

Activity 7: Process Synchronization

วัตถุประสงค์

1. เพื่อให้ห็นสิดเข้าใจหลักการของ process synchronization
2. เพื่อให้ห็นสิดสามารถเขียนโปรแกรมใช้งาน semaphore ได้

เตรียมตัว

1. ศึกษาหลักการ semaphore ในบทที่ 6 Process Synchronization
2. ศึกษา Linux POSIX named semaphore

ความรู้พื้นฐาน

Process Synchronization เป็นองค์ประกอบที่สำคัญในการทำงานร่วมกันของ Process หรือ Thread ซึ่งเครื่องมือใน Linux จะรองรับทั้งการทำ Semaphore และ Shared Memory

Semaphore เป็นตัวแปรประเภท counter ที่แสดงถึงสถานะของทรัพยากร โดยที่ counter แบบ semaphore จะมีลักษณะพิเศษคือ Operating System จะทำการดูแลไม่ให้เกิด race condition กล่าวคือ ผู้ใช้งานสามารถมั่นใจได้ว่า ณ เวลาใดเวลาหนึ่ง ค่าใน counter จะถูกแก้ไขได้โดยเพียง Process หรือ Thread เดียวเท่านั้น

Semaphore มี 2 ประเภทคือ Named Semaphore สำหรับรองรับการทำงาน ระหว่างหลายๆ Process (สร้างโดยคำสั่ง **sem_open**) และ Unnamed Semaphore สำหรับรองรับการทำงานของหลาย Thread ภายใน Process เดียวกัน (สร้างโดยคำสั่ง **sem_init**)

การทำงานของ Semaphore จะขึ้นอยู่กับค่าของ counter โดย Process หรือ Thread สามารถทำการลดค่า (ทำการ Lock) ด้วยคำสั่ง **sem_wait** หรือเพิ่มค่า (ทำการ Unlock) ด้วยคำสั่ง **sem_post** ของ Semaphore ได้ โดยที่ถ้า Process หรือ Thread พยายามจะลดค่าของ Semaphore ในขณะที่มีค่าเป็นศูนย์ Process หรือ Thread นั้นจะถูก block และจะต้องรอจนกว่าค่า Semaphore จะถูกเพิ่มจนมีค่ามากกว่าศูนย์ ซึ่งจะเกิดขึ้นได้ก็ต่อเมื่อมี Process หรือ Thread อื่นมาทำการเพิ่มค่า หรือ Unlock Semaphore นี้สำหรับ Binary Semaphore จะเป็น Semaphore ที่มีค่าอยู่ระหว่าง 0 กับ 1 ซึ่งจะมีชื่อเรียกเป็นพิเศษว่า Mutex โดยจะถูกใช้สำหรับการ Lock/Unlock Critical Section

รายละเอียดของ Semaphore ศึกษาเพิ่มเติมได้จาก

https://linux.die.net/man/7/sem_overview

กิจกรรม Callcenter Simulation

ในกิจกรรมนี้จะให้ผลิตทำการปรับปรุง source code ของโปรแกรม Callcenter Simulator โดยให้ผลิตดาวน์โหลดไฟล์ activity-7.zip จาก Course Material “Activity 7: Synchronization (activity-7.zip)” ใน CourseVile ภายหลังจากการทำ unzip จะพบไฟล์ 4 ไฟล์ใน semaphore คือ

- makefile – สำหรับการใช้คำสั่ง make ในการ compile
- callcenter.c – เป็นโปรแกรมในส่วนของ server ที่จำลองระบบ call center ที่มีพนักงานให้บริการจำนวน n คน หรืออาจเรียกว่ามี n คู่สาย (n จะเป็นค่าที่ส่งผ่านทาง command line ไปยังตัวโปรแกรมเช่น ถ้า run ด้วยคำสั่ง callcenter 3 หมายถึงให้ทำการจำลองระบบ call center จำนวน 3 คู่สาย) โดยทำการสร้าง Named Semaphore “callcenter” พร้อมทั้งระบุค่าตั้งต้นของ Semaphore เป็น n
- customer.c – เป็นโปรแกรมที่จำลองลูกค้าหรือผู้ใช้บริการที่พยายามจะโทรเข้า callcenter โดยผู้โทรจะทำการติดต่อไปยัง callcenter เพื่อคุยกับพนักงาน (โดยการใช้ Named Semaphore ชื่อ “callcenter”) เมื่อมีพนักงานว่างมารับสายแล้ว ก็จะคุยเป็นระยะเวลาสุ่มระหว่าง 1-5 วินาที (สมมติว่าแทนเวลาจริง 1-5 นาที) หลังจากนั้นจะวางสาย โปรแกรม caller จะรอเป็นระยะเวลาสุ่มระหว่าง 1-3 วินาที ก่อนที่จะจำลองลูกค้าคนต่อไปที่จะโทรเข้า callcenter
- callcenter_rm.c – เป็นโปรแกรมที่ทำการยกเลิก Semaphore ที่ใช้ใน callcenter

ใน source code ของ callcenter.c และ customer.c ที่ได้รับจะมีรายละเอียดไม่ครบถ้วน กล่าวคือในส่วนคำสั่งที่เกี่ยวข้องกับ Semaphore ได้ถูกแทนที่ด้วย Comment ตัวอย่างเช่น ในไฟล์ customer.c บรรทัดที่ 16-18 จะมีข้อความ

```
//  
// OS -- OPEN NAMED SEMAPHORE HERE  
//
```

เป็นการระบุว่า ให้นำคำสั่งเกี่ยวกับการเปิด named semaphore มาแทนที่ comment นี้

สิ่งที่ต้องทำ

- ปรับปรุง source code โดยการเพิ่มคำสั่งเกี่ยวกับ Semaphore ที่เหมาะสม
- ใช้คำสั่ง make เพื่อคอมไพล์โปรแกรม ซึ่งจะได้ผลลัพธ์เป็นโปรแกรมสองโปรแกรมชื่อ callcenter และ customer
- ทำการทดสอบด้วยการรันโปรแกรม callcenter โดยมี argument เป็นจำนวนพนักงาน และรันโปรแกรม customer หลาย ๆ ครั้ง ให้มีจำนวนโปรเซสมากกว่าจำนวนคู่สาย (แต่ละ process อยู่คนละหน้าต่าง terminal กัน)
- ตัวอย่างดังต่อไปนี้เป็นการให้ callcenter สร้างคู่สายจำนวน 1 คู่สาย และมี customer จำนวน 2 process ถ้าทำได้ถูกต้อง ควรจะได้ผลลัพธ์ในลักษณะดังนี้

```
$ ./callcenter 2  
Starting a call center with 2 agents.  
There are 2 agents available now.  
There are 2 agents available now.  
There are 1 agents available now.  
There are 1 agents available now.  
There are 0 agents available now.
```

```
There are 1 agents available now.  
There are 0 agents available now.  
There are 1 agents available now.  
There are 2 agents available now.  
...
```

```
$ ./customer  
Starting customer  
Wait for 2 minutes  
Next customer calls the call center, press 10 buttons, and listens to silly music.  
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.  
Customer ends the call.  
Wait for 1 minutes  
Next customer calls the call center, press 10 buttons, and listens to silly music.  
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 1 minutes.  
Customer ends the call.  
Wait for 3 minutes  
Next customer calls the call center, press 10 buttons, and listens to silly music.  
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 1 minutes.  
Customer ends the call.  
Wait for 2 minutes  
...
```

ให้ capture หน้าจอผลลัพธ์เก็บไว้

สิ่งที่ต้องส่งใน courseville

- 1) source code ที่ได้แก้แล้ว
- 2) ภาพหน้าจอผลลัพธ์

จะใส่สิ่งที่ต้องส่งโดยเพิ่มลงในไฟล์นี้ หรือส่งเป็นไฟล์แยกต่างหากก็ได้

Activity 7

ธนัส วงศ์สมุทร 6432067021

กองภพ จริยาสดภาพร 6430014321

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>

int main(int argc, char **argv)
{
    int num_agents = 2;
    if (argc > 1)
        num_agents = atoi(argv[1]);
    printf("Starting a call center with %d agents.\n", num_agents);

    //
    // OS -- CRAETE NAMED SEMAPHORE HERE
    //
    sem_t *sem = sem_open("callcenter", O_CREAT, 0644, num_agents);
    if (sem == SEM_FAILED)
    {
        perror("sem_open failed");
        exit(EXIT_FAILURE);
    }

    int semval;
    while (1)
    {
        //
        // OS -- PLACE CURRENT VALUE OF SEMAPHORE IN 'semval' HERE
        //
        if (sem_getvalue(sem, &semval) == -1)
        {
            perror("sem_getvalue failed");
            exit(EXIT_FAILURE);
        }

        printf("There are %d agents available now.\n", semval);
        sleep(3);
    }
}
```

callcenter.c

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <semaphore.h>

int rand_int(int n)
{
    // Generate random integer number between [1, n]
    int r = rand();
    return (r % n) + 1;
}

int main(int argc, char **argv)
{
    printf("Starting customer\n");

    //
    // OS -- OPEN NAMED SEMAPHORE HERE
    //
    sem_t *sem = sem_open("/callcenter", 0); // Assuming the semaphore is already created by callcenter
    if (sem == SEM_FAILED)
    {
        perror("sem_open failed");
        exit(EXIT_FAILURE);
    }

    while (1)
    {
        // Customer will wait between 1-3 seconds before placing the next phone call
        int wait_time = rand_int(3);
        printf("Wait for %d minutes\n", wait_time);
        sleep(wait_time);
        printf("Next customer calls the call center, press 10 buttons, and listens to silly music.\n");
        time_t t0 = time(NULL);
        // Wait for an agent

        //
        // OS -- LOCK SEMAPHORE HERE
        //
        if (sem_wait(sem) == -1)
        {
            perror("sem_wait failed");
            exit(EXIT_FAILURE);
        }

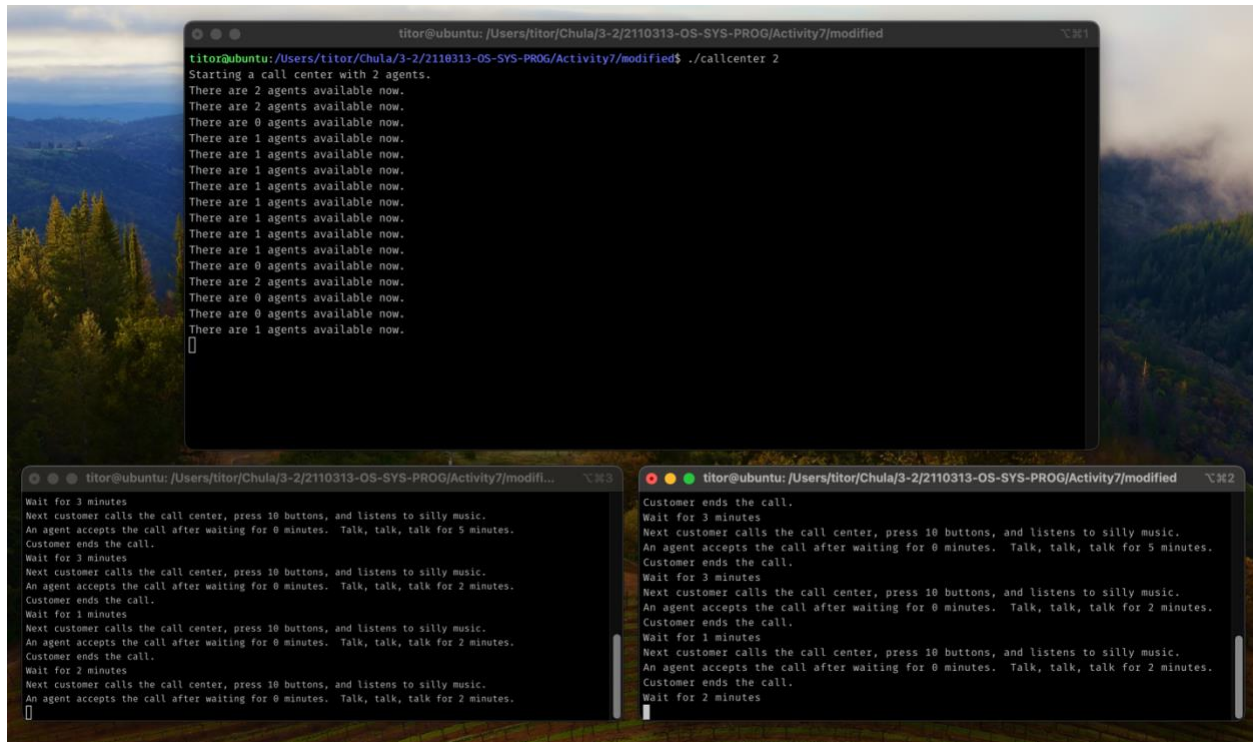
        time_t t = time(NULL) - t0;
        // An agent accepts the call, using it for 1-5 seconds.
        int call_time = rand_int(5);
        printf("An agent accepts the call after waiting for %ld minutes. Talk, talk, talk for %d\n", t, call_time);
        sleep(call_time);
        // Customer hangs up the phone

        //
        // OS -- UNLOCK SEMAPHORE HERE
        //
        if (sem_post(sem) == -1)
        {
            perror("sem_post failed");
            exit(EXIT_FAILURE);
        }

        printf("Customer ends the call.\n");
    }
}

```

Screenshot



```
titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity7/modified
titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity7/modified$ ./callcenter 2
Starting a call center with 2 agents.
There are 2 agents available now.
There are 2 agents available now.
There are 0 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 1 agents available now.
There are 0 agents available now.
There are 2 agents available now.
There are 0 agents available now.
There are 0 agents available now.
There are 1 agents available now.
[]

titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity7/modifi...
Wait for 3 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 5 minutes.
Customer ends the call.
Wait for 3 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.
Customer ends the call.
Wait for 1 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.
Customer ends the call.
Wait for 2 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.
[]

titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity7/modified
Customer ends the call.
Wait for 3 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 5 minutes.
Customer ends the call.
Wait for 3 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.
Customer ends the call.
Wait for 1 minutes
Next customer calls the call center, press 10 buttons, and listens to silly music.
An agent accepts the call after waiting for 0 minutes. Talk, talk, talk for 2 minutes.
Customer ends the call.
Wait for 2 minutes
```