ใบงานที่ 4

เรื่อง Semephores

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นางสาวรัชนีกร เชื้อดี 65543206077-1

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

ใบงานที่ 4

Semephores

ลำดับขั้นการทดลอง

1. ให้เขียนโปรแกรมที่กาหนดให้พร้อมอธิบายคาสั่งของโปรแกรมและอธิบายหลักการทำงาน
2. บันทึก และ สรุปผลการทดลอง

บันทึกผลการทดลอง

1. **Example (Signaling)**

```c
#include <stdio.h>
#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");
        }
        x = 55;
        sem_post(&sync);
        pthread_join(thread, NULL);
        sem_destroy(&sync);
```

บรรทัดที่ 1-4: เป็นการประกาศไลบรารีที่จำเป็นสำหรับโค้ดนี้ ไลบรารี stdio.h ให้การเข้าถึงฟังก์ชันอินพุต/เอาต์พุต เช่น printf ไลบรารี stdlib.h ให้การเข้าถึงฟังก์ชันทั่วไป เช่น exit ไลบรารี pthread.h ให้การเข้าถึงฟังก์ชันสำหรับสร้างและจัดการเธรด และไลบรารี semaphore.h ให้การเข้าถึงฟังก์ชันสำหรับใช้สัญญาณเพื่อประสานงานกัน

บรรทัดที่ 5: บรรทัดนี้ประกาศตัวแปรแบบจำนวนเต็มแบบ Global x ด้วยค่าเริ่มต้นเป็น 0

บรรทัดที่ 6: บรรทัดนี้ประกาศตัวแปรแบบสัญญาณ sync แต่ยังไม่กำหนดค่า

บรรทัดที่ 7: ประกาศฟังก์ชันชื่อ my_func ที่รับอาร์กิวเมนต์แบบ void pointer (arg) และส่งคืนแบบ void pointer

บรรทัดที่ 8: รอสัญญาณ sync เธรดจะถูกบล็อกจนกว่าค่าของสัญญาณจะมากกว่า 0

บรรทัดที่ 9: พิมพ์ค่าของตัวแปร x ไปยังคอนโซล

บรรทัดที่ 10: เพิ่มค่าของสัญญาณ sync ขึ้น 1

บรรทัดที่ 11: ประกาศตัวแปรชื่อ thread เพื่อเก็บ ID ของเธรดที่สร้างขึ้น

บรรทัดที่ 12: กำหนดค่าสัญญาณ sync ด้วยค่าเริ่มต้นเป็น 0 และตรวจสอบข้อผิดพลาด หากการกำหนดค่าล้มเหลว ฟังก์ชัน perror() จะพิมพ์ข้อความข้อผิดพลาดไปยังคอนโซล จากนั้นฟังก์ชัน exit() จะสิ้นสุดโปรแกรมด้วยรหัสข้อผิดพลาด 2

บรรทัดที่ 13: สร้างเธรดใหม่ที่จะเรียกใช้ฟังก์ชัน my_func หากการสร้างเธรดล้มเหลว ฟังก์ชัน perror() จะพิมพ์ข้อความข้อผิดพลาดไปยังคอนโซล

บรรทัดที่ 15: กำหนดค่า 55 ให้กับตัวแปร x

บรรทัดที่ 16: เพิ่มค่าของสัญญาณ sync ขึ้น 1 ซึ่งอนุญาตให้เธรดที่รอ sem_wait ดำเนินการต่อได้

บรรทัดที่ 18: รอให้เธรดที่ระบุด้วย thread สิ้นสุดการทำงาน

บรรทัดที่ 19: ทำลายสัญญาณ sync เพื่อปลดปล่อยทรัพยากรที่ใช้

ผลลัพธ์

```
X = 55
```

ผลลัพธ์นี้เกิดขึ้นเนื่องจากเธรดหลักกำหนดค่าตัวแปร x เป็น 55 จากนั้นส่งสัญญาณ sync ให้กับเธรดที่สร้างขึ้นใหม่ เธรดที่สร้างขึ้นใหม่จะรอสัญญาณ sync ก่อนที่จะเข้าถึงตัวแปร x เมื่อเธรดหลักส่งสัญญาณ sync ให้กับเธรดที่สร้างขึ้นใหม่ เธรดที่สร้างขึ้นใหม่จึงสามารถเข้าถึงตัวแปร x และพิมพ์ค่าของตัวแปร x ไปยังคอนโซล

2. Example (Critical Section)

```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 initialize mylock semaphore");
        exit(2);
        }
        for (i=0; i<10; i++)
{
        if (pthread_create(&tid[i], NULL, thread, NULL) < 0) {
                perror("Error: thread cannot be created");
                exit(1);
            }
}
            for (i=0; i<10; i++) pthread_join(tid[i], NULL);
            printf("Final value of x is %d\n", x);
            exit(0);
}
```

บรรทัดที่ 1-4:

- #include <stdio.h>: ประกาศการใช้งานไลบรารี stdio.h ซึ่งมีฟังก์ชันสำหรับการพิมพ์ข้อมูล
- #include <stdlib.h>: ประกาศการใช้งานไลบรารี stdlib.h ซึ่งมีฟังก์ชันสำหรับการจัดการหน่วยความจำ
- #include <pthread.h>: ประกาศการใช้งานไลบรารี pthread.h ซึ่งมีฟังก์ชันสำหรับการสร้างและจัดการเธรด
- #include <semaphore.h>: ประกาศการใช้งานไลบรารี semaphore.h ซึ่งมีฟังก์ชันสำหรับการใช้สัญญาณเพื่อประสานงานกัน

บรรทัดที่ 5:

- int x = 0;: ประกาศตัวแปรแบบ integer ชื่อ x และกำหนดค่าเริ่มต้นเป็น 0

บรรทัดที่ 6:

- sem_t m;: ประกาศตัวแปรแบบ semaphore ชื่อ m

บรรทัดที่ 7-13:

- void *thread(void *arg): นิยามฟังก์ชันชื่อ thread ที่รับอาร์กิวเมนต์เป็น void pointer (arg) และส่งคืนค่าแบบ void pointer
- sem_wait(&m);: รอสัญญาณ semaphore m เธรดจะถูกบล็อกจนกว่าค่าของ semaphore มากกว่า 0
- x = x + 1;: เพิ่มค่าของตัวแปร x ทีละ 1
- sem_post(&m);: ส่งสัญญาณ semaphore m แจ้งให้เธรดอื่นทราบว่าสามารถเข้าถึงตัวแปร x ได้แล้ว

บรรทัดที่ 14-20:

- pthread_t tid[10];: ประกาศตัวแปรแบบ array ชื่อ tid ประกอบด้วย 10 thread ID
- int i;: ประกาศตัวแปร integer ชื่อ i
- if (sem_init(&m, 0, 1) == -1) { ... }: กำหนดค่า semaphore m ด้วยค่าเริ่มต้นเป็น 1 และตรวจสอบข้อผิดพลาด
- for (i=0; i<10; i++): ทำ loop 10 รอบ
    - if (pthread_create(&tid[i], NULL, thread, NULL) < 0) { ... }: สร้างเธรดใหม่ 10 เธรด เรียกใช้ฟังก์ชัน thread และบันทึก thread ID ไว้ใน array tid
- for (i=0; i<10; i++) pthread_join(tid[i], NULL);: รอให้เธรดทั้งหมดทำงานเสร็จสิ้น

- printf("Final value of x is %d\n", x);: พิมพ์ค่าสุดท้ายของ x

บรรทัดที่ 21-22:

- exit(0);: สิ้นสุดการทำงานของโปรแกรม

ผลลัพธ์

```
Final value of x is 10
```

ผลลัพธ์ของโค้ดนี้จะขึ้นอยู่กับระบบที่รัน ซึ่งขึ้นอยู่กับการทำงานของ scheduler และความเร็วของ CPU แต่โดยทั่วไป ค่าสุดท้ายของ x ควรจะอยู่ใกล้เคียงกับ 10

3. เพิ่มเติมจากจากโปรแกรมที่ 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++;
        printf("Thread ID: %lu, x = %d\n", pthread_self(), x);
        sem_post(&m);
        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);
         sem_destroy(&m);
         exit(0);

}
```

บรรทัดที่ 1-4:

- #include <stdio.h>: ประกาศการใช้งานไลบรารี stdio.h ซึ่งมีฟังก์ชันสำหรับการพิมพ์ข้อมูล
- #include <stdlib.h>: ประกาศการใช้งานไลบรารี stdlib.h ซึ่งมีฟังก์ชันสำหรับการจัดการหน่วยความจำ
- #include <pthread.h>: ประกาศการใช้งานไลบรารี pthread.h ซึ่งมีฟังก์ชันสำหรับการสร้างและจัดการเธรด

- #include <semaphore.h>: ประกาศการใช้งานไลบรารี semaphore.h ซึ่งมีฟังก์ชันสำหรับใช้สัญญาณในการประสานงาน

บรรทัดที่ 5:

- int x = 0;: ประกาศตัวแปรแบบ integer ชื่อ x และกำหนดค่าเริ่มต้นเป็น 0

บรรทัดที่ 6:

- sem_t m;: ประกาศตัวแปรแบบ semaphore ชื่อ m

บรรทัดที่ 7-13:

- void *thread(void *arg): นิยามฟังก์ชันชื่อ thread ที่รับอาร์กิวเมนต์เป็น void pointer (arg) และส่งคืนค่าแบบ void pointer
- sem_wait(&m);: รอสัญญาณ semaphore m เธรดจะถูกบล็อกจนกว่าค่าของ semaphore มากกว่า 0
- x++;: เพิ่มค่าของตัวแปร x ทีละ 1
- printf("Thread ID: %1u, x = %d\n", pthread_self(), x);: พิมพ์ ID thread และค่า x ไปยังคอนโซล
- sem_post(&m);: ส่งสัญญาณ semaphore m แจ้งให้เธรดอื่นทราบว่า thread นี้เสร็จสิ้นการเข้าถึงตัวแปร x แล้ว
- return NULL;: ส่งคืนค่า NULL

บรรทัดที่ 14-21:

- pthread_t tid[10];: ประกาศตัวแปรแบบ array ชื่อ tid ประกอบด้วย 10 thread ID
- int i;: ประกาศตัวแปร integer ชื่อ i
- if (sem_init(&m, 0, 1) == -1) { ... }: กำหนดค่า semaphore m ด้วยค่าเริ่มต้นเป็น 1 และตรวจสอบข้อผิดพลาด
- for (i = 0; i < 10; i++): ทำ loop 10 รอบ
  - if (pthread_create(&tid[i], NULL, thread, NULL) < 0) { ... }: สร้างเธรดใหม่ 10 เธรด เรียกใช้ฟังก์ชัน thread และบันทึก thread ID ไว้ใน array tid
- for (i = 0; i < 10; i++): ทำ loop 10 รอบ
  - pthread_join(tid[i], NULL);: รอให้เธรดทั้งหมดทำงานเสร็จสิ้น
- printf("Final value of x is %d\n", x);: พิมพ์ค่าสุดท้ายของ x

บรรทัดที่ 22-23:

- sem_destroy(&m);: ทำลาย semaphore m
- exit(0);: สิ้นสุดการทำงานของโปรแกรม

ผลลัพธ์

```
Thread ID: 2146596608, x = 1
Thread ID: 2138203904, x = 2
Thread ID: 2129811200, x = 3
Thread ID: 2121418496, x = 4
Thread ID: 2113025792, x = 5
Thread ID: 2104633088, x = 6
Thread ID: 2096240384, x = 7
Thread ID: 2087847680, x = 8
Thread ID: 2079454976, x = 9
Thread ID: 2071062272, x = 10
Final value of x is 10
```

โปรแกรมนี้จะพิมพ์ ID thread และค่า x ในแต่ละเธรด ค่าสุดท้ายของ x ควรเป็น 10 แต่ขึ้นอยู่กับ timing และ scheduling ค่าสุดท้ายอาจแตกต่างกันเล็กน้อย

สรุปผลการทดลอง

Semaphores เป็นเครื่องมือที่มีประโยชน์ในการประสานงานระหว่างเธรดและสามารถช่วยป้องกันการแข่งขันระหว่างเธรดในการเข้าถึงตัวแปรแบบแชร์ การใช้ semaphores อย่างถูกต้องจะช่วยให้มั่นใจว่าข้อมูลมีความถูกต้อง แต่ก็มีข้อควรระวังคือการใช้ semaphores ที่ไม่ถูกต้องอาจทำให้เกิดปัญหา เช่น ข้อมูลสูญหายหรือเสียหาย

สื่อ / เอกสารอ้างอิง

อาจารย์ปิยพล ยืนยงสถาวร: เอกสารประกอบการสอน 4 Process Synchronization การประสานเวลาของโปรเซส