**รายงานการออกแบบโปรแกรม**

**รายวิชา** ระบบปฏิบัติการ **รหัสวิชา** 01204332

**เสนอ**

อาจารย์ สรยุทธ กลมกล่อม

**ชื่อกลุ่ม A4OS**

**สมาชิกในกลุ่ม**

นายพงศกร สุตมา 6040202998

นางสาวจารุพร ทัดสี 6040200634

นายกิตติชัย นิวงษา 6040200278

นางสาวโฉมธิดา วงค์ตาแสง 6040200855

นางสาวลดาวัลย์ ใจหมาย 6040204095

นายศิรวิชญ์ คำสงค์ 6040204559

นายสาธิต ทรัพย์เมฆ 6040204923

นางสาววริศรา จรรยาวดี 6040206578

นางสาวธิติสุดา จิตตะยโศธร 6040202106

นางสาวสุภาวดี มาลา 6040205211

นายสถาพร สายืน 5940205046

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการสาขาวิศวกรรมคอมพิวเตอร์

ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตเฉลิมพระเกียรติ จังหวัดสกลนคร

**คำนำ**

รายงานเล่นนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการOS รหัส 01204332 เพื่อศึกษาการทำงานของ Mutual Exclution โดยการเขียนโปรแกรม Producer and Consumer เพื่อศึกษาการออกแบบโปรแกรม เงื่อนไข การทำงานของ Append และRemove และการใช้ Semaphore เพื่อควบคุมการทำงานของโปรแกรม

โดยหวังเป็นอย่างยิ่งว่าการจัดทำรายงานฉบับนี้จะเป็นประโยชน์กับผู้อ่าน ที่กำลังศึกษาหาข้อมูลเรื่องนี้อยู่ หากมีข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับและขออภัยมา ณ ที่นี้ด้วย

**สารบัญ**

**เนื้อหา หน้า**

คำนำ I

สารบัญ II

การออกแบบโปรแกรม 1

1.1การออกแบบ Buffer 1

1.2 การออกแบบ Append 2

1.3 การออกแบบ Remove 3

1.4 Flowchart ของโปรแกรม 4

2.เงื่อนไข วิธีการทำงาน และการพิสูจน์คุณสมบัติของ Append 5

2.1 เงื่อนไขของ Append 5

2.2 วิธีการทำงานของ Append 6

2.3 การพิสูจน์คุณสมบัติของ Append 7

3.เงือนไข และวิธีการทำงานของ Remove 8

3.1 เงื่อนไขของ Remove 9

3.2 วิธีการทำงานของ Remove 9

4.ผลการ Run & Result 10

5.Sourcecode ของโปรแกรม 10

อ้างอิง

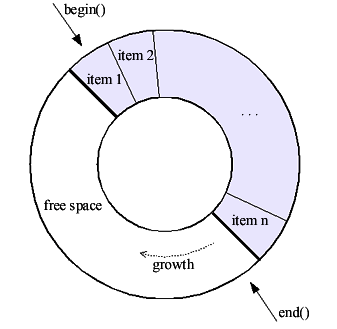
# 1.การออกแบบโปรแกรม

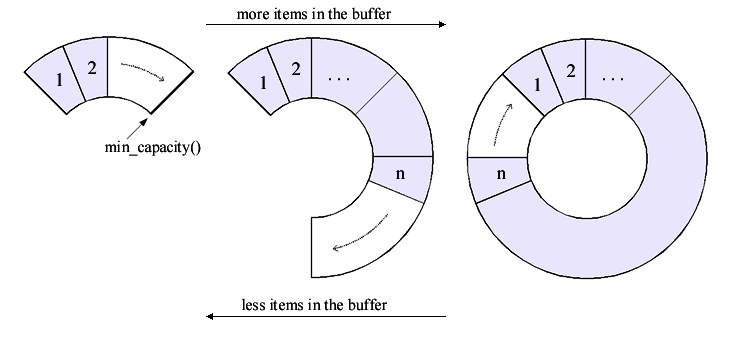
## 1.1 การออกแบบ Buffer

1. #include <boost/circular\_buffer.hpp>
2. boost::circularbuffer<int> buffername(buffer size);

**บัฟเฟอร์แบบวงกลม โดยใช้ Boost Library**

บัฟเฟอร์แบบวงกลม (เรียกอีกอย่างว่าวงแหวนหรือบัฟเฟอร์แบบวงกลม) หมายถึงพื้นที่ในหน่วยความจำซึ่งใช้เพื่อเก็บข้อมูลที่เข้ามา เมื่อบัฟเฟอร์เต็มข้อมูลใหม่จะถูกเขียนเริ่มต้นที่จุดเริ่มต้นของบัฟเฟอร์และเขียนทับเก่า **boost :: circular\_buffer** เป็นคอนเทนเนอร์ที่สอดคล้องกับ STL มันเป็นชนิดของลำดับที่คล้ายกับ std :: list หรือ std :: deque สนับสนุนการเข้าถึงตัววนซ้ำแบบสุ่มแทรกเวลาอย่างต่อเนื่องและลบการดำเนินการที่จุดเริ่มต้นหรือจุดสิ้นสุดของบัฟเฟอร์และการทำงานร่วมกันกับอัลกอริทึม std Circular\_buffer ได้รับการออกแบบมาเป็นพิเศษเพื่อให้มีความจุคงที่ เมื่อความจุของมันหมดลงองค์ประกอบที่เพิ่งแทรกเข้าไปใหม่จะทำให้องค์ประกอบถูกเขียนทับทั้งที่จุดเริ่มต้นหรือจุดสิ้นสุดของบัฟเฟอร์ (ขึ้นอยู่กับการใช้งานการแทรก) Circular\_buffer จัดสรรหน่วยความจำเฉพาะเมื่อสร้างขึ้นเมื่อความจุถูกปรับอย่างชัดเจนหรือตามความจำเป็นเพื่อปรับขนาดหรือกำหนดการดำเนินการ





## 1.2 การออกแบบ Append

Append ฟังก์ชัน รูปร่างหน้าตาของ append

void append\_(int id)

{

++num\_append\_working;

while (i < REQUEST)

{

add\_item(id);

++i;

}

this\_thread::sleep\_for(chrono::nanoseconds(wait\_time));

--num\_append\_working;

}

ฟังก์ชัน append จะเพิ่มค่าของ num\_append\_work แล้วทำการเพิ่มค่าลงไปในบัฟเฟอร์คิว ตามจำนวนของการ รีเควส โดยการให้เทรดที่เรียกใช้งาน append ทำงานในฟังก์ชัน add\_item และเมื่อทำงานเสร็จสิ้นจะหยุดการทำงานของเทรดที่เรียก append และ ลดค่าของ num\_append\_working ลง

**ฟังก์ชั่น add\_item**

add\_item ฟังก์ชัน ที่ถูกเรียกใช้งานโดย เทรดต่าง ๆ ของ append เพื่อนำค่าเข้าไปในคิว

void add\_item(int append\_id)

{

int random\_num = rand() % 100;

unique\_lock<mutex> lock(ymutex);

is\_not\_full.wait(lock, [] { return myringbuf.size() != BUFFER\_SIZE; });

myringbuf.push\_back(random\_num);

// cout << "Append ID : " << append\_id << " add " << random\_num << endl;

is\_not\_empty.notify\_all();

}

add\_item() ออกแบบให้สามารถเข้าใช้งานได้เพียงเทรดเดียวเท่านั้นโดยค่าที่ จะถูกส่งเข้ามาเป็นค่า thread id ของ append thread จากนั้นจะใช้ mutex ล็อกทรัพยากรบัฟเฟอร์ให้เทรดนั้น ๆ เพื่อไม่ให้เทรดอื่น ๆ ที่รอคอยอยู่เข้ามาทำงานได้ แล้วจะมีการทำงานอยู่ 2 แบบดังนี้

1.) **บัฟเฟอร์เต็ม** ก็จะทำการหยุดการทำงานเทรดนั้นชั่วขณะ และส่งสัญญาณว่าคิวไม่ว่างออกไป

2.) **บัฟเฟอร์ไม่เต็ม** ก็จะทำการเรียกดูขนาดของคิวมาเก็บไว้ในตัวแปร และ เขียนค่าตัวแปรนั้นลง ไปในคิว และเมื่อทำงานเสร็จก็จะส่งสัญญาณออกไปว่าคิวไม่ว่างแล้ว

## 1.3 การออกเเบบ Remove

remove\_ ฟังก์ชัน หน้าตาของ remove

void remove\_(int id)

{

while (num\_append\_working == 0)

{

this\_thread::yield();

}

while (num\_append\_working != 0 || myringbuf.size() > 0)

{

remove\_item(id);

this\_thread::sleep\_for(chrono::nanoseconds(wait\_time));

}

}

ฟังก์ชัน remove จะตรวจสอบว่าไม่มีเทรด append ทำงานอยู่หรือบัฟเฟอร์ไม่ว่างก็จะเรียกการทำงานของ remove\_item เพื่อนำคิวออกจากบัฟเฟอร์จากนั้นก็จะหยุดการทำงานของเทรดนั้นลง

**ฟังก์ชั่น remove\_item**

remove\_items ฟังก์ชันที่ถูกเรียกใช้งานโดย เทรดต่าง ๆ ของ remove\_ เพื่อนำค่าออกจากคิว

void remove\_item(int remove\_id)

{

unique\_lock<mutex> lock(ymutex);

int product;

if (is\_not\_empty.wait\_for(lock, chrono::nanoseconds(wait\_time),

[] { return myringbuf.size() > 0; }))

{

product = myringbuf.front();

myringbuf.pop\_front();

++c\_count;

// cout << "Remove ID : " << remove\_id << " remove " << product << endl;

is\_not\_full.notify\_all();

}

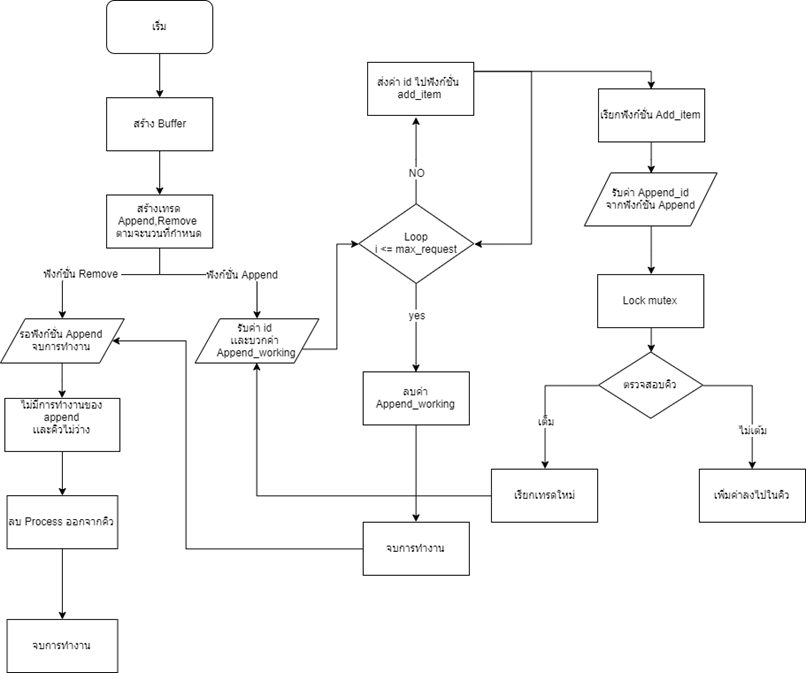
}

ฟังก์ชัน remove\_item จะมีการเข้าถึงฟังก์ชันนี้ได้เพียงแค่เทรดเดียวเท่านั้น เมื่อทำการเรียกใช้งานฟังก์ชั่นก็จะใช้ mutex ล็อกทรัพยาการบัฟเฟอร์ให้เข้าถึงได้เพียงแค่เทรดเดียวและมีการทำงานอยู่ 2 แบบดังนี้

1.) **บัฟเฟอร์ว่าง** จะหยุดการทำงานเทรดนั้น ๆ และส่งสัญญาณออกไปให้เทรดอื่น ๆ ที่รอทำงาน

2.) **บัฟเฟอร์ไม่ว่างหรือเต็ม** จะไอเท็มที่หน้าคิวและทำการ dequeue ออกจากบัฟเฟอร์ และส่งสัญญาณให้เทรดอื่น ๆ ที่รอทำงาน

## 1.4 Flowchart ของโปรแกรม



# 2.เงื่อนไข วิธีการทำงาน และการพิสูจน์คุณสมบัติของ Append

ฟังก์ชั่น Append

void append\_(int id)

{

++num\_append\_working;

while (i < REQUEST)

{

add\_item(id);

++i;

}

this\_thread::sleep\_for(chrono::nanoseconds(wait\_time));

--num\_append\_working;

}

## 2.1 เงื่อนไขของ Append

เงื่อนไขการทำงานของ append

1.) ทำงานตามรีเควสที่กำหนด

2.) เข้าถึงบัฟเฟอร์ได้เพียงเทรดเดียว

3.) ทุกเทรดมีเวลาทำงานจำกัดในการทำงาน

## 2.2 วิธีการทำงานของ Append

1.) เพิ่มค่าของ num\_append\_working ขึ้นตามจำนวนเทรดที่ใช้งาน

2.) วนลูปทำงานตามรีเควสที่ได้กำหนดไว้

3.) เรียกใช้งาน add\_item โดยส่งค่า Append\_id ไปให้

- รับค่าของ append id

- สุ่มตัวเลขและเก็บไว้ในตัวแปร

- ล็อก metex ให้เทรดที่เรียกใช้งาน

- ตรวจสอบบัฟเฟอร์

is\_not\_full.wait(lock, [] { return myringbuf.size() != BUFFER\_SIZE; });

เมื่อบัฟเฟอร์ไม่เต็มก็จะไม่ถูกเรียกใช้คำสั่ง wait() แต่หากบัฟเฟอร์ เต็มเทรดนั้น ๆ ก็จะถูกเปลี่ยนสถานะให้รอจนกว่าจะได้รับสัญญาน

- เพิ่มค่าของ random\_num เข้าไปที่ท้ายคิวของบัฟเฟอร์

- แสดงผลการเพิ่ม

- ส่งสัญญาณให้ทุกเทรดที่รอการทำงานว่าบัฟเฟอร์ไม่ว่าง

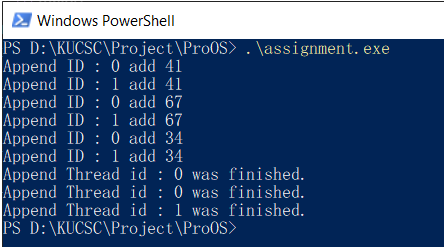
4.) เพิ่มค่า i

5.) เมื่อเทรดใดเทรดหนึ่งทำงานถึงระยะเวลาที่ได้กำหนดไว้ใน wait\_time ก็จะหยุดการทำงานของเทรดลงไป

6.) ลดค่าของ num\_append\_working ลงตามจำนวนเทรดที่ปิดใช้งานไป

## การพิสูจน์คุณสมบัติของ Append

ผลการทดสอบคุณสมบัติของ append โดยการขอรีเควสไปทำ 6 ตัว และ เทรดของ append 3 เธรด



# 3. เงื่อนไขการทำงานของ Remove

ฟังก์ชัน remove

void remove\_(int id)

{

while (num\_append\_working == 0)

{

this\_thread::yield();

}

while (num\_append\_working != 0 || myringbuf.size() > 0)

{

remove\_item(id);

this\_thread::sleep\_for(chrono::nanoseconds(wait\_time));

}

cout << "Remove thread id : " << id << " was finished.\n" << endl;

}

## 3.1 เงื่อนไขของ Remove

1.) ตรวจสอบการทำงานของ append ว่าไม่มีการทำงานอยู่

2.) ตรวจสอบบัฟเฟอร์ว่าไม่ได้ว่างอยู่

3.) เข้าถึงบัฟเฟอร์ได้เพียงเทรดเดียว

4.) ทุกเทรดที่ทำงานมีเวลาจำกัดในการทำงาน

## 3.2 วิธีการทำงานของ Remove

1.) ตรวสอบว่าเทรดของ append ไม่ได้มีการทำงานอยู่ โดยตรวจสอบจากตัวแปร num\_append\_working

2.) หากไม่มีเทรดใดของ append ทำงานอยู่แล้ว ให้ทำการรอชั่วขณะ

3.) หากบัฟเฟอร์มีขนาดมากกว่า 0 ให้ทำการเรียกใช้งาน remove\_item พร้อมส่งค่า id ของเทรดไปด้วย

- รับค่า thread id

- ล็อกการทำงานของ mutex ให้เทรดที่เรียกใช้งาน

- ตรวจสอบว่าบัฟเฟอร์ไม่ว่าง

if (is\_not\_empty.wait\_for(lock, chrono::nanoseconds(wait\_time),

[] { return myringbuf.size() > 0; })

- เลือกตำแหน่งต้นคิว และ ลบออกจากคิว

- เพิ่มค่า c\_count

- แสดงผลการทำงาน

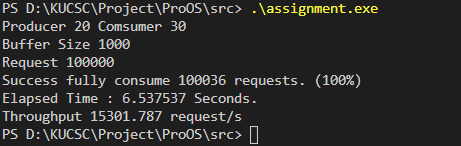
- ส่งสัญญาณว่าบัฟเฟอร์ไม่เต็มออกไปให้ทุกเทรดที่รอการทำงาน

4.) หากเทรดทำงานจนถึงเวลาที่กำหนดให้หยุดการทำงานของเทรดนั้น ๆ

5.) แสดงผลการทำงาน

## ผลการรันทดสอบ remove

# 4.ผลการ Run & Result



# 5. Source code

/\*

    คอมไพล์ไฟล์นี้โดยใช้คำสั่ง : g++ ชื่อไฟล์.cpp -pthread -lpthread -o ชื่อโปรแกรมที่ต้องการ

    รันโปรแกรม : ./ชื่อโปรแกรม

    คอมไพล์เลอร์ : MinGW-W64

    ลิงก์ดาวน์โหลด : https://sourceforge.net/projects/mingw-w64/

    ติดตั้งและเซตพาธให้เรียบร้อย

    ไลบราลี่ที่จำเป็น : Boost C++ Libraly

    ลิงก์ดาวน์โหลด : https://www.boost.org/

\*/

#include <iostream>

#include <mutex>

#include <thread>

#include <condition\_variable>

#include <atomic>

#include <vector>

#include <chrono>

#include <bits/stdc++.h>

#include <boost/circular\_buffer.hpp>

//or using this include if you don't install boost libraly//

#include "circular\_buffer.hpp" //<- uncomment this line

using namespace std;

#define BUFFER\_SIZE 1000

#define REQUEST 100000

#define PROD 20

#define CONS 30

int c\_count = 0;

int i = 0;

const int wait\_time = 1;

atomic<int> num\_append\_working(0);

condition\_variable is\_not\_full;

condition\_variable is\_not\_empty;

mutex xmutex;

boost::circular\_buffer<int> myringbuf(BUFFER\_SIZE);

void add\_item(int append\_id)

{

    int random\_num = rand() % 100;

    unique\_lock<mutex> lock(xmutex);

    is\_not\_full.wait(lock, [] { return myringbuf.size() != BUFFER\_SIZE; });

    myringbuf.push\_back(random\_num);

    // cout << "Append ID : " << append\_id << " add " << random\_num << endl;

    is\_not\_empty.notify\_all();

}

void remove\_item(int remove\_id)

{

    unique\_lock<mutex> lock(xmutex);

    int product;

    if (is\_not\_empty.wait\_for(lock, chrono::milliseconds(wait\_time),

                              [] { return myringbuf.size() > 0; }))

    {

        product = myringbuf.front();

        myringbuf.pop\_front();

        ++c\_count;

        // cout << "Remove ID : " << remove\_id << " remove " << product << endl;

        is\_not\_full.notify\_all();

    }

}

void append\_(int id)

{

    ++num\_append\_working;

    while (i < REQUEST)

    {

        add\_item(id);

        ++i;

    }

    this\_thread::sleep\_for(chrono::milliseconds(wait\_time));

    // cout << "Append Thread id : " << id << " was finished." << endl; // <-- uncomment to display end append\_s thread

    --num\_append\_working;

}

void remove\_(int id)

{

    while (num\_append\_working == 0)

    {

        this\_thread::yield();

    }

    while (num\_append\_working != 0 || myringbuf.size() > 0)

    {

        remove\_item(id);

        this\_thread::sleep\_for(chrono::milliseconds(wait\_time));

    }

    // cout << "Remove thread id : " << id << " was finished." << endl;

}

int main()

{

    // Record Start Time //

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

    // unsync the I/O of C and C++.

    ios\_base::sync\_with\_stdio(false);

    // Start Producer Consumer Program //

    cout << "Producer " << PROD

         << " Comsumer " << CONS

         << "\nBuffer Size " << BUFFER\_SIZE

         << "\nRequest " << REQUEST

         << endl;

    vector<thread> vec\_of\_thread;

    for (int i = 0; i < PROD; i++)

    {

        vec\_of\_thread.push\_back(thread(append\_, i));

    }

    for (int i = 0; i < CONS; i++)

    {

        vec\_of\_thread.push\_back(thread(remove\_, i));

    }

    for (auto &t : vec\_of\_thread)

    {

        t.join();

    }

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

    // Calculating total time taken by the program.

    double time\_taken =

        chrono::duration\_cast<chrono::nanoseconds>(end - start).count();

    time\_taken \*= 1e-9; // Chang nanosecond to second

    double through\_put = c\_count / time\_taken;

    double percentage = (c\_count / REQUEST) \* 100.0;

    string c\_consume = to\_string(c\_count);

    cout << "Success fully consume " << c\_consume

         << " requests. (" << percentage

         << "%)" << endl;

    cout << "Elapsed Time : " << fixed << time\_taken << setprecision(3)

         << " Seconds." << endl;

    cout << "Throughput " << through\_put << " request/s" << endl;

    return 0;

}

**อ้างอิง**

**std::mutex**. cppreference. https://en.cppreference.com/w/cpp/thread/mutex

Ankit Lathiya. (2019). **Multithreading In C++ Example | C++ Multithreading Tutorial.** Appdividend. https://appdividend.com/2019/08/21/multithreading-in-cpp-example-cpp-multithreading-tutorial/

**Tutorials**. cplusplus. http://www.cplusplus.com/  
**producer–consumer problem**. wikipedia. https://en.wikipedia.org/wiki/Producer%E2%80%93consumer\_problem