

Assignment ครั้งที่ 8 – Synchronization: Peterson, Semaphore

วัตถุประสงค์ของการปฏิบัติการ

- ทดลองการใช้คำสั่ง make และการสร้าง makefile
- ศึกษาการจัดการ Critical Section ด้วย Peterson Algorithm
- ศึกษาการจัดการ Critical Section ด้วย POSIX Semaphore

Remark เนื้อหา

Race Condition คือปัญหาการเข้าถึงหรือเรียกใช้งานทรัพยากรพร้อมกันของ Process/Thread ในโปรแกรม โดยที่เราจะเรียกส่วนของโปรแกรมที่มีการเข้าถึงตัวแปรพร้อมกันว่า Critical Section ซึ่งเป็นปัญหาที่จะก่อให้เกิดการทำงานที่ผิดพลาดของโปรแกรมได้

Peterson Algorithm เป็นอัลกอริทึมที่สามารถช่วยอำนวยความสะดวก Critical Section โดยสามารถสร้างกลไกเสมือนการล็อกคกุญแจพื้นที่ดังกล่าวให้มีการใช้งานเพียงแค่ Process/Thread เดียวเท่านั้น จะเรียกว่า Mutual Exclusion เมื่อ Process/Thread ดังกล่าวทำงานเสร็จก็จะปลดล็อกคกุญแจเพื่อให้พื้นที่ดังกล่าวสามารถเข้าใช้งานด้วย Process/Thread อื่นต่อไป Peterson Algorithm มีข้อจำกัดคือ สามารถทำการล็อกระหว่าง 2 Process/Thread เท่านั้น (Algorithm ที่พัฒนาต่อจาก Peterson เพื่อล็อกระหว่าง n โพรเซส คือ Bakery)

/* Process i */	/* Process j */
<pre> flag[i] = TRUE; turn = j; while (flag[j] && turn == j); -- CRITICAL SECTION -- flag[i] = FALSE; -- REMAINDER SECTION -- </pre>	<pre> flag[j] = TRUE; turn = i; while (flag[i] && turn == i); -- CRITICAL SECTION -- flag[j] = FALSE; -- REMAINDER SECTION -- </pre>

รูปที่ 1 แสดงหลักการจัดการ Critical Section ของ Peterson Algorithm ระหว่าง 2 Process i, j

Code ที่ 1 ให้นักศึกษาทำความเข้าใจกระบวนการทำงานของ Peterson Algorithm โดยการทดสอบ Compile และรันโปรแกรม

No.	File Name: osLab08_peterson.h
1	#define TRUE 1
2	#define FALSE 0
3	struct Memory {
4	int turn;
5	int flag[2];
6	};
7	void initializePeterson();
8	void removePeterson();
9	void enterCriticalSection(int i);
10	int exitCriticalSection(int i);
No.	File Name: osLab08_peterson.c
1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <unistd.h>
4	#include <sys/types.h>
5	#include <sys/ipc.h>
6	#include <sys/shm.h>
7	#include <sys/wait.h>
8	#include "peterson.h"
9	static struct Memory *ptr;
10	static int shmID = 0;
11	void initializePeterson() {
12	shmID = shmget(IPC_PRIVATE, sizeof(struct Memory),
	IPC_CREAT 0666);
13	if (shmID < 0) {
14	printf("*** shmget error (server) ***\n"); exit(1);
15	}
16	ptr = (struct Memory *) shmat(shmID, NULL, 0);
17	if (ptr == NULL) {
18	printf("*** shmat error (server) ***\n"); exit(1);
19	}
20	//initialize
21	ptr->turn = 0;
22	ptr->flag[0] = FALSE;

23	ptr->flag[1]= FALSE;
24	}
25	void removePeterson() {
26	shmdt((void *) ptr);
27	shmctl(shmID, IPC_RMID, NULL);
28	}
29	void enterCriticalSection(int i) {
30	int j = 0;
31	if (i == 0) {
32	j = 1;
33	} else {
34	j = 0;
35	}
36	ptr->turn = j; // Work if and only if this statement is atomic
37	ptr->flag[i] = TRUE; // Work if and only if this statement is atomic
38	while(ptr->flag[j] && ptr->turn == j);
39	}
40	int exitCriticalSection(int i) {
41	ptr->flag[i] = FALSE; // Work if and only it this statement is atomic
42	}
No.	File Name: osLab08_testPeterson.c
1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <unistd.h>
4	#include <string.h>
5	#include <sys/types.h>
6	#include <sys/ipc.h>
7	#include <sys/shm.h>
8	#include <sys/wait.h>
9	#include "peterson.h"
10	void childProcess(int *);
11	void parentProcess(int *);
12	int main(int argc, char *argv[]) {
13	int shmID, status, *count;
14	pid_t pid;
15	shmID = shmget(IPC_PRIVATE, sizeof(struct Memory),
	IPC_CREAT 0666);
16	if (shmID < 0) {

17	printf("*** shmget error (server) ***\n"); exit(1);
18	}
19	count = (int *) shmat(shmID, NULL, 0);
20	if ((int) *count == -1) {
21	printf("*** shmat error (server) ***\n"); exit(1);
22	}
23	*count = 5;
24	initializePeterson(); // Initialize shared memory for Peterson
25	pid = fork();
26	if (pid < 0) {
27	printf("*** fork error (server) ***\n");
28	exit(1);
29	}
30	else if (pid == 0) {
31	childProcess(count); // Child process do
32	exit(0);
33	}
34	parentProcess(count); // Parent process do
35	wait(&status);
36	printf("The final value of count is %d\n", *count);
37	removePeterson(); // Remove shared memory for Peterson
38	shmdt((void *) count); // Remove shared memory
39	shmctl(shmID, IPC_RMID, NULL);
40	return 0;
41	}
42	void parentProcess(int *count) {
43	enterCriticalSection(0);
44	int temp = *count; temp++;
45	sleep(rand() % 4);
46	*count = temp;
47	exitCriticalSection(0);
48	}
49	void childProcess(int *count) {
50	enterCriticalSection(1);
51	int temp = *count; temp--;
52	sleep(rand() % 4);
53	*count = temp;
54	exitCriticalSection(1);

55	}
Compile Program: gcc -o testPeterson osLab08_peterson.c osLab08_testPeterson.c	

Unix/Linux มีเครื่องมือเพื่อให้นักพัฒนาโปรแกรมสามารถเขียน Script สำหรับ Compile โปรแกรมที่มี Source Code หลายไฟล์ เครื่องมือดังกล่าวคือคำสั่ง “make” ดังอย่าง Script สำหรับ Compile โปรแกรม testPeterson แสดงดัง รูปที่ 2

```

1 all: testPeterson
2
3 testPeterson: osLab11_testPeterson.o osLab11_peterson.o
4     gcc -o testPeterson osLab11_testPeterson.o osLab11_peterson.o
5
6 testPeterson.o: osLab11_testPeterson.c
7     gcc -c osLab11_testPeterson.c
8
9 peterson.o: osLab11_peterson.c
10    gcc -c osLab11_peterson.c
11
12 clean:
13    rm -rf *.o testPeterson

```

รูปที่ 2 แสดงตัวอย่างของ Makefile Script (ชื่อไฟล์ Peterson.mk) สำหรับ Compile โปรแกรม testPeterson

ไวยากรณ์ใน makefile เป็นดังนี้ เรียกคำเรียกหน้า “:” ว่า target เช่น all: testPeterson ส่วนที่ตามหลัง target คือ รายชื่อไฟล์ที่จะนำมาสร้าง หรือ target เช่น all คือ default target ว่าให้ไปทำที่ target testPeterson เมื่อไปถึง testPeterson - make จะทราบว่า testPeterson ต้องใช้ testPeterson.o และ peterson.o

บรรทัดต่อจาก target line ใช้ tab เสมอ ไม่งั้น make จะไม่ทำงาน ส่วนไวยากรณ์สำหรับการ compile option - ของ gcc -c คือ compile ให้เป็น .o (object ไฟล์ ไม่ใช่ binary ไฟล์ (ซึ่งเป็น platform dependent)) .o นอกจากจะอำนวยความสะดวก platform dependent แล้ว ยังเป็นการให้ผู้พัฒนาสามารถ distribute code โดยไม่ต้องเผยแพร่ source code อีกด้วย

ให้นักศึกษาทดสอบ Compile Program ด้วยการใช้คำสั่ง make เช่น “make -f peterson.mk” หรือ หากต้องการ Clean Program ให้ใช้คำสั่ง “make -f Peterson clean”

1. Peterson's Algorithm

แก้ไข Code ที่ 2 ให้สามารถ Compile ได้

- ใช้ไฟล์ Source ของ Code ที่ 1 ในส่วนของ “osLab08_peterson.h” และ “osLab08_peterson.c” มาร่วมใช้ในการ Compile Program
- ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 1.1 - 1.5

Code ที่ 2 Synchronization with Peterson's Algorithm

No.	File Name: osLab08_peterson01.c
1	int main(int argc, char *argv[]) {
2	int shmID, status, *count;
3	int round = 10;
4	shmID = shmget(IPC_PRIVATE, sizeof(struct Memory),
	IPC_CREAT 0666);
5	count = (int *) shmat(shmID, NULL, 0);
6	*count = 99;
7	initializePeterson(); // Initialize Peterson's algorithm
8	if ((.....1.1.....)) {
9	for (int i = 0; i < round; i++) {
10	childProcess(count);
11	}
12	exit(0);
13	}
14	for (int i = 0; i < round; i++) {
15	(.....1.2.....)
16	}
17	wait(&status);
18	printf("The final value of count is %d\n", *count);
19	removePeterson(); // Remove Peterson's algorithm
20	shmdt((void *) count);
21	shmctl(shmID, IPC_RMID, NULL);
22	return 0;
23	}
24	void parentProcess(int *count) {
25	enterCriticalSection(0);
26	int temp = *count; temp++;
27	sleep(rand() % 2);
28	*count = temp;

29	(.....1.3.....)
30	}
31	void childProcess(int *count) {
32	(.....1.4.....)
33	int temp = *count; temp--;
34	sleep(rand() % 2);
35	*count = temp;
36	exitCriticalSection(1);
37	}

1.1	fork() == 0
1.2	parentProcess(count);
1.3	exitCriticalSection(0);
1.4	enterCriticalSection(1);
1.5	<div> แสดงผลลัพธ์จากการรันโปรแกรม <pre>jeadsada@salmon:/mnt/c/windows/system32\$./peterson01 The final value of count is 99</pre> </div>

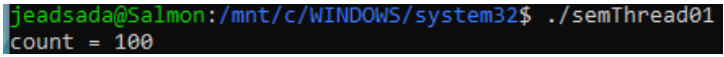
Semaphore เป็นกลไกที่ใช้ในการจัดการ Critical Section ที่ OS ในตระกูล Unix/Linux มีมาให้ แต่การเรียกใช้และกำหนดจังหวะเป็นหน้าที่ของผู้พัฒนา โดยมี 4 ฟังก์ชันดังต่อไปนี้

POSIX Semaphore	
1	int sem_init (sem_t *sem, int pshared, unsigned int value);
2	int sem_wait (sem_t *sem);
3	int sem_post (sem_t *sem);
4	int sem_destroy (sem_t *sem);

2. Semaphore

แก้ไข Code ที่ 3 (Client & Server) ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 2.1 - 2.7	
Code ที่ 3 Synchronization with Semaphore (1)	
No.	File Name: osLab08_semThread01.c
1	int count = 100;

	<pre> 2 sem_t mutex; 3 void *increment(void *param); 4 (.....2.1.....) int main() { 5 pthread_t tid1, tid2; 6 pthread_attr_t attr; 7 pthread_attr_init(&attr); 8 sem_init(&mutex, (.....2.2.....), 1); 9 pthread_create(&tid1, &attr, increment, NULL); 10 pthread_create(&tid2, &attr, decrement, NULL); 11 /* Not wait for the thread to exit */ 12 (.....2.3.....) 13 pthread_join(tid2, NULL); 14 printf("count = %d\n", count); 15 sem_destroy((.....2.4.....)); 16 return 0; 17 } 18 void *increment(void *param) { 19 int temp; 20 sem_wait(&mutex); 21 temp = count; 22 sleep(rand() % 5); temp+=1; 23 count = temp; 24 sem_post(&mutex); 25 pthread_exit(0); 26 } 27 (.....2.5.....){ 28 int temp; 29 sem_wait(&mutex); 30 temp = count; 31 sleep(rand() % 5); temp-=1; 32 count = temp; 33 sem_post(&mutex); 34 pthread_exit(0); 35 } </pre>	
2.1	void *decrement(void *param);	

- 2.2. 0
- 2.3. pthread_join(tid1, NULL);
- 2.4. &mutex
- 2.5. void *decrement(void *param)
- 2.6. จงแสดงผลลัพธ์ของการรันโปรแกรม.. 
- 2.7. ให้นักศึกษาทดสอบรันโปรแกรมด้วยคำสั่ง “for i in {1..10}; do ./osLab11_semThread01; done”
- ศึกษาและอธิบายผลลัพธ์จากการรัน ..ผลลัพธ์ถูกต้องครับ เนื่องจากคำสั่ง for loop สั่งให้โปรแกรมทำงานซ้ำ 10 รอบ ..
- และในแต่ละรอบจะมีการสร้าง 2 เธรดเพื่อบวกและลบค่าออกจาก 100 อย่างละหนึ่ง โดยมี Semaphore
- คอยป้องกันไม่ให้เกิด Race Condition จึงทำให้ผลลัพธ์สุดท้ายเป็น count = 100 เสมอในทุกครั้งที่รัน

แก้ไข Code ที่ 4 (Client & Server) ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 2.8 - 2.10

Code ที่ 4 Synchronization with Semaphore (2)

No.	File Name: osLab08_semThread02.c
1	#define NITER 100000
2	pthread_attr_t attr[2];
3	pthread_t tid[2];
4	sem_t mySemaphore;
5	int cnt;
6	void * Count(void* a) {
7	int i;
8	for (i = 0; i < NITER; i++) {
9	(.....2.8.....)
10	cnt++;
11	sem_post(&mySemaphore);
12	}
13	void* lastSeen = malloc(sizeof(int));
14	// LastSeen now refers to an actual piece of memory
15	if (pthread_self() == tid[0]) {
16	*(int *)lastSeen = cnt;
17	printf("thr %lu exits. lastSeen = %d\n", pthread_self(),
	*(int *)lastSeen);
18	pthread_exit((void*)lastSeen);
19	}
20	}

```

21
22 int main() {
23     (.....2.9.....)
24     pthread_attr_init(&attr[0]);
25     pthread_attr_init(&attr[1]);
26     pthread_create(&tid[0], &attr[0], Count, NULL);
27     pthread_create(&tid[1], &attr[1], Count, NULL);
28     void* returnVal;
29     pthread_join(tid[0], &returnVal);
30     int x = *(int*)returnVal;
31     printf("Last cnt from tid[0] is %d\n", x);
32     pthread_join(tid[1], NULL);
33     printf("final cnt = %d\n", cnt);
34     sem_destroy(&mySemaphore);
35     return 0;
36 }

```

2.8. sem_wait(&mySemaphore);

2.9. sem_init(&mySemaphore, 0, 1);

2.10. จงแสดงผลลัพธ์ของการรันโปรแกรม

```

jeadsada@Salmon:/mnt/c/WINDOWS/system32$ ./semThread02
thr 127994282636992 exits. lastSeen = 200000
Last cnt from tid[0] is 200000
final cnt = 200000

```