ใบงานที่ 4

เรื่อง semephores

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นาย กวีวัธน์ กาญจน์สุพัฒนากุล 65543206003-7

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

การทดลองโปรแกรมที่ 2 ให้แก้ไขโค้ดโปรแกรม

1. แสดงผลลัพธ์ หมายเลข TID และค่าของตัวแปร X ของการทำงานในแต่ละครั้ง

ตัวอย่างผลลัพธ์
TID 506783 Value X = 1
TID 506892 Value X = 2
TID 506905 Value X = 3
TID 507181 Value X = 4
TID 507193 Value X = 5
TID 507201 Value X = 6
TID 507205 Value X = 7
TID 507212 Value X = 8
TID 507218 Value X = 9
TID 507225 Value X = 10

Final value of X = 10

หมายเหตุ : หมายเลขของ TID ในแต่ละเครื่อง หรือ แต่ละครั้งที่โปรแกรมทำงาน จะมีการเปลี่ยนแปลงตลอด

ตัวอย่างที่ 1

```c
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int x = 0 ;
sem_t sync ;
void *my_func(void *arg)
{
        sem_wait(&sync) ;
        printf("x = %d\n", x) ;
}
void main ()
{
        pthread_t thread ;
        if (sem_init(&sync, 0, 0) == -1) {
                perror("Could not initialize mylock semaphore") ;
                exit(2);
        }
        if (pthread_create (&thread, NULL , my_func, NULL) < 0 ){
                perror("Error: thread cannot be created") ;
                exit(1);
        }
        x=55;
        sem_post (&sync);
        pthread_join(thread, NULL);
        sem_destroy(&sync);
        exit(0);
}
```

ผลลัพธ์

```
[Kaweewat@localhost lab4]$ ./ex1
x = 55
```

1.ใน main function ทำการเริ่มต้น semaphore (sync) ด้วยค่าเริ่มต้นเป็น 0

2.ถ้าการเริ่มต้น semaphore ไม่สำเร็จ โปรแกรมจะแสดงข้อผิดพลาดและจบการทำงาน

3.สร้าง thread โดยใช้ pthread_create, ถ้าสร้างไม่สำเร็จ จะแสดงข้อผิดพลาดและจบการทำงาน

4.main thread กำหนดค่าตัวแปร x เป็น 55

5.main thread เพิ่มค่า semaphore (sem_post(&sync)) เพื่อให้ thread my_func เริ่มทำงาน

6.main thread รอให้ thread my_func ทำงานเสร็จสิ้นด้วย pthread_join

7.ทำลาย semaphore ด้วย sem_destroy

8.โปรแกรมจบการทำงาน

ตัวอย่างที่ 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int x = 0;
sem_t m;

void *thread(void *arg){
        sem_wait(&m);
        x = x + 1;
        sem_post(&m);
}
void main(){
        pthread_t tid[10];
        int i;
        if (sem_init(&m, 0, 1)== -1){
        perror("could not initializa mylock semaphore");
        exit(2);
        }
        for (i=0; i<10; i++){
        if(pthread_create(&tid[i], NULL, thread, NULL) < 0){
        perror("Error: thread cannot be ceated");
        exit(1);
        }
}
        for (i=0; i<10; i++) pthread_join(tid[i], NULL);
        printf("Final value of x is %d\n", x);
        exit(0);
}
```

ผลลัพธ์

```
[Kaweewat@localhost lab4]$ ./ex3
Final value of x is 10
```

1.โปรแกรมสร้างอาร์เรย์ของ 10 Thread ID (tid) และเริ่มต้นเซมาโฟร์ m ด้วยค่าเริ่มต้นเป็น 1.

2.แต่ละ Thread บวกค่าตัวแปรที่ถูกแชร์ x.

3.ฟังก์ชัน main สร้าง Thread 10 ตัวและรอให้สายงานเสร็จสิ้นด้วย pthread_join.

4.เมื่อ Thread ทั้งหมดเสร็จสิ้นแล้ว โปรแกรมแสดงค่า x ที่สุดท้าย.

5.เซมาโฟร์ถูกทำลายและโปรแกรมจบการทำงาน.

ตัวอย่างที่ 3

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int x = 0;
sem_t m;

void *thread(void *arg) {
    sem_wait(&m); // Acquire lock before accessing x
    x++; // Atomically increment x
    // Print TID and value of x
    printf("Thread ID: %lu, x = %d\n", pthread_self(), x);
    sem_post(&m); // Release lock after accessing x
    return NULL;
}

int main() {
    pthread_t tid[10];
    int i;
    if (sem_init(&m, 0, 1) == -1) {
        perror("Could not initialize semaphore");
        exit(2);
    }
    for (i = 0; i < 10; i++) {
        if (pthread_create(&tid[i], NULL, thread, NULL) < 0) {
            perror("Error creating thread");
            exit(1);
        }
    }
    for (i = 0; i < 10; i++) {
        pthread_join(tid[i], NULL);
    }
    printf("Final value of x is %d\n", x); // x should be 10
    sem_destroy(&m);
    exit(0);
}
```

ผลลัพธ์

```
[Kaweewat@localhost lab4]$ ./ex2
Thread ID: 139802328196864, x = 1
Thread ID: 139802319804160, x = 2
Thread ID: 139802361767680, x = 3
Thread ID: 139802353374976, x = 4
Thread ID: 139802344982272, x = 5
Thread ID: 139802311411456, x = 6
Thread ID: 139802303018752, x = 7
Thread ID: 139802294626048, x = 8
Thread ID: 139802336589568, x = 9
Thread ID: 139802370160384, x = 10
Final value of x is 10
```

1.ฟังก์ชัน thread เพิ่มค่าตัวแปรที่ถูกแชร์ x
2.แต่ละ Thread พิมพ์ Thread ID (TID) และค่าปัจจุบันของ x.
3.ในฟังก์ชัน main, เซมาโฟร์ m ถูกเริ่มต้นด้วยค่าเริ่มต้นเป็น 1.
4. Thread 10 ตัวถูกสร้างโดยใช้ pthread_create.
5.สายหลักรอให้สายงานทั้งหมดเสร็จสิ้นด้วย pthread_join.
6.เมื่อ Thread ทั้งหมดเสร็จสิ้นแล้ว ค่า x ที่สุดท้ายถูกพิมพ์. ควรจะเป็น 10 เนื่องจากแต่ละ Thread IDทำการเพิ่ม x อย่างละครั้ง.
7.เซมาโฟร์ถูกทำลาย และโปรแกรมจบการทำงาน.

สรุป

เซมาโฟร์ (Semaphore) เป็นเทคนิคหนึ่งในการจัดการการแบ่งประสบการณ์ทรัพยากรร่วมกันระหว่างสายงาน (thread) ในโปรแกรมหลายสาย (multithreading) หรือกระบวนการหลายประกาศ (multiprocessing) เพื่อป้องกันปัญหาแขนง (race condition) และทำให้การใช้ทรัพยากรนั้นๆ เป็นไปอย่างปลอดภัย สามารถล็อกการทำงานและปลดล็อกการทำงาน เพื่อประหยัดทรัพยากรของเครื่อง

เอกสารอ้างอิง

อาจารย์ปิยพล ยืนยงถาวร: เอกสารประกอบการสอน 4 Process Synchronization การประสานเวลาของโพรเซส