Operating System 112 Fall
Homework 3 -
Multi-thread & Mutex & Semaphore

Prof. 蔡文錦
TA 林浩君, 薛乃仁, 周彥昀, 許承壹

Hardware & Software Requirements

Hardware
CPU > 4 thread

Software
Bash
Python
C++20 if you want to use <semaphore>

Part 1: Sequential Output (25%) & Preview

Question:

Open 100 threads. For each thread, count from 0 to 1000000 and print a message when done.

Task:

Given hw3-1.cpp, modify it to ensure that the output of the thread index is in ascending order.

Hint:

Semaphore

hw3-1.cpp

```cpp
#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

void count(int index) {
  static mutex io_mutex;
  int num = 1000000;
  while (num--) {}
  {
    lock_guard<mutex> lock(io_mutex);
    cout << "I'm thread " << index << ", local count: 1000000\n";
  }
}

int main(void) {
  thread t[100];

  for (int i = 0; i < 100; i++)
    t[i] = thread(count, i);

  for (int i = 0; i < 100; i++)
    t[i].join();
}
```

Modify this part

Part 1: Scoring

Sequential output (25%)

output

```
I'm thread 1, local count: 1000000
I'm thread 32, local count: 1000000
I'm thread 26, local count: 1000000
I'm thread 3, local count: 1000000
...
```


output

```
I'm thread 0, local count: 1000000
I'm thread 1, local count: 1000000
I'm thread 2, local count: 1000000
I'm thread 3, local count: 1000000
...
```

Part 2: Maximal Independent Set (25%)

Question:
Find one of the maximal independent sets in graph G.

Task:
Given hw3-2.cpp, modify it to solve the problem without exceeding the time limit.

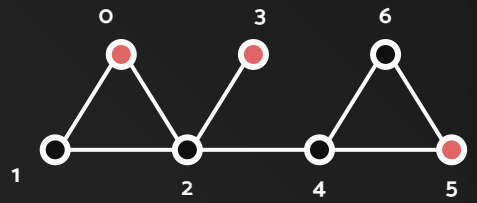
Solution:
For each vertex v, create a thread to check the status of its neighbors. If any neighbor is in the maximal independent set, remove v from the set. Otherwise, add v to the set. Repeat this process until every vertex has been checked and no further updates are made.

Part 2: Testcase 1

case1.txt

```
7 8
0 1
0 2
1 2
2 3
2 4
4 5
4 6
5 6
```

V = 7, E = 8
Then E edges with ascending lexicographic order.



output

```
0 3 5
```

Part 2: Preview

hw3-2.cpp

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#include <semaphore>
#include <chrono>
#include <random>

using namespace std;

constexpr int MAX_VERTICES = 750;
constexpr ptrdiff_t MAX_THREADS{4};

mutex mutexes[MAX_VERTICES];
counting_semaphore semaphores{MAX_THREADS};

// random number generator
random_device dev;
mt19937 rng(dev());
uniform_int_distribution<mt19937::result_type> dist(1, 10);

// num of vertices & num of edges
int V, E;
vector<int> adjacent_matrix[MAX_VERTICES];
```

```cpp
// vertex_check[i]:
//    0: not checked
//    1: checked
bool vertex_checked[MAX_VERTICES];

// vertex_status[i]:
//    0: not in independent set
//    1: in independent set
bool vertex_status[MAX_VERTICES];

// if every vertex is checked, then the graph is converged
bool is_converged() {
  for (int i = 0; i < V; i++)
    if (!vertex_checked[i])
      return false;
  return true;
}

// if any neighbor is in the set, then leave the set
bool best_response(int v) {
  for (int u : adjacent_matrix[v])
    if (vertex_status[u])
      return false;
  return true;
}
```

Part 2: Preview

Modify this part

```
void maximum_independent_set(int v) {
  bool converged = false;

  while (!converged) {
    if (vertex_checked[v]) {
      converged = is_converged();
    } else {
      bool old_response = vertex_status[v];
      vertex_status[v] = best_response(v);

      vertex_checked[v] = true;
      if (vertex_status[v] != old_response) {
        for (int u : adjacent_matrix[v])
          vertex_checked[u] = false;
      }
    }
  }
}
```

```
int main(void) {
  cin >> V >> E;
  thread t[V];

  for (int i = 0; i < V; i++) {
    vertex_checked[i] = false;
    vertex_status[i] = false;
  }
  for (int i = 0; i < E; i++) {
    int u, v;
    cin >> u >> v;
    adjacent_matrix[v].push_back(u);
    adjacent_matrix[u].push_back(v);
  }

  for (int i = 0; i < V; i++)
    t[i] = thread(maximum_independent_set, i);
  for (int i = 0; i < V; i++)
    t[i].join();

  for (int i = 0; i < V; i++) {
    if (!vertex_status[i])
      continue;
    cout << i << ' ';
  }
  cout << '\n';
}
```

Part 2: Hint - The Reasons of TLE...(Maybe)

TLE with high CPU load:
The vertex checked has not converged (Data Race) -> mutex

TLE with low CPU load:
Deadlock -> prevent conditions of deadlock

TLE with medium CPU load:
Busy waiting? -> random delays?

Other design problem:
https://stackoverflow.com/questions/71893279/why-does-stdcounting-semaphoreacquire-suffer-deadlock-in-this-case

Part 2: Scoring

Run test.sh

Pass all test cases (25%)

Part 3 & 4

1. For these two sections, you are required to optimize using threading techniques, starting from the serial version of the code. Any attempts to enhance performance through **complexity reduction**, providing **fixed answers**, or other non-threading methods will result in a score of zero.
2. We'll compile your code using "g++ hw3-3.cpp -lpthread" and "g++ hw3-4.cpp -lpthread".
3. We'll compute speedup by comparison with serial version code.
4. You need to implement a flag "-t" to indicate number of threads.
(part 3: 1 <= t <= 8, part4: t = {1, 2, 4, 8}).

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-4$ g++ hw3-4.cpp -lpthread
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-4$ ./a.out -t 4 < testcase/case3.txt
90020896
```

Part 3 & 4: Speed up

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-3$ g++ hw3-3_serial.cpp
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-3$ time ./a.out < testcase/case3.txt
362352

real    0m2.039s
user    0m2.000s
sys     0m0.036s
```

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-3$ g++ hw3-3.cpp -lpthread
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW3/hw3_answer/hw3-3$ time ./a.out -t 4 < testcase/case3.txt
362352

real    0m0.545s
user    0m2.161s
sys     0m0.000s
```

Speedup

= 2.039 / 0.545

= 3.74

Part 3: Counting Prime (25%) & Preview

Question:

Given a number N, find the total number of prime numbers less than or equal to N.

Task:

Given hw3-3_serial.cpp, write hw3-3.cpp to solve the problem with multithreading.

hw3-3_serial.cpp

```cpp
#include <iostream>

using namespace std;

int n, global_count = 0;

bool is_prime(int num) {
    if (num == 1) return false;

    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}

int main(int argc, char* argv[]) {
    cin >> n;

    for (int i = 1; i <= n; i++) {
        if (is_prime(i)) global_count++;
    }

    cout << global_count << endl;
    return 0;
}
```

Part 3: Scoring

5 test cases, if any of the test case fail **-> 0%**
Modify the algorithm **-> 0%**

2 times speedup using 4 threads on ”testcase/case3.txt” (**15%**)

You will compete against your classmates for the fastest program speedup. Participants will be divided into four groups, receiving scores of **10%**, **7%**, **4%**, and **1%**, respectively. We’ll also test this part in 4 threads with input size near ”testcase/case3.txt”.

Part 4: Set Covering Problem (25%)

Question:
Find the number of subset combinations where the union equals the universal set.

Task:
Given hw3-4_serial.cpp, write hw3-4.cpp to solve the problem with multithreading.

Solution:
For each subset, test the result of whether adding or not adding the subset.
Count the number of subset combinations where the union equals the universal set.

Part 4: Testcase 1

N (size of universal set): 2, M (number of subset): 3
Then M subset with p (size of subset) and followed by its elements.
S1: {0}
S2: {0, 1}
S3: {1}

There're 5 combinations union to the universal set
1. S2 -> {0, 1}
2. S1 ∪ S2 -> {0} ∪ {0, 1} = {0, 1}
3. S1 ∪ S3 -> {0} ∪ {1} = {0, 1}
4. S2 ∪ S3 -> {0, 1} ∪ {1} = {0, 1}
5. S1 ∪ S2 ∪ S3 -> {0} ∪ {0, 1} ∪ {1} = {0, 1}
So the answer is 5.

Part 4: Preview

hw3-4_serial.cpp

```cpp
#include <iostream>
#include <cstdint>
#include <vector>

using namespace std;

int n, m;
vector<uint64_t> subsets;
uint64_t one = static_cast<uint64_t>(1), global_count = 0;

void solve(int index, uint64_t current) {
    if (index == m) {
        if (current == (one << n) - 1) global_count++;
    } else {
        solve(index + 1, current);
        solve(index + 1, current | subsets[index]);
    }
}
```

```cpp
int main() {
    cin >> n >> m;

    subsets.resize(m);
    for (int i = 0; i < m; i++) {
        int p, temp;
        cin >> p;
        for (int j = 0; j < p; j++) {
            cin >> temp;
            subsets[i] |= (one << temp);
        }
    }

    solve(0, 0);

    cout << global_count << endl;
    return 0;
}
```

Part 4: Scoring

5 test cases, if any of the test case fail **-> 0%**
Modify the algorithm **-> 0%**

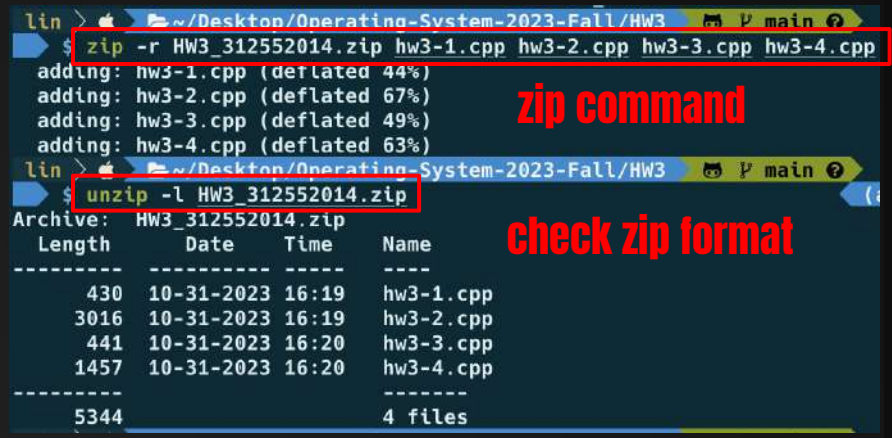
1.2 times speedup using 4 threads on ”testcase/case3.txt” (**15%**)

You will compete against your classmates for the fastest program speedup. Participants will be divided into four groups, receiving scores of **10%**, **7%**, **4%**, and **1%**, respectively. We’ll also test this part in 4 threads with input size near ”testcase/case3.txt”.

Submission and Rules

Submission:

1. You should write your code in C++
2. Please upload your homework in such format:
   - HW3_studentID.zip (e.g. HW3_312552014.zip)
     - hw3-1.cpp
     - hw3-2.cpp
     - hw3-3.cpp
     - hw3-4.cpp



Rule:

- No **plagiarism** is allowed, since the grade of this course is critical for **graduate program application in CS related field**, we will not pardon such behavior at all, so please be responsible to yourself. You can discuss with your classmates, but don't copy and paste.
- Incorrect filename / file format will get -10% point.
- Delayed submission will get -20% point per day.