



รายงาน การออกแบบ

จัดทำโดย

นางสาวกนกวรรณ บัวภาคำ 6040200049

นายจารุเดช ก่อรักเสวต 6040200618

นายธรรมรัช ตันติไพฑรรม 6040201983

นางสาวชญลักษณ์ โภคธนาพิพัฒน์พร 6040202033

นายธีระวัฒน์ ชรินทร์ 6040202203

นายณฐปราชญ์ ศักดิ์ชัยพานิชกุล 6040202343

นายพีรพล วรรณพันธ์ 6040203412

นางสาวเพ็ญรติ โพนเมืองหล้า 6040203463

นายภาณุภูมิ วาควงศ์ 6040203609

นางสาวรัตดาพร อักษรทอง 6040204010

นายอนุชา ศรีลาแก้ว 6040205407

นายอำนาจ ทนงนวล 6040205938

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตเฉลิมพระเกียรติ จังหวัดสกลนคร

ภาคต้น ปีการศึกษา 2562

คำนำ

รายงานการออกแบบวิชาการระบบปฏิบัติการคอมพิวเตอร์ รหัสวิชา ๐๑๒๐๔๓๓๒
หมู่เรียน ๑ มีจุดประสงค์เพื่อรายงานผลการวิเคราะห์การออกแบบ
ซึ่งรายงานฉบับนี้มีเนื้อหาเกี่ยวกับการออกแบบ เงื่อนไข วิธีการทำงานของฟังก์ชันต่างๆ
อธิบายความถูกต้องของโปรแกรม ผลลัพธ์ของการรันโปรแกรม ซอร์สโค้ดของโปรแกรม
คณะผู้จัดทำได้ศึกษาค้นคว้าข้อมูลของบัฟเฟอร์แบบวนกลับ เทรด การเขียนฟังก์ชัน
append remove และ buff ตามโจทย์ที่อาจารย์ผู้สอนได้ให้มา จากแหล่งข้อมูลต่างๆ
การจัดทำรายงานฉบับนี้สำเร็จตามวัตถุประสงค์ไปด้วยดี
ทั้งนี้ทางคณะผู้จัดทำได้หวังเป็นอย่างยิ่งว่าการศึกษาค้นคว้าข้อมูลจะมีประโยชน์ต่อผู้ที่ได้มาศึกษา
ต่อเป็นอย่างดี หากมีสิ่งที่จะต้องปรับปรุงแก้ไขเพิ่มเติมประการใด
คณะผู้จัดทำน้อมรับฟังคำแนะนำ
ข้อเสนอแนะที่มีต่อเนื้อหาเพื่อนำไปปรับปรุงเอกสารรายงานให้สมบูรณ์ยิ่งขึ้น

๒ พฤศจิกายน ๒๕๖๒

คณะผู้จัดทำ

สารบัญ

เนื้อหา	หน้า
คำนำ.....	I
สารบัญ.....	II
สารบัญภาพ.....	III
1. การออกแบบโปรแกรม.....	
1.1 การออกแบบ Buffer.....	
1.2 การออกแบบ Append.....	
1.3 การออกแบบ Remove.....	
1.4 Flowchart ของโปรแกรม.....	
2. เงื่อนไข วิธีการทำงาน และการพิสูจน์คุณสมบัติของ Append.....	
2.1 เงื่อนไขของ Append.....	
2.2 วิธีการทำงานของ Append.....	
2.3 การพิสูจน์คุณสมบัติของ Append.....	
3. เงื่อนไข และวิธีการทำงานของ Remove.....	
3.1 เงื่อนไขของ Remove.....	
3.2 วิธีการทำงานของ Remove.....	
4. ผลการ Run & Result.....	
5. ซอร์สโค้ดของโปรแกรม.....	
เอกสารอ้างอิง.....	

สารบัญภาพ

เนื้อหา	หน้า
รูปที่ 1 อธิบายการทำงานของโปรแกรม	7
รูปที่ 2 Flowchart อธิบายการทำงานของ Append	8

1. การออกแบบโปรแกรม

1.1 การออกแบบ Buffer

Buffer ที่ออกแบบเป็นประเภท Circular Buffer โดย Implement เป็น Array of Boolean โดยแรกเริ่ม Buffer ตั้งต้นมีค่าเป็น 0 (No Item) และมี Integer 2 จำนวน เสมือนเป็น Pointer ที่ตำแหน่ง Head กับ Tail ทั้งสองจะถูกกำหนดให้อยู่ในตำแหน่งเดียวกัน คือ Index ที่ 0 เมื่อมีการ Add Item จะเปลี่ยนค่า Buffer ในตำแหน่งที่ Tail ขึ้นเป็น 1 และขยับตำแหน่งของ Tail ไปในตำแหน่งถัดไป ในทำนองเดียวกัน เมื่อมีการ Remove Item จะเปลี่ยนค่า Buffer ในตำแหน่งที่ Head ขึ้นเป็น 0 และขยับตำแหน่งของ Head ไปในตำแหน่งถัดไปเช่นกัน เมื่อ Head หรือ Tail ถึงตำแหน่งสุดท้ายของ Array ในการประมวลผลครั้งต่อไปจะถูกขยับกลับมาเริ่มใหม่ในตำแหน่งแรกเริ่ม

มี Producer และ Consumer ซึ่งเป็น Thread ที่ใช้งาน Buffer โดย Producer จะเรียกใช้งาน Function Append ที่เพิ่มข้อมูลลงใน Buffer ในตำแหน่งที่ Tail นี้อยู่ และ Consumer จะเรียกใช้งาน Function Remove ที่ลบข้อมูลออกจาก Buffer ในตำแหน่งที่ Head ขึ้นอยู่

Buffer

มีการวัดประสิทธิภาพการทำงานโดยการจับเวลาตั้งแต่เริ่มทำงานจนจบการทำงาน

มีการนับจำนวน Request

ทั้งหมดที่ทำงานสำเร็จแล้วคำนวณออกมาเป็นเปอร์เซ็นต์และอัตราการทำงานต่อวินาทีได้

1.2 การออกแบบ Append

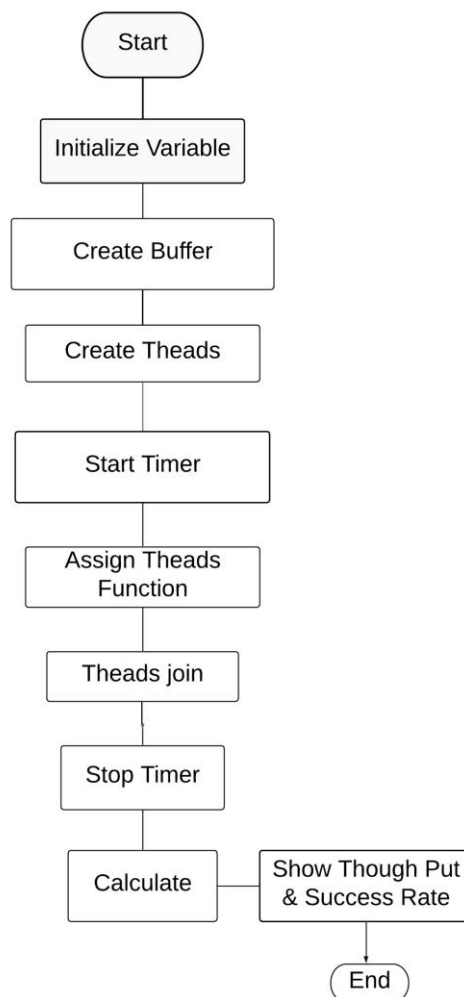
Append ทำหน้าที่ Add item ลงใน Buffer กรณีที่ Buffer เต็มจะให้ Request ที่เข้ามารอจนกว่า Buffer จะว่าง หรือ TimeOut และเนื่องจาก Producer อาจจะมีแค่ 1 Thread เพื่อป้องกันการถูกทับซ้อนกันของ Buffer ดังนั้นเมื่อมีการเข้าสู่การทำงานของ Append จะมีการ Lock ไว้ด้วย โดยมีการใช้คุณสมบัติของ Mutex จนกว่าฟังก์ชันของ Append

จะจบการทำงานหรือจำนวน Requeat หหมดจะ Unlock เพื่ออนุญาตให้ Thread อื่นสามารถเข้าใช้ Buffer ได้ต่อไป

1.3 การออกแบบ Remove

Remove ทำหน้าที่ Remove Item ออกจาก Buffer กรณีที่ Buffer ว่าง จะไม่เกิดการ Remove ขึ้น เมื่อเข้าสู่การทำงานของ Remove จะมีการล็อกไว้ด้วย โดยใช้คุณสมบัติของ Muter จนกว่าฟังก์ชัน Remove จบการทำงาน หรือ จำนวน Request หหมด และ Buffer ว่าง

1.4 Flowchart ของโปรแกรม



รูปภาพที่ 1 อธิบายการทำงานของโปรแกรม

2. เงื่อนไข วิธีการทำงาน และการพิสูจน์คุณสมบัติของ Append

2.1 เงื่อนไขของ Append

```
void *append_buffer() { //รับฟังก์ชัน append_buffer
    printf("Append thread number %ld\n", pthread_self()); //print ถ้า thread ที่รับใช้
    while(request<REQUEST) { //ถ้า request<REQUEST จะทำ loop while
        if(!pthread_mutex_trylock(&mutex) && request<REQUEST) { //สิทธิ์ในการแก้ไขบัฟเฟอร์ ถ้าไม่ตรงตามเงื่อนไขจะล็อค
            if(buffer[head] == 0) {
                add_item();
                request++;
                printf(" + thread %ld append success\n", pthread_self());
            }
            else { //ถ้าไม่ใช่ จะเกิด buffer overflow
                printf("Buffer overflow\n");
            }
            pthread_mutex_unlock(&mutex); //ปลดล็อค mutex
        }
    }
    pthread_exit(NULL);
}
```

รูปภาพที่ 2 คือเงื่อนไขของ Append

2.2 วิธีการทำงานของ Append

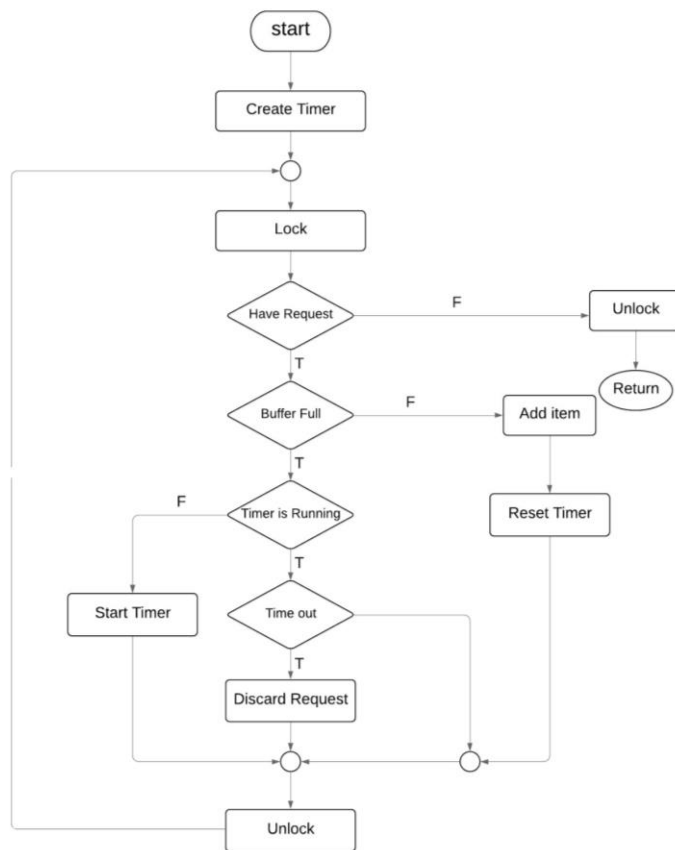
เมื่อทำการเลือกฟังก์ชัน Add_item

โปรแกรมจะเพิ่มรายการที่เรากรอกเข้าไปที่ท้ายแถวของบัฟเฟอร์ โดยสามารถเพิ่มได้ N

รายการเมื่อเพิ่มรายการจนถึง N

รายการแล้วจะไม่สามารถเพิ่มรายการได้อีกแล้ว โปรแกรมจะแจ้งว่า Buffer is overflow

ซึ่งจะต้องรอให้บัฟเฟอร์มีพื้นที่ว่างก่อนถึงจะสามารถเรียกใช้ฟังก์ชัน Add_item ได้อีก



รูปภาพที่ 2 Flowchart อธิบายการทำงานของ Append

2.3 การพิสูจน์คุณสมบัติของ Append

ผู้จัดทำพิสูจน์คุณสมบัติของ Append โดยการ กำหนดค่า Buffer และ Remove มีจำนวนน้อยลง กำหนดค่า Producer มีจำนวนมากขึ้น ทั้งนี้เพื่อให้การ Run

โปรแกรมมีโอกาสเกิด TimeOut จากจำนวนของ Buffer เต็มมากขึ้น แสดงลักษณะ ดังรูป

```
PRODUCERS:
30
Consumer:
30
Buffer:
10
Request:
100
Append thread number 1
Append thread number 2
Append thread number 3
+ thread 1 append success
+ thread 2 append success
Append thread number 6
Append thread number 5
+ thread 3 append success
Append thread number 4
Append thread number 7
Append thread number 8
Append thread number 9
+ thread 6 append success
Append thread number 10
Append thread number 11
Append thread number 12
Append thread number 13
+ thread 7 append success
+ thread 5 append success
+ thread 8 append success
+ thread 6 append success
Append thread number 14
+ thread 10 append success
Append thread number 15
Append thread number 16
+ thread 5 append success
Append thread number 17
Append thread number 18
Append thread number 19
Append thread number 21
Append thread number 22
Append thread number 23
Append thread number 24
Append thread number 25
Append thread number 26
Append thread number 27
Append thread number 20
Buffer overflow
Append thread number 28
Append thread number 29
Remove thread number 31
Remove thread number 32
Remove thread number 33
Remove thread number 34
Remove thread number 36
Remove thread number 37
Remove thread number 35
Buffer overflow
- thread 31 remove success
Remove thread number 39
Remove thread number 41
Remove thread number 50
Remove thread number 42
Remove thread number 43
Remove thread number 44
Remove thread number 45
Remove thread number 46
Remove thread number 48
Remove thread number 38
Remove thread number 47
+ thread 25 append success
Remove thread number 49
Remove thread number 40
Remove thread number 53
Remove thread number 52
Remove thread number 54
Remove thread number 56
Remove thread number 55
Remove thread number 57
Remove thread number 51
Remove thread number 58
Buffer overflow
Remove thread number 59
Remove thread number 60
Buffer overflow
Buffer overflow
Buffer overflow
- thread 45 remove success
+ thread 3 append success
Buffer overflow
Buffer overflow
- thread 55 remove success
+ thread 12 append success
- thread 58 remove success
+ thread 20 append success
- thread 43 remove success
- thread 55 remove success
- thread 51 remove success
- thread 58 remove success
+ thread 3 append success
+ thread 2 append success
```

```
- thread 32 remove success
+ thread 22 append success
+ thread 6 append success
- thread 51 remove success
+ thread 14 append success
- thread 50 remove success
+ thread 4 append success
Buffer overflow
Buffer overflow
Buffer overflow
Buffer overflow
- thread 40 remove success
- thread 43 remove success
- thread 35 remove success
- thread 54 remove success
+ thread 1 append success
+ thread 22 append success
+ thread 30 append success
+ thread 9 append success
- thread 58 remove success
+ thread 3 append success
- thread 58 remove success
+ thread 21 append success
Buffer overflow
Buffer overflow
- thread 45 remove success
+ thread 30 append success
Buffer overflow
- thread 43 remove success
- thread 44 remove success
+ thread 15 append success
- thread 41 remove success
- thread 55 remove success
- thread 42 remove success
+ thread 4 append success
+ thread 3 append success
+ thread 8 append success
- thread 55 remove success
- thread 36 remove success
- thread 51 remove success
- thread 56 remove success
+ thread 7 append success
- thread 37 remove success
+ thread 24 append success
- thread 60 remove success
+ thread 30 append success
+ thread 3 append success
- thread 46 remove success
- thread 53 remove success
- thread 46 remove success
```

3. เงื่อนไข และวิธีการทำงานของ Remove

3.1 เงื่อนไขของ Remove

```
void *remove_item() { //สร้างฟังก์ชัน remove_item  
  
    buffer[tail++] = 0; //buffer[tail++] มีค่า=0  
  
    tail = tail % BUFFER_SIZE; //tail มีค่าเท่ากับ tail / buffer_size เอาแต่เศษ
```

3.2 วิธีการทำงานของ Remove

เมื่อทำการเลือกฟังก์ชัน Remove_item โปรแกรมจะลบรายการในบัฟเฟอร์ออก โดยจะลบรายการที่อยู่หน้าแถวออกก่อน กล่าวคือ โปรแกรมจะทำการลบรายการที่ถูกเพิ่มเข้ามาในบัฟเฟอร์ก่อน และถ้าลบข้อมูลในบัฟเฟอร์จนหมดแล้ว โปรแกรมจะแจ้งว่า No buffer underflow คือไม่มีรายการเหลืออยู่ในบัฟเฟอร์แล้ว และจะต้องรอให้บัฟเฟอร์ไม่ว่างถึงจะเรียกฟังก์ชัน Remove_item ได้

4. ผลการ Run & Result

สามารถอธิบายผลการ Run & Result ได้ดังรูปที่

```
Producers 20, Consumers 30  
Buffer size 1000  
Requests 100000  
  
Successfully consumed 100000 requests (100.0%)  
Elapsed Time 157.09 s  
Throughput 636.59 successful requests/s  
  
-----  
Process exited after 178.2 seconds with return value 0  
Press any key to continue . . .
```

5. ซอร์สโค้ดของโปรแกรม

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

void *add_item();
void *remove_item();
void *append_buffer();
void *remove_buffer();
int i;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int PRODUCERS, CONSUMERS, BUFFER_SIZE, REQUEST;
int buffer[100000];
int tail = 0, head = 0, request = 0, success = 0;
clock_t timer1, timer2;
// ระบบตัวแปรที่ฟังก์ชันจะส่งคืน
// ฟังก์ชัน main รับ int
int main (int argc, char*argv)
{
    printf("PRODUCERS:\n") ;
    scanf("%d", &PRODUCERS);
    printf("Consumer:\n");
    scanf("%d", &CONSUMERS);
    printf("Buffer:\n");
    scanf("%d", &BUFFER_SIZE);
    printf("Request:\n");
    scanf("%d", &REQUEST);
```

```

timer1 = clock(); //วัด clock1 cpu ว่าจะกินมาได้

pthread_t thread_producer[PRODUCERS];
pthread_t thread_consumer[CONSUMERS];

/* สร้าง loop ในการสร้าง thread producer */
for( i=0; i<PRODUCERS; i++){

    pthread_create(&thread_producer[i], NULL, append_buffer, NULL);

}

/* สร้าง loop ในการสร้าง thread consumer */
for(i=0; i<CONSUMERS; i++){

    pthread_create(&thread_consumer[i], NULL, remove_buffer, NULL);

}

/* รอจนกว่า thread จะเสร็จสมบูรณ์ก่อนที่ main จะดำเนินการต่อ */
/* รอ producers และ consumers รัน ถ้าเกิด error อะไรจะทำการหยุด */
/* กระบวนการและ thread ทั้งหมด ก่อนที่จะเสร็จสมบูรณ์ */
for(i=0; i<CONSUMERS; i++){

    pthread_join(thread_consumer[i], NULL);

}

for(i=0; i<PRODUCERS; i++){

    pthread_join(thread_producer[i], NULL);

}

timer2 = clock(); //วัด clock2 cpu ว่าจะกินมาได้
float elapsed = ((float)(timer2 - timer1) / CLOCKS_PER_SEC);

printf("\n");
printf("[+] Producers %d, Consumers %d\n", PRODUCERS, CONSUMERS);
printf("[+] Buffer size %d\n", BUFFER_SIZE);
printf("[+] Requests %d\n\n", request);

```

```

printf("\n");
printf("[+] Producers %d, Consumers %d\n", PRODUCERS, CONSUMERS);
printf("[+] Buffer size %d\n", BUFFER_SIZE);
printf("[+] Requests %d\n", request);
printf("[+] Successfully consumed %d requests (%.1f%%)\n", success, (float)success * 100 / request);
printf("[+] Elapsed Time %.2f s\n", elapsed);
printf("[+] Throughput %.2f successful requests/s\n", (float)(success) / elapsed);

exit(EXIT_SUCCESS);
}

void *add_item() { //ฟังก์ชัน add_item
    buffer[head++] = 1; //buffer[head++] มีค่า=1;
    head = head % BUFFER_SIZE; //head = head หารด้วย buffer_size
}

void *remove_item() { //ฟังก์ชัน remove_item
    buffer[tail++] = 0; //buffer[tail++] มีค่า=0
    tail = tail % BUFFER_SIZE; //tail หารด้วย tail / buffer_size หารด้วย
}

void *append_buffer() { //ฟังก์ชัน append_buffer
    printf("Append thread number %ld\n", pthread_self()); //print ค่า thread ที่เรียกใช้
    while(request < REQUEST) { //ถ้า request < REQUEST while loop while
        if(!pthread_mutex_trylock(&mutex) && request < REQUEST) { //สิทธิ์ในการแก้ไขข้อมูล ถ้าไม่ตรงตามเงื่อนไขจะลูป
            if(buffer[head] == 0) {
                add_item();
                request++;
                printf(" + thread %ld append success\n", pthread_self());
            }
            else { //ถ้าไม่พอ และ buffer overflow
                printf("Buffer overflow\n");
            }
        }
        pthread_mutex_unlock(&mutex); //ปลดล็อก mutex
    }
    pthread_exit(NULL);
}

void *remove_buffer() {
    printf("Remove thread number %ld\n", pthread_self());
    while(success < REQUEST) { //จนกว่า success มีค่า request
        if(!pthread_mutex_trylock(&mutex) && success < REQUEST) { //ถ้า pthread ไม่สามารถล็อก mutex และ success < Request จะถ้า if
            if(buffer[tail] == 1) {
                remove_item();
                success++;
                printf(" - thread %ld remove success\n", pthread_self());
            }
            else {
                printf("Buffer underflow\n");
            }
            pthread_mutex_unlock(&mutex);
        }
    }
    pthread_exit(NULL);
}

```

```

#include <stdio.h>
#include <conio.h>
#include <pthread.h>
#include <unistd.h>
#include <assert.h>
#define N 5;
#define NUM_THREADS 1
pthread_t tid[2];
pthread_mutex_t lock;
int count=0;
int front=0;int rear=0;
char buffer[5];
int prodsleep=0;int consleep=0;
int b;
void *perform_work(void *arguments){
    scanf("%a",&b);
    int index = *((int *)arguments);
    printf("THREAD %d: Started.\n", index);
}

void add_items(void)
{
    char item;
    if (count<5)//ถ้า count < 5 เข้า loop
    {
        printf("\n Enter Input Circular Buffer:");
        scanf(" %c",&item);//รับค่า char เก็บไว้ใน item
        buffer [front]=item;//buffer [front=0] = item
        front = (front+1)%5;//front มีค่า= (front+1) หาก 5 เสร็จแล้ว
        count++;// count เพิ่มขึ้น 1
        if(consleep==1 && count==1)//ถ้า consleep=1 และ count=1
        {
            printf("\n Consumer Wakeup ");

```

```

{
printf("\n Consumer Wakeup ");
}
}
else
{
printf("\n Buffer is Overflow");
prodsleep=1;
}
}

void remove_items(void)
{
char item;
if (count>0)
{
item = buffer[rear]; //item = buffer[rear=0]
buffer[rear]=' ';
printf("\n Remove_items: %c",item);
rear=(rear+1)%5;
count--; //count ลดลง1
if(prodsleep==1 && count==4) //ถ้า prodsleep=1 และ count=4
{
printf("\n No buffer overflow"); //แสดงว่า No buffer overflow
}
}
else
{
printf("\n No buffer underflow");
consleep=1;
}
}

void view(void) //ฟังก์ชัน view
{

```



```

int i;
printf("\n Buffer Data : ");
for(i=0;i<5;i++)//for i=0 to i<5 ; i++
{
printf("| %c ",buffer[i]);
}
}

void Exit()//for Exit
{
printf("THREAD 0: Ended.\n");
}

main()
{
int i,choice,flag=0;
pthread_t threads[NUM_THREADS];
int thread_args[NUM_THREADS];
int g;
int result_code;
if (pthread_mutex_init(&lock, NULL) != 0) //if pthread mutex is not lock then 0
{
printf("\n mutex init has failed\n");
return 1;
}

for (g = 0; g < NUM_THREADS; g++) {
printf("\nIN MAIN: Creating thread %d.\n", g);

thread_args[g] = g;
result_code = pthread_create(&threads[g], NULL, perform_work, &thread_args[g]);//if pthread is not threads[g] then attribute is =Null
assert(!result_code);//if result_code
}

printf("IN MAIN: All threads are created.\n");

```

```

printf("\n 1:Add_Items ");
printf("\n 2:Remove_Items ");
printf("\n 3:View buffer data ");
printf("\n 4:Exit ");

pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_mutex_destroy(&lock);

do
{
printf("\n\n Enter your choice :");
scanf("%d",&choice);//if int is not choice
switch(choice)//if switch is not choice
{
case 1:add_items();//if 1 is not add_item
//wait for each thread to complete

break;
case 2:remove_items();
break;
case 3:view();
break;
case 4: Exit();
printf(".....Exiting.....");
break;
default:printf("\n Invalid Choice!");//if invalid choice is not Invalid Choice!
break;
}
}while(choice!=4);
}

```

เอกสารอ้างอิง

Producer Consumer Problem without using semaphore (ออนไลน์) แหล่งที่มา :

<http://babulax.blogspot.com/2010/04/producer-consumer-problem-without-using.html>

POSIX thread libraries (ออนไลน์) แหล่งที่มา :

<http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

Pthread with circular buffer (ออนไลน์) แหล่งที่มา :

<https://github.com/igorza/pthread-with-circular-buffer>