ใบงานที่ 5

เรื่อง Threads Creation and Execution

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นายอธิศ สุนทโรดม

รหัส 65543206086-2

ใบงานนี้เป็นส่วนหนึ่งของวิชาระบบปฏิบัติการ (ENGCE125)

หลักสูตรวิศกรรมมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา เชียงใหม่

ประจำภาคเรียนที่ 2 ปีการศึกษา 2566

จุดประสงค์การสอน

- เพื่อให้มีความรู้ความเข้าใจการสร้าง Creating Threads
- เพื่อให้มีความรู้ความเข้าใจ Terminating Thread Execution
- เพื่อให้มีความรู้ความเข้าใจ Passing Arguments To Threads
- เพื่อให้มีความรู้ความเข้าใจ Thread Identifiers
- เพื่อให้มีความรู้ความเข้าใจ Joining Threads
- เพื่อให้มีความรู้ความเข้าใจ Detaching / Undetaching Threads

อุปกรณ์ที่ใช้งาน

- คอมพิวเตอร์ 1 เครื่อง พร้อมติดตั้ง Virtualbox
- CentOS เวอร์ชัน 7 หรือสูงกว่าที่ติดตั้งบน Virtualbox

ทฤษฎี

การตั้งชื่อ: ข้อกำหนดทั้งหมดเริ่มต้นด้วย pthread_ฟังก์ชั่นการจัดการ

pthread_	Threads themselves and miscellaneous subroutines
pthread_t	Thread objects
pthread_attr	Thread attributes objects
pthread_mutex	Mutexes
pthread_mutexattr	Mutex attributes objects.
pthread_cond	Condition variables
pthread_condattr	Condition attributes objects
pthread_key	Thread-specific data keys

ฟังก์ชั่นการจัดการ

ฟังก์ชัน pthread_create ถูกใช้เพื่อสร้างเธรดใหม่ และฟังก์ชัน pthread_exit ใช้เพื่อยุติเธรดนั้นเอง การดำเนินการนี้จะรอการสิ้นสุดของเธรดอื่นโดยใช้ฟังก์ชัน pthread_join

Function:	int pthread_create (pthread_t * threadhandle, /* Thread handle returned by reference */ pthread_attr_t *attribute, /* Special Attribute for starting thread, may be NULL */ void *(*start_routine)(void *), /* Main Function which thread executes */ void *arg /* An extra argument passed as a pointer */);
Info:	Request the PThread library for creation of a new thread. The return value is 0 on success. The return value is negative on failure. The pthread_t is an abstract datatype that is used as a handle to reference the thread.

Function:	void pthread_exit (void *retval /* return value passed as a pointer */);
Info:	This Function is used by a thread to terminate. The return value is passed as a pointer. This pointer value can be anything so long as it does not exceed the size of (void *). Be careful, this is system dependent. You may wish to return an address of a structure, if the returned data is very large.

Function:	int pthread_join (pthread_t threadhandle, /* Pass threadhandle */ void **returnvalue /* Return value is returned by ref. */);
Info:	Return 0 on success, and negative on failure. The returned value is a pointer returned by reference. If you do not care about the return value, you can pass
	NULL for the second argument.

เริ่มทำงาน

รวมไลบรารี pthread.h:

#รวม <pthread.h>

ประกาศตัวแปรประเภท pthread_t

pthread_t the_thread

เมื่อคุณคอมไพล์ไฟล์ lpthread ไปยังตัวเชื่อมโยง:

gcc thread.c -o เธรด -lpthread

ขั้นตอนก่อนหน้านี้จะถูกสร้างขึ้นภายในกระบวนการ เมื่อสร้างขึ้นแล้วมันเป็นความรู้สึก คงเหมือนกับนักบัญชีของคุณนะสร้างเธรดอื่น โปรดทราบว่า "เธรดเริ่มต้น" มีอยู่ตามค่าเริ่มต้นและเป็นเธรดการยุติหลักเธรดการดำเนินการ

int pthread_attr_destroy (pthread_attr_t *attr) pthread_attr_destroy

ทำลายอ็อบเจ็กต์แอ็ตทริบิวต์ที่ชี้ไปที่ attr และเผยแพร่ทรัพยากรทั้งหมดที่เกี่ยวข้องกับ attr ยังคงไม่ได้กำหนดไว้ และคุณไม่จำเป็นต้องใช้อีกครั้งเมื่อเรียกใช้ฟังก์ชัน pthreads จนกระทั่ง เริ่มต้นใหม่แล้ว

int pthread_attr_setattr (pthread_attr_t *obj, ค่า int)

ตั้งค่า attr ในค่าในแอตทริบิวต์ ตัวชี้ไปที่ obj ดูรายการไฟที่มีอยู่ด้านล่างเป็นไปได้และค่าที่สามารถนำมาใช้เกี่ยวกับความสำเร็จของฟังก์ชัน...เหล่านี้คือ 0

int pthread_attr_getattr (const pthread_attr_t *obj, int *ค่า)

คงการตั้งค่าปัจจุบันของ attr ใน obj ไว้เป็นสมุนไพร ค่าฟังก์ชันเหล่านี้จะกลับเป็น 0

การทดลองที่ 5

วิธีการทดลอง

1. เขียนโปรแกรมตามโค้ดตัวอย่างบนระบบปฏิบัติการ CentOS และแสดงผลทดลอง
2. อธิบายการทำงานของโปรแกรม
3. สรุปผลการทดลอง
4. ส่งงานใน Microsoft Teams เป็นไฟล์ PDF

<u>Example: Pthread Creation and Termination</u>

ข้อ 1. อธิบายคำสั่งที่กำหนดให้ต่อไปนี้ พร้อมกับทำการสร้าง Thread ขึ้นมาจากโปรแกรมคำสั่งที่กำหนดให้

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

void *print_message_function(void *ptr);
/* struct to hold data to be passed to a thread this shows how multiple data items can be
passed to a thread */
int main()
{
    pthread_t thread1, thread2;
    /* thread variables */
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;

    /* Create independent threads each of which will execute function */
    iret1 = pthread_create(&thread1, NULL, print_message_function, (void *)message1);
    iret2 = pthread_create(&thread2, NULL, print_message_function, (void *)message2);

    /* Wait till threads are complete before main continues. Unless we */
    /* wait we run the risk of executing an exit which will terminate */
    /* the process and all threads before the threads have completed. */

    pthread_join(thread1, NULL); /* Start waiting for thread1. */
    pthread_join(thread2, NULL); /* Start waiting for thread2. */

    printf("Thread 1 returns: %d\n", iret1);
    printf("Thread 2 returns: %d\n", iret2);
    exit(0);
}
void *print_message_function(void *ptr)
{
    char *message;
    message = (char *)ptr;
    printf("%s\n", message);
}
```

```
[root@localhost Lab5]# ./ex1
Thread 2
Thread 1
Thread 1 returns: 0
Thread 2 returns: 0
```

โปรแกรมจัดการเธรด โปรแกรมนี้สร้างเธรดขึ้นมาสองเธรด โดยแต่ละเธรดจะทำงานซ้ำๆ กัน โดยการพิมพ์ข้อความ "Thread 1" และ "Thread 2" ตามลำดับโดยเริ่มต้น #include <stdio.h>, #include<stdlib.h>, #include <pthread.h>คำสั่งเหล่านี้ใช้ในการนำเข้าไลบรารีที่จำเป็นสำหรับการทำงานของโปรแกรม void *print_message_function(void *ptr) ฟังก์ชันนี้ทำหน้าที่เป็นฟังก์ชันทำงาน (runnable function) ของเธรด ฟังก์ชันนี้จะพิมพ์ข้อความที่ส่งเข้ามาเป็นอาร์กิวเมนต์ pthread_t thread1, thread2; ตัวแปรเหล่านี้ใช้เพื่อเก็บข้อมูลของเธรด

ข้อ 2. นำ Example: pthread1.c มาทำการ เพิ่ม ฟังก์ชั่นการทำงาน เข้าไป ใน pthread1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function(void *ptr);
/* struct to hold data to be passed to a thread this shows how multiple data items can be
passed to a thread */
int count = 0;

int main()
{
    pthread_t thread1, thread2, thread3;
    /* thread variables */
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    char *message3 = "Thread 3";
    int iret1, iret2, iret3;
    /* Create independent threads each of which will execute function */
    iret1 = pthread_create(&thread1, NULL, print_message_function, (void *)message1);
    iret2 = pthread_create(&thread2, NULL, print_message_function, (void *)message2);
    iret3 = pthread_create(&thread3, NULL, print_message_function, (void *)message3);
    /* Wait till threads are complete before main continues. Unless we */
    /* wait we run the risk of executing an exit which will terminate */
    /* the process and all threads before the threads have completed. */

    pthread_join(thread1, NULL); /* Start waiting for thread1. */
    pthread_join(thread2, NULL); /* Start waiting for thread2. */
    pthread_join( thread3, NULL);

    printf("Thread 1 returns: %d\n", iret1);
    printf("Thread 2 returns: %d\n", iret2);
    exit(0);
}
void *print_message_function(void *ptr)
{

        char *message;
        message = (char *)ptr;
        printf("%s pid = %d tid = %u\n", message, getpid(), (unsigned int)pthread_self());

        int i = 0;
        for (i = 0; i < 10; i++)
        {
                sleep(1);
                printf("%u -> %d count = %d", (long)pthread_self(), i, count);
                count++;
                printf("\n");
        }
}
```

```
[root@localhost Lab5]# ./ex2
Thread 2 pid = 8478 tid = 38364928
Thread 3 pid = 8478 tid = 29972224
Thread 1 pid = 8478 tid = 46757632
29972224 -> 0 count = 0
38364928 -> 0 count = 1
46757632 -> 0 count = 2
38364928 -> 1 count = 3
29972224 -> 1 count = 4
46757632 -> 1 count = 5
29972224 -> 2 count = 6
38364928 -> 2 count = 7
46757632 -> 2 count = 8
38364928 -> 3 count = 9
29972224 -> 3 count = 10
46757632 -> 3 count = 11
29972224 -> 4 count = 12
38364928 -> 4 count = 13
46757632 -> 4 count = 14
38364928 -> 5 count = 15
29972224 -> 5 count = 16
46757632 -> 5 count = 17
38364928 -> 6 count = 18
29972224 -> 6 count = 19
46757632 -> 6 count = 20
29972224 -> 7 count = 21
46757632 -> 7 count = 22
38364928 -> 7 count = 23
29972224 -> 8 count = 24
46757632 -> 8 count = 25
38364928 -> 8 count = 26
29972224 -> 9 count = 27
46757632 -> 9 count = 28
38364928 -> 9 count = 29
Thread 1 returns: 0
Thread 2 returns: 0
```

ได้ทำการเพิ่มฟังก์ชัน print_message_funtion() เพื่อรับอาร์กิวเมนต์เป็นตัวแปร message ซึ่งเป็นตัวชี้ไปยังข้อความที่จะพิมพ์ ฟังก์ชันนี้จะเริ่มทำงานโดยพิมพ์ข้อความที่ส่งเข้ามาเป็นอาร์กิวเมนต์ จากนั้นจะวนซ้ำไปเรื่อยๆ โดยพิมพ์ข้อความ "Thread 1" และ "Thread 2" สลับกัน

ขั้นตอนการทำงานของฟังก์ชันprint_message_functio() มีดังนี้ ดึงข้อความที่จะพิมพ์ออกมาจากตัวแปร message พิมพ์ข้อความที่ดึงออกมาวนซ้ำไปเรื่อยๆ โดยพิมพ์ข้อความ "Thread 1" และ "Thread 2" สลับกันเมื่อฟังก์ชันนี้ทำงานเสร็จสิ้น จะคืนค่า NULL ให้กับฟังก์ชัน pthread_create() ในภาพประกอบ ฟังก์ชัน print_message_function() ถูกเรียกใช้จากฟังก์ชัน main() โดยส่งข้อความ "Thread 1" และ "Thread 2" เป็นอาร์กิวเมนต์ ส่งผลให้เธรดทั้งสองจะทำงานซ้ำๆ กัน โดยพิมพ์ข้อความ "Thread 1" และ "Thread 2" สลับกัน

สรุปผลการทดลอง

เพื่อจัดการเธรด ผลการทดลองพบว่าสามารถสร้างเธรดขึ้นมาได้สำเร็จ โดยเธรดแต่ละเธรดจะทำงานซ้ำๆ กัน โดยพิมพ์ข้อความ "Thread 1" หรือ "Thread 2" ตามลำดับจากผลการทดลองพบว่า การสร้างเธรดโดยใช้ไลบรารี pthread สามารถทำได้อย่างง่ายดาย โดยเพียงแค่นำเข้าไลบรารี pthread ประกาศตัวแปรเพื่อเก็บข้อมูลของเธรด สร้างฟังก์ชันทำงาน (runnable function) ของเธรด และเรียกใช้ฟังก์ชัน pthread_create() เพื่อสร้างเธรดนอกจากนี้ ผลการทดลองยังแสดงให้เห็นว่าเธรดแต่ละเธรดจะทำงานซ้ำๆ กัน โดยทำงานแยกจากกัน ไม่ขึ้นต่อกันส่งผลให้สามารถทำงานหลายอย่างพร้อมกันได้