|  |
| --- |
| *Simple Multi-Threader* |
| **Contributors:**  **Vishal Kumar Maurya (2022580)**  **Subham Maurya (2022510)** |

**GitHub Link for the Repository (Private) :-**

[**https://github.com/vmaurya6622/OS-Assignments/tree/main/Asssignment-05**](https://github.com/vmaurya6622/OS-Assignments/tree/main/Asssignment-05)

Files Contained :- simple-multithreader.h , README , Documentation, vector.c and Makefile, matrix.c we have used deberian based (KALI linux) to complete our assignment and we have completed Bonus part also .

**Contribution by Vishal Kumar Maurya (2022580) :-** implemented the simplemuitithreader.h and made all the code for storing the thread data and running time of the thread. I have made the basic and advanced structures for parallel\_for function and improved the running cost. I have also learned lambda functionality from the internet and use it in the project. I have enhanced the functionality of the code using **GDB** debugger and made the code error free and more to the point. I have also created documentation and readme files to show our implementation.

**Contribution by Subham Maurya (2022580)** :- implemented the simple-multithreader.h and helped in creating parallel\_for 2D and 1\_D and improved the code. I have also debugged the code and improved the running environments of the threads. I have also read some relevant sources to improve my code and learn about the functionality of the lambda function and then implemented that in our code.

Click Here for the Source Code (Seen locally) :- [CLICK HERE](src.html)

*#include* <iostream>

*#include* <list>

*#include* <functional>

*#include* <stdlib.h>

*#include* <pthread.h>

*#include* <cstring>

*#include* <chrono> *//included extra can be removed used only for super precision of time*

int *user\_main*(int *argc*, char \*\**argv*);

struct ThreadData

{

    std::function*<*void(int)*>* *lambda*;

    int *low*;

    int *high*;

};

struct ThreadData2D

{

    std::function*<*void(int, int)*>* *lambda*;

    int *low1*;

    int *high1*;

    int *low2*;

    int *high2*;

};

static void \**parallel\_for\_helper*(void \**arg*)

{

    ThreadData *\*data* *=* *static\_cast<*ThreadData *\*>*(*arg*);

*for* (int *i* *=* *data*->*low*; *i* *<* *data*->*high*; *++i*)

    {

*data*->*lambda(i)*;

    }

*return* nullptr;

}

static void \**parallel\_for\_helper\_2d*(void \**arg*)

{

    ThreadData2D *\*data* *=* *static\_cast<*ThreadData2D *\*>*(*arg*);

*for* (int *i* *=* *data*->*low1*; *i* *<* *data*->*high1*; *++i*)

    {

*for* (int *j* *=* *data*->*low2*; *j* *<* *data*->*high2*; *++j*)

        {

*data*->*lambda(i*, *j)*;

        }

    }

*return* nullptr;

}

void *parallel\_for*(int *low*, int *high*, std::function<void(int)> &&*lambda*, int *numThreads*)

{

    auto *start* *=* std::chrono::high\_resolution\_clock::*now*();

    std ::*cout* *<<* "1D-Thread Creation Started\n";

*// Create threads*

    pthread\_t *threads*[*numThreads*];

    ThreadData *data*[*numThreads*];

    int *chunk* *=* (*high* *-* *low*)*/numThreads*;

*for* (int *i* *=* 0; *i* *<* *numThreads*; *++i*)

    {

*data*[*i*].*low=i\*chunk*; *data*[*i*].*high=*(*i+*1)*\*chunk*;

*data*[*i*].*lambda* *=* *lambda*;

*pthread\_create*(*&threads*[*i*], nullptr, *parallel\_for\_helper*,(void*\**)*&data*[*i*]);

    }

*// Join threads*

*for* (int *i* *=* 0; *i* *<* *numThreads*; *++i*)

    {

*pthread\_join*(*threads*[*i*], nullptr);

    }

    std ::*cout* *<<* "All 1D-Threads Completed\n";

    auto *end* *=* std::chrono::high\_resolution\_clock::*now*();

    std::chrono::duration*<*double*>* *duration* *=* *end* *-* *start*;

    std::*cout* *<<* "Execution time for parallel\_for(1D): " *<<* *duration*.*count*() *<<* " seconds\n";

}

*/\*Parallelizing for i\*/*

void *parallel\_for*(int *low1*, int *high1*, int *low2*, int *high2*, std::function<void(int, int)> &&*lambda*, int *numThreads*)

{

    auto *start* *=* std::chrono::high\_resolution\_clock::*now*();

    std ::*cout* *<<* "2D-Thread Creation Started\n";

*// Create threads*

    pthread\_t *threads*[*numThreads*];

    ThreadData2D *data*[*numThreads*];

    int *chunk* *=* (*high1-low1*)*/numThreads*;

*for* (int *i* *=* 0; *i* *<* *numThreads*; *++i*)

    {

*data*[*i*].*low1=i\*chunk*; *data*[*i*].*high1=*(*i+*1)*\*chunk*;

*data*[*i*].*low2=low2*; *data*[*i*].*high2=high2*;

*data*[*i*].*lambda* *=* *lambda*;

*pthread\_create*(*&threads*[*i*], nullptr, *parallel\_for\_helper\_2d*,(void*\**) *&data*[*i*]);

    }

*// Join threads*

*for* (int *i* *=* 0; *i* *<* *numThreads*; *++i*)

    {

*pthread\_join*(*threads*[*i*], nullptr);

    }

    std ::*cout* *<<* "All 2D-Threads Completed\n";

    auto *end* *=* std::chrono::high\_resolution\_clock::*now*();

    std::chrono::duration*<*double*>* *duration* *=* *end* *-* *start*;

    std::*cout* *<<* "Execution time for parallel\_for (2D): " *<<* *duration*.*count*() *<<* " seconds\n";

}

*/\*Parallelizing for j\*/*

*// void parallel\_for(int low1, int high1, int low2, int high2, std::function<void(int, int)> &&lambda, int numThreads)*

*// {*

*//     auto start = std::chrono::high\_resolution\_clock::now();*

*//     std ::cout << "2D-Thread Creation Started\n";*

*//     // Create threads*

*//     int loop\_1 = (high1-low1);*

*//     pthread\_t threads[numThreads \* loop\_1];*

*//     ThreadData2D data[numThreads \* loop\_1];*

*//     int chunk = (high2-low2)/numThreads;*

*//     int index = 0;*

*//     for (int i = 0; i < loop\_1; ++i)*

*//     {*

*//       for (int j = 0 ; j < numThreads ; j++)*

*//       {*

*//           data[index].low1=i; data[index].high1=(i+1);*

*//           data[index].low2=j\*chunk; data[index].high2=(j+1)\*chunk;*

*//           data[index].lambda = lambda;*

*//           pthread\_create(&threads[index], nullptr, parallel\_for\_helper\_2d,(void\*) &data[index]);*

*//           index++;*

*//       }*

*//     }*

*//     // Join threads*

*//     for (int i = 0; i < numThreads\*loop\_1; ++i)*

*//     {*

*//         pthread\_join(threads[i], nullptr);*

*//     }*

*//     std ::cout << "All 2D-Threads Completed\n";*

*//     auto end = std::chrono::high\_resolution\_clock::now();*

*//     std::chrono::duration<double> duration = end - start;*

*//     std::cout << "Execution time for parallel\_for (2D): " << duration.count() << " seconds\n";*

*// }*

*/\* Demonstration on how to pass lambda as parameter.*

*\* "&&" means r-value reference. You may read about it online.*

*\*/*

void *demonstration*(std::function<void()> &&*lambda*)

{

*lambda()*;

}

int *main*(int *argc*, char \*\**argv*)

{

*/\**

*\* Declaration of a sample C++ lambda function*

*\* that captures variable 'x' by value and 'y'*

*\* by reference. Global variables are by default*

*\* captured by reference and are not to be supplied*

*\* in the capture list. Only local variables must be*

*\* explicity captured if they are used inside lambda.*

*\*/*

    int *x* *=* 5, *y* *=* 1;

*// std ::cout<<"Hello Hi";*

*// Declaring a lambda expression that accepts void type parameter*

    auto */\*name\*/* *lambda1* *= /\*capture list\*/* [*/\*by value\*/* *x*, */\*by reference\*/* *&y*](void)

    {

*/\* Any changes to 'x' will throw compilation error as x is captured by value \*/*

*y* *=* 5;

        std::*cout* *<<* "====== Welcome to Assignment-" *<<* *y* *<<* " of the CSE231(A) ======\n";

*/\* you can have any number of statements inside this lambda body \*/*

    };

*// Executing the lambda function*

*demonstration*(*lambda1*); *// the value of x is still 5, but the value of y is now 5*

    int *rc* *=* *user\_main*(*argc*, *argv*);

    auto */\*name\*/* *lambda2* *=* [*/\*nothing captured\*/*]()

    {

        std::*cout* *<<* "====== Hope you enjoyed CSE231(A) ======\n";

*/\* you can have any number of statements inside this lambda body \*/*

    };

*demonstration*(*lambda2*);

*return* *rc*;

}

*#define* *main* *user\_main*