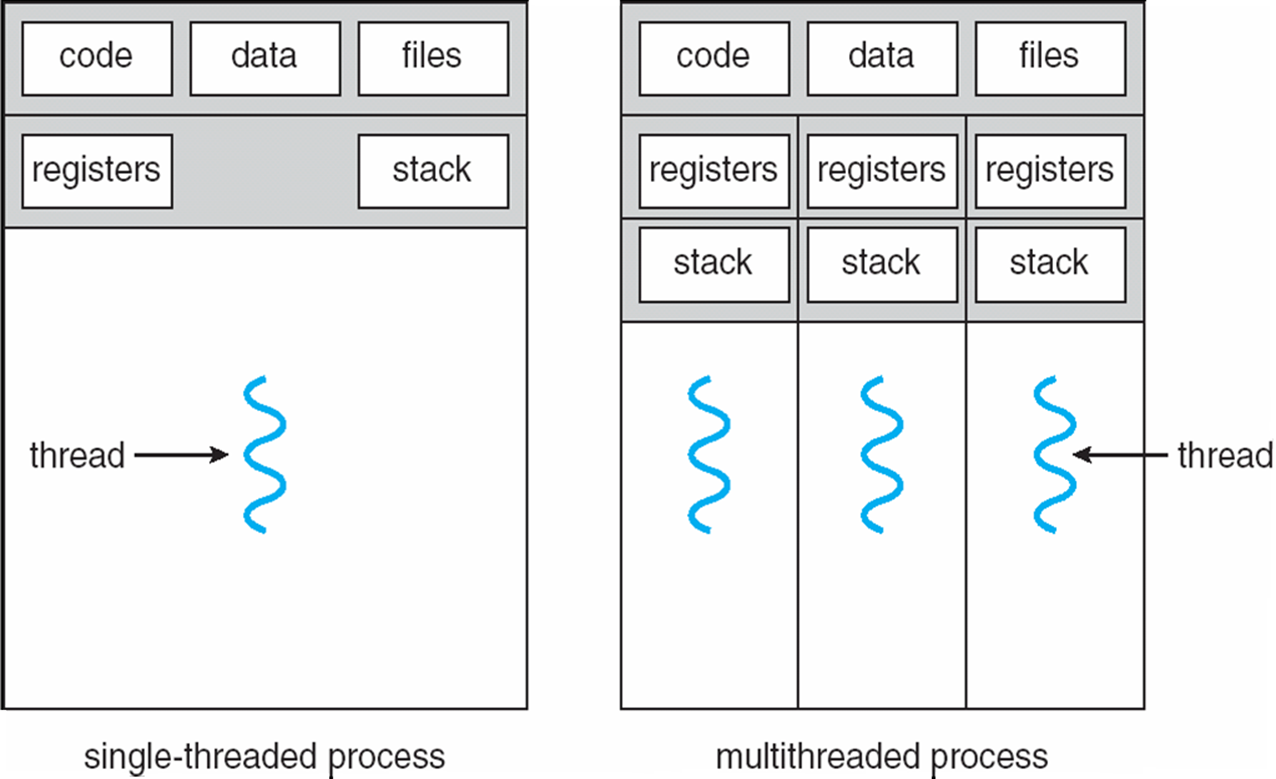
Declare s[20]

Print a string of numbers

Read s

Saul(s)

}



In the next part, there will be a Sudoku solution validator which will check whether the input entered for a NXN Sudoku is valid or not.

This checking will be carried out with the help of threads again.

Sudoku puzzles are always a type of [Latin square](http://en.wikipedia.org/wiki/Latin_square) with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column or in any of the sub regions of the playing board.

Manually solving a Sudoku takes some minutes even for an expert. However, computers can solve it in milliseconds. In our project, we code and implement a Sudoku validator which tells us whether the inputted number is valid for a given column/row/sub region.

In this way, we have a solution of the given Sudoku in optimal time. We’ll be using threads for this purpose.

These will involve the use of a multithreading model.

These are of three types. We’ll be preferably be using here a ‘Many – to –One’ model.

The benefits of such a model would be **Responsiveness, scalability and resource sharing.**

1. Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85. The program will report. The average value is 82. The minimum value is 72. The maximum value is 95.The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

Code:

#include<iostream>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

#include<stdio.h>

using namespace std;

void \*saul(char ch[])

{

int length;

int temp;

length=strlen(ch);

temp=atoi(ch);

int a[50]; int k=0;

while(temp!=0)

{ a[k++]=temp%10;

temp=temp/10;

}

int max,min,i;

float sum=0;

max=min=a[0];

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

{ if(a[i]>max)

max=a[i];

if(a[i]<min)

min=a[i];

sum+=a[i];

}

cout<<"\n Max:"<<max<<"\n Min:"<<min<<"\n Average:"<<sum/k;

}

int main()

{

Pthread\_t t1;

char s[20];

cout<<"Enter a string of numbers";

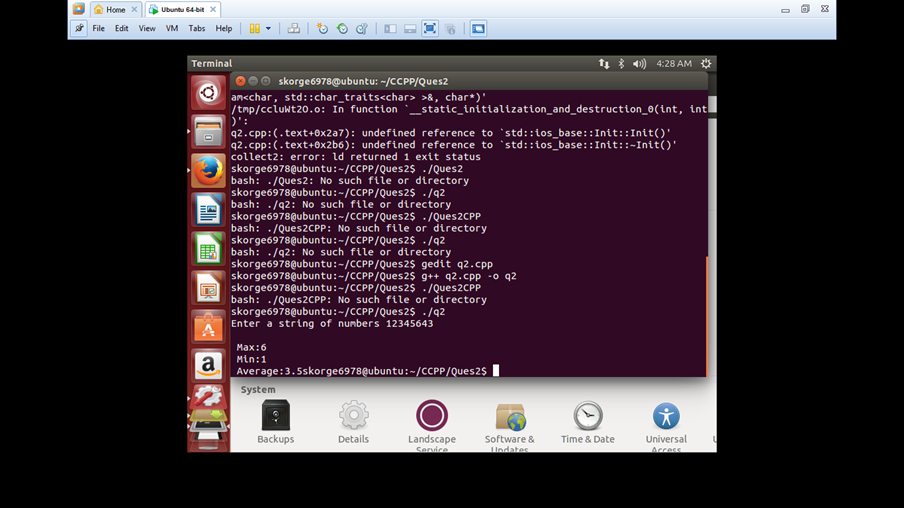
cin>>s;

saul(s);

return 0;

}

Output :



1. Sudoku Solution Validator

Code:

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

#define size 9

using namespace std;

int sudoku[size][size];

int Check (int y[]);

int temp[size];

void \*print(int matrix[size][size])

{

for (int i = 0; i < size; i++){

for (int j = 0; j < size; j++)

cout << matrix[i][j] << " ";

cout << endl;

}

return 0;

}

void \*RowCheck ( int x[size][size] )

{ int j;

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

{

for ( j= 0; j<size; j++)

temp[j] = x[i][j];

if ( Check (temp) == 0 )

{

cout<<"\n Not a correct solution";

exit(0);

}

}

}

int Check (int y[size]) {

int lineartemp,j;

for (int i=0; i<size; i++) {

lineartemp = y[i];

for ( j=i+1; j<size; j++) {

if (y[j] == lineartemp)

return 0;

}

}

return 1;

}

void \*ColumnCheck ( int x[size][size]) {

int j;

for (int i=0; i<size; i++) {

for (j= 0; j<size; j++)

temp[j] = x[j][i];

if ( Check (temp) == 0 )

{cout<<"\n Not a correct solution";

exit(0);

}

}

}

void \*SquaresCheck (int x[size][size])

{

int i;

int j;

int column;

int row;

int k;

for (column=0; column <= 6; column += 3) {

for (row=0; row <= 6; row += 3) {

for (k=0, i=0; i<3; i++) {

for (j=0; j<3; j++)

temp[k++] = x[row+i][column+j];

}

if (Check(temp) ==0)

{ cout<<"\n Not a correct solution";

exit(0);

}}}}

int main()

{

cout<< " \n Enter all the elements row wise ( Use space bar)\n";

int i,j;

pthread\_t t1,t2,t3;

typedef struct

{

int RowCheck;

int ColumnCheck;

int SquaresCheck;

}

parameters;

parameters \*data = (parameters \*) malloc(sizeof(parameters));

data->RowCheck = 0;

data->ColumnCheck = 0;

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

for(j=0;j<size;j++)

cin>>sudoku[i][j];

RowCheck(sudoku);

ColumnCheck(sudoku);

SquaresCheck(sudoku);

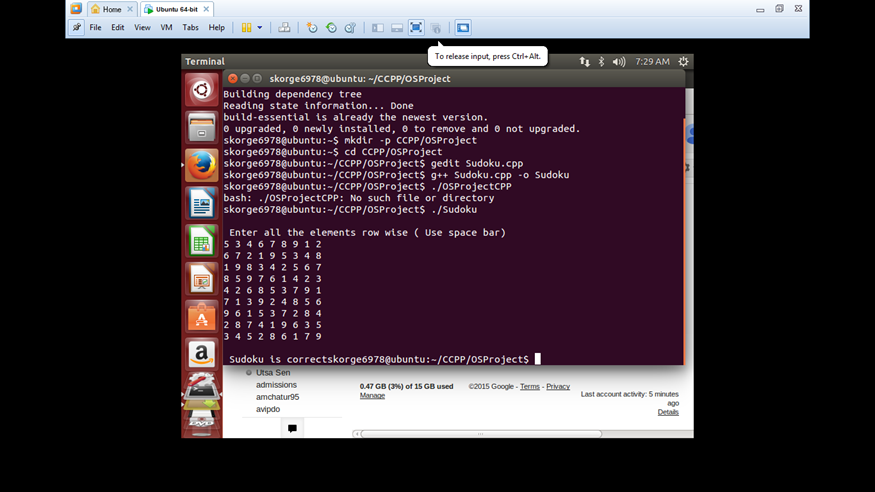
cout<<"\n Sudoku is correct";

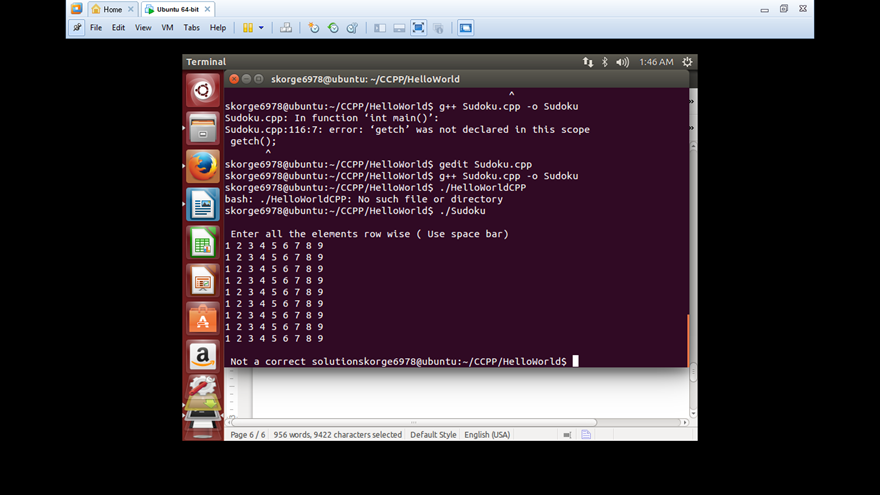
getch();

return 0;

}

OUTPUT:





**FUZZY BASED CPU SCHEDULING**

Scheduling is a method by which thread or process are granted access to computer resources.

Five basic features to decide best scheduling algorithm are as follow :

* Utilization of CPU time
* Throughput
* Turnaround Time
* Waiting Time
* Response time

There are number of CPU scheduling algorithms available, but it is very difficult task to decide which one is better.

Fuzzy based CPU Scheduling demonstrates that scheduling done with **new priority improves average waiting time and average turnaround time.**

Fuzzy based scheduling algorithm proposed by Kadhim, Shatha J., and Kasim M. Al-Aubidy in .This algorithm takes input both job priority and execution time and decides new priority using some fuzzy rules.

**FUZZY LOGIC**: Fuzzy logic is a style of multi-valued logic. It deals with approximation rather than exactness. In contrast to classical sets (Classical set takes true or false values) fuzzy logic variables (also known as linguistic variable) can have a truth value that ranges in interval between 0 and 1. Fuzzy logic has been prolonged to grasp the concept of fractional truth, where the truth value may range between completely true and completely false. Moreover, when linguistic variables are practiced, these degrees may be determined by specific methods.

PROPOSED ALGORITHM: This scheduling deal with some fuzzy rules and these rules are based on assigned priority and execution time. This work is proposed to compute the New Priority (NP) for all tasks using pre priority and execution time.

There is use of suitable linguistic variables as input and output for compute a crisp value for new priority. Pre Priority (PP) measured as Very Low, Low, Medium, High and Very High. Execution Time (ET) measured as Very Small, Small, Medium, Long and Very Long. New Priority (NP) measured as Very Low, Low, Medium, High and Very High. The proposed scheduling is a collection of linguistic fuzzy rules which describe the relationship between defined input variables (PP and ET) and output (NP).

HOW TO CALCULATE THE NEW PRIORITY USING FUZZY RULES:

Pseudo code and flow chart of the proposed algorithm:

1. Intialize n processes with their burst time, arrival time and priorities.

2. Evaluate µp i.e membership value of task priority for individual processes by using the formula actual task priority / (maximum task priority+1)

3. Evaluate µb that is membership value of burst time for individual processes

( 1-(actual burst time/(maximum burst time+1) )

4. Find response ratios of individual processes after each iteration.

5. Evaluate µh i.e membership value of response ratio

actual response ratio/(maximum response ratio+1)

6. Evaluate µni: membership value of new priority after fuzzification for ith process by the formula pni = max{ µbi, µpi, µhi } where 1<=i<=n

7. Apply bubble sort to get the descending order of new priorities.

for(i=1;i<n;i++) {

for(j=1;j<n-i;j++)

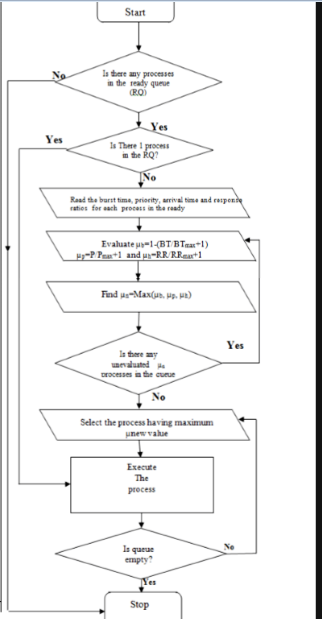
{ if (pni<pni+1)

{

Swap the two processes } } }

8. Execute the processes in the sorted sequence

9. Stop & exit



After assigning new priority, execution is start according to assigned priority and arrival time. Firstly find out the entire tasks which have minimum arrival time (0 sec) and execute the task which has higher priority from them till next task will enter in queue. Whenever a new task will enter again check the priority of entered task and execute the higher priority task till next tasks will enter and so on. After entering all tasks in queue execute remaining task as per higher to lower priority.

1.) Initialize n processes with their Execution Time, Priority and Arrival Time.

2.) Compute New Priority (NP) using fuzzy rules with the help of Execution Time and Priority.

3.) Apply sorting method on arrival time to arrange in ascending order.

for (i=1; i<n; i++)

{ for (j=1; j<n-i; j++)

if (ATnj>ATnj+1)

swap

4.) Compute the frequency of same arrival time and store in x variable.

z=1, AT = ATni, x=1

for (i=z; a<=n; i++)

if (ATni == ATni+1)

AT = ATni, x=x+1, z=z+1

Else

AT1 =AT ni+1, z=z+1

Break

5.) Compare New Priority of all the process which is having arrival time 0 and find Highest Priority (HP).

HP= NPni

for (i=1; i<=x; i++) x is frequency of arrival time 0

if (NPni < NPni+1)

HP = NPni+1

Execute the process which has Highest Priority.

6.) Execute this process until new process will not enter in queue.

for (i=1; i<AT1-AT; i++)

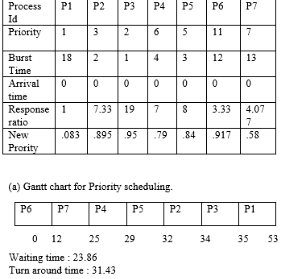
Execute P

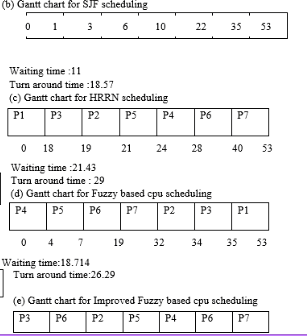
7.) Whenever new process is enter, break execution of this process and then again find highest priority among all entered processes.

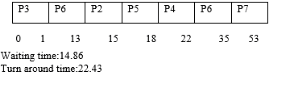
8.) After entering all processes in queue apply sorting method on New Priority to arrange in ascending order.

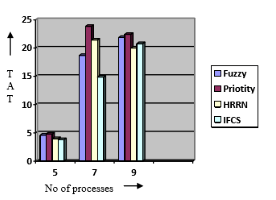
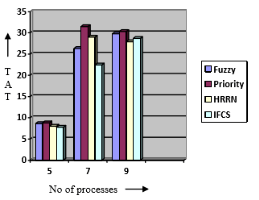
9.) Execute all processes as per assigned priority.

**A MODIFIED FUZZY BASED EXAMPLE:**







CONCLUSIONS

1) Previously, backtracking had been used for a Sudoku validator. It did do a ‘row check’, ’column check’ and ‘grid check’. But it did not give the results of these ‘checks’ simultaneously. Hence, it took a lot of time to check whether the Sudoku inputted was valid as a whole or not. However, with the help of threads, we were able to carry out these ‘checks’ in one simultaneous check and find out an algorithm that took much less time.

2) The proposed fuzzy based scheduling algorithm is an efficient scheduling algorithm that is obtained batter result rather than other algorithm. There is a comparison between SJF, Priority scheduling algorithm, Fuzzy based CPU scheduling algorithm and proposed new fuzzy based scheduling algorithm. The average waiting time and average turnaround time of proposed fuzzy based algorithm is much better than the Priority algorithm, Fuzzy based CPU scheduling algorithm and closer to obtain by SJF algorithm, but SJF algorithm doesn’t deal with task priority. Results prove that algorithm proposed in this project is much better than existing algorithms.