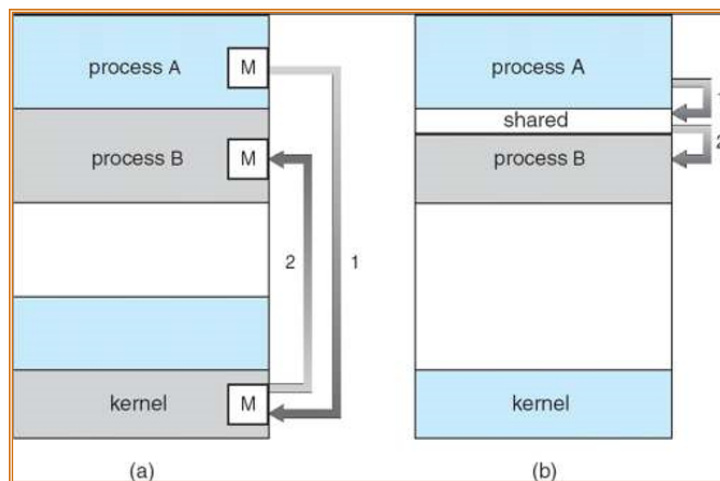


Assignment ครั้งที่ 7 – IPC: Shared Memory

วัตถุประสงค์ของการปฏิบัติการ

- สามารถ รับ/ส่ง ข้อมูลระหว่าง Process ผ่าน Shared memory บนระบบ UNIX/Linux OS ได้
- สามารถจัดจังหวะการรับ/ส่งข้อมูลระหว่าง Process (Process Synchronization) ได้

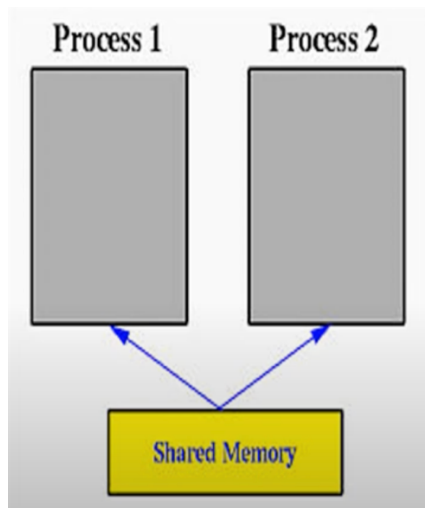
ทบทวนเนื้อหาคร่าว ๆ



รูปที่ 1 แสดง IPC (a) Message Passing และ (b) Shared Memory

จากรูปที่ 1 เป็นการแสดงรูปเพื่อจำลองกระบวนการระหว่างการสื่อสารของ Process (IPC) แบบ (a) Message Passing และแบบ (b) Shared memory โดยปฏิบัติการนี้จะมุ่งถึงการสื่อสารของ Process แบบ Shared Memory โดยที่ข้อดีของการสื่อสารแบบ Shared Memory คือทำงานได้รวดเร็วกว่าการส่งแบบ Message Passing เพราะไม่ต้องเสียเวลาในการ Copy ข้อมูลไปมาระหว่าง Process และสามารถกำหนดขนาดพื้นที่ที่ใช้ในการรับส่งข้อมูล (สื่อสาร) ระหว่าง Process ได้มากกว่าการสื่อสารแบบ Message Passing – แต่ข้อด้อย

ของการสื่อสารแบบ Shared Memory จะต้องจัดจังหวะของการรับส่งข้อมูล (สื่อสาร) ระหว่าง Process ด้วยตัวเอง



รูปที่ 2 แสดงการเข้าถึง Shared Memory ของ Processes

หัวใจที่สำคัญที่ทำให้วิธีการรับส่งข้อมูล (สื่อสาร) ระหว่าง Process แบบ Shared Memory คือ “Key” เปรียบเสมือนกุญแจที่ทั้ง 2 Process ที่ต้องการติดต่อสื่อสารกันจะต้องมีเหมือนกัน เพื่อให้เข้าถึงพื้นที่ที่ Shared เดียวกันได้ แสดงการเข้าถึงพื้นที่ที่ Shared ดังรูปที่ 2. โดยวิธีการสร้าง Key มีอยู่ด้วยกัน 3 แบบ ดังนี้

- 1) สร้างโดยกำหนดค่าคงที่แบบ Specific Integer Value เช่น “int shm_key = 1234”
- 2) สร้างโดยใช้ Function Call: `ftok()` – ใช้ในกรณี Process ที่ติดต่อสื่อสารกันไม่มีความสัมพันธ์เชิง Parent and Child
- 3) สร้างโดยใช้ “IPC_PRIVATE” – ใช้ในกรณี Process ที่ติดต่อสื่อสารกันมีความสัมพันธ์เชิง Parent and Child กล่าวคือทั้งสอง Process เกิดจากการ Fork

คำสั่งมาตรฐานสำหรับการสร้าง Shared Memory		
1	<code>int shmget(key_t key, size_t size, IPC 0666);</code>	จองพื้นที่บน Memory
2	<code>void *shmat(int shmid, NULL, 0);</code>	เชื่อมต่อแปรที่ต้องการ Shared ไปยัง Memory ที่จอง
3	<code>int shmdt(const void *shmaddr);</code>	ปลดการเชื่อมต่อแปรที่ต้องการ Shared จาก Memory ที่จอง
4	<code>int shmctl(int shmid, IPC_RMID, NULL);</code>	คืนพื้นที่ Memory ที่จอง

แสดงตัวอย่างโครงสร้างของการเขียนโปรแกรมแบบ Share Memory (Server) ดัง Code ที่ 1

Code ที่ 1 (ให้นักศึกษาทำความเข้าใจกระบวนการสร้าง Shared Memory ผ่านคำสั่งมาตรฐานทั้ง 4)

No.	File Name: osLab7_shmSkel.c
1	#include <stdio.h>
2	#include <sys/ipc.h>
3	#include <sys/shm.h>
4	
5	void main() {
6	/* ftok to generate unique key */
7	key_t key = ftok(".", 99);
8	
9	/* shmget return an identifier in shmid */
10	int shmid = shmget(key, 1024, 0666 IPC_CREAT);
11	
12	/* shmat to attach to shared memory */
13	char* str = (char*) shmat(shmid, NULL, 0);
14	
15	/* read original data in shared memory */
16	printf("Read from mem: %s\n", str);
17	
18	/* write new data to shared memory */
19	printf("Data written to mem: %s\n", str);
20	
21	/* shmdt to detach from shared memory */
22	shmdt(str);
23	
24	/* shmctl to destroy the shared memory */
25	shmctl(shmid, IPC_RMID, NULL);
26	}

1. การสร้าง Key เพื่อใช้ในการเข้าพื้นที่ Shared Memory กรณี Process มีความสัมพันธ์แบบ Parent and Child (เกิดจากการ fork()) จะให้การสร้าง Key แบบ “IPC_PRIVATE”

แก้ไข Code ที่ 2 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 1.1 - 1.5

Code ที่ 2 IPC using shared memory with “IPC_PRIVATE” for key generate (1)

No.	File Name: osLab7_shmFork01.c
1	#include <sys/ipc.h>

2	#include <sys/shm.h>
3	void main() {
4	int *count, shm_id;
5	/* get identifier of shared memory in shm_id */
6	//shm_id = shmget(key, sizeof(int), 0666 IPC_CREAT);
7	//shm_id = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT 0666);
8	/* shmat to attach to shared memory */
9	count = (.....1.2.....) shmat(shm_id, NULL, 0);
10	count[0] = 100; // initial value
11	if (fork() == 0) {
12	int temp = count[0]; sleep(1);
13	temp -= 1; count[0] = temp;
14	printf("Child decrements value at %p\n", &count);
15	exit(0);
16	}
17	wait(NULL);
18	int temp = count[0]; sleep(1);
19	temp += 1; count[0] = temp;
20	printf("Parent increments value at %p\n", &count);
21	sleep(3); printf("final answer is %d\n", count[0]);
22	/* shmdt to detach from shared memory */
23	(.....1.3.....)
24	/* shmctl to destroy the shared memory */
25	shmctl(shm_id, (.....1.4.....), NULL);
26	}

- 1.1. ในการสร้าง Key สำหรับใช้ในการเข้าถึง Shared Memory ของโปรแกรมจะเลือก Uncomment ในบรรทัดใดระหว่างบรรทัดที่ 6 กับ 7 เพราะอะไร ? ...Uncomment บรรทัดที่ 7 เพราะมัน Key ไม่ซ้ำ.....
- 1.2. (int*).....
- 1.3. shmdt(count);.....
- 1.4. IPC_RMID.....
- 1.5. หาก Comment ในบรรทัดที่ 17 แล้วทำการ Compile และทดสอบรันโปรแกรม ผลลัพธ์ของโปรแกรมจะให้ค่าได้กี่แบบ อะไรบ้าง เพราะอะไร ?

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./shmFork01_run
Child decrements value. Current value: 99
Parent increments value. Current value: 100
Final answer is 100
```

Parent process รอ Child process

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./shmFork01_run
Parent increments value. Current value: 101
Child decrements value. Current value: 99
Final answer is 99
```

Parent process ไม่ต้องรอ Child process

แก้ไข Code ที่ 3 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 1.6 - 1.10

Code ที่ 3 IPC using shared memory with “IPC_PRIVATE” for key generate (2)

No.	File Name: osLab7_shmFork02.c
1	void main(int argc, char *argv[]) {
2	int shm_id, *shm_addr;
3	pid_t pid;
4	if (argc != 5) {
5	printf("Use: %s #1 #2 #3 #4\n", argv[0]); exit(1);
6	}
7	shm_id = (.....1.6.....)
8	if (shm_id < 0) {
9	printf("*** shmget error (server) ***\n"); exit(1);
10	}
11	printf("Server has received a shared memory of four integers ...\n");
12	shm_addr = (.....1.7.....)
13	if (shm_addr == NULL) {
14	printf("*** shmat error (server) ***\n"); exit(1);
15	}
16	printf("Server has attached the shared memory ...\n");
17	/* Set value from argv to shared memory */
18	shm_addr[0] = (.....1.8.....) (argv[1]);
19	shm_addr[1] = (.....1.8.....) (argv[2]);
20	shm_addr[2] = (.....1.8.....) (argv[3]);
21	shm_addr[3] = (.....1.8.....) (argv[4]);
22	printf("Server has filled %d %d %d %d in shared memory ...\n",
	shm_addr[0], shm_addr[1], shm_addr[2], shm_addr[3]);
23	printf("Server is about to fork a child process ...\n");
24	if ((pid = fork()) < 0) {
25	printf("*** fork error (server) ***\n"); exit(1);
26	} else if (pid == 0) {

27	clientProcess(.....1.9.....); exit(0);
28	} else {
29	wait(NULL);
30	printf("Server has detected the completion of its child ...\n");
31	shmdt((void *) shm_addr);
32	printf("Server has detached its shared memory ...\n");
33	(.....1.10.....)
34	printf("Server has removed its shared memory ...\n");
35	printf("Server exits ...\n");
36	}
37	}
38	void clientProcess(int shm_addr[]) {
39	printf("\tClient process started\n");
40	printf("\tClient found %d %d %d %d in shared memory\n",
	shm_addr[0], shm_addr[1], shm_addr[2], shm_addr[3]);
41	printf("\tClient is about to exit\n");
42	}

1.6. shmget(IPC_PRIVATE, 4 * sizeof(int), IPC_CREAT | 0666)

1.7. (int *) shmat(shm_id, NULL, 0)

1.8. (* มีคำตอบเดียว) atoi

1.9. shm_addr

1.10. shmctl(shm_id, IPC_RMID, NULL)

2. การสร้าง Key เพื่อใช้ในการเข้าพื้นที่ Shared Memory กรณี Process ไม่มีความสัมพันธ์แบบ Parent and Child จะให้การสร้าง Key แบบ “ftok()” Function Call

แก้ไข Code ที่ 4 (Client & Server) ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 2.1 - 2.10

Code ที่ 4 สร้าง IPC Shared Memory ระหว่าง Server และ Client Process โดยใช้ “kill()” Function Call ในการจัดจ้งวะ (Synchronization)

No.	File Name: osLab7_shmServer01.c
1	char *shm_addr; int isLoop = 1; // Global variables
2	void SIGHandler(int sig) {

3	signal(sig, SIG_IGN);
4	printf("-\tSIGHandler on server catch a signal %d from client ...\n", sig);
5	isLoop = 0; signal(sig, SIGHandler); // Signal register
6	}
7	void main() {
8	(.....2.1.....) // Signal register
9	key_t key = ftok(".", 99);
10	int shm_id = shmget(.....2.2.....);
11	shm_addr = (char *) shmat(shm_id, NULL, 0);
12	sprintf(shm_addr, "%d", getpid());
13	printf("(1) [Server] Writing the server pid = %s to shared memory ...\n", shm_addr);
14	while (isLoop); isLoop = 1;
15	int client_pid = atoi(shm_addr);
16	printf("(4) [Server] Reading the client pid = %s from shared memory ...\n", shm_addr);
17	kill(client_pid, (.....2.3.....)); // Send signal
18	printf("-\tServer send a signal %d to client ...\n", SIGUSR2);
19	while (isLoop);
20	printf("(6) [Server] Reading data \"%s\" from shared memory ...\n", shm_addr);
21	shmdt(shm_addr);
22	(.....2.4.....)
23	}
No.	File Name: osLab7_shmClient01.c
1	char *shm_addr; int isLoop = 1; // Global variables
2	void SIGHandler(int sig) {
3	signal(sig, SIG_IGN);
4	printf("-\tSIGHandler on client catch a signal %d from server ...\n", sig);
5	isLoop = 0; signal(sig, SIGHandler); // Signal register
6	}
7	void main() {
8	signal(SIGUSR2, SIGHandler); // Signal register
9	key_t key = (.....2.5.....)
10	int shm_id = shmget(.....2.6.....);
11	shm_addr = (char *) shmat(shm_id, NULL, 0);

12	int server_pid = atoi(shm_addr);
13	printf("(2) [Client] Reading the server pid = %s from shared memory ...\n", shm_addr);
14	sprintf(shm_addr, "%d", getpid());
15	printf("(3) [Client] Writing the client pid = %s to shared memory ...\n", shm_addr);
16	kill(server_pid, (.....2.7.....)); // Send signal
17	printf("-\tClient send a signal %d to server ...\n", SIGUSR1);
18	while (isLoop);
19	sprintf(shm_addr, "%s", "CS KMITL");
20	printf("(5) [Client] Writing data \"\n\" to shared memory ...\n", shm_addr);
21	kill(server_pid, (.....2.8.....)); // Send signal
22	printf("-\tClient send a signal %d to server ...\n", SIGUSR1);
23	(.....2.9.....)
24	}
หมายเหตุ: การรันโปรแกรมให้ใช้ 2 Terminal โดยรัน Server ก่อน 1 Terminal และตามด้วย การรัน Client อีก 1 Terminal	

2.1 signal(SIGUSR1, SIGHandler); signal(SIGUSR2, SIGHandler);

2.2 key_1024, IPC_CREAT | 0666

2.3 SIGUSR2

2.4 shmctl(shm_id, IPC_RMID, NULL);

2.5 ftok(".", 99)

2.6 key_1024, 0666

2.7 SIGUSR1

2.8 SIGUSR1

2.9 shmdt(shm_addr);

2.10. ทำการรันโปรแกรมและศึกษาผลลัพธ์ – จงอธิบายหลักการจัดจังหวะของทั้งสอง Process ในการเข้าใช้งานพื้นที่ Shared Memory ด้วยการส่ง Signal ตามความเข้าใจของนักศึกษา
 Server เขียน PID ของตัวเองลง Shared Memory → รอจน Client ส่ง SIGUSR1 มา
 Client อ่าน PID ของ Server → เขียน PID ของตัวเองกับลง Shared Memory → ส่ง SIGUSR1 ไปบอก Server
 Server รับ SIGUSR1 แล้วอ่าน PID ของ Client → ส่ง SIGUSR2 ตอบกลับ
 Client รอ SIGUSR2 → เมื่อได้แล้วจึงเขียนข้อความ "CS KMITL" ลง Shared Memory → ส่ง SIGUSR1 อีกครั้ง

Server ได้สัญญาณจึงอ่านข้อความจาก Shared Memory แล้วจบการทำงาน

อาจารย์: อนุพงษ์ บรรจงการ

29/08/2568


```

lead sada@Salmon: /mnt/c/Users/feuku$ nano osLab7_shmServer01.c
lead sada@Salmon: /mnt/c/Users/feuku$ gcc -Wall -o osLab7_shmServer01 osLab7_shmServer01.c
lead sada@Salmon: /mnt/c/Users/feuku$ ./osLab7_shmServer01
(1) [Server] Writing the server pid = 668 to shared memory ...
- SIGHandler on server catch a signal 10 from client ...
(u) [Server] Reading the client pid = 674 from shared memory ...
- Server send a signal 12 to client ...
- SIGHandler on server catch a signal 10 from client ...
(6) [Server] Reading data "CS KMITL" from shared memory ...
lead sada@Salmon: /mnt/c/Users/feuku$

lead sada@Salmon: /mnt/c/Users/feuku$ nano osLab7_shmClient01.c
lead sada@Salmon: /mnt/c/Users/feuku$ gcc -Wall -o osLab7_shmClient01 osLab7_shmClient01.c
lead sada@Salmon: /mnt/c/Users/feuku$ ./osLab7_shmClient01
(2) [Client] Reading the server pid = 668 from shared memory ...
(3) [Client] Writing the client pid = 674 to shared memory ...
- Client send a signal 10 to server ...
- SIGHandler on client catch a signal 12 from server ...
(5) [Client] Writing data "CS KMITL" to shared memory ...
- Client send a signal 10 to server ...
lead sada@Salmon: /mnt/c/Users/feuku$

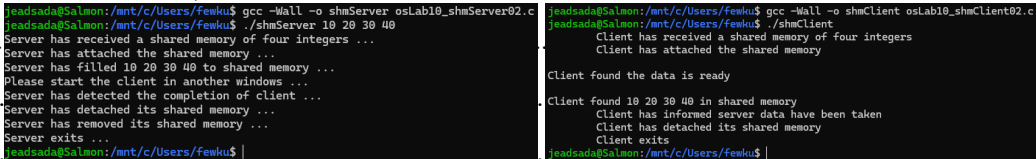
```

แก้ไข Code ที่ 5 (Client & Server) ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 2.11 - 2.15

Code ที่ 5 สร้าง IPC Shared Memory ระหว่าง Server และ Client Process โดยใช้ Shared Variable ในการจัดจ้งวะ (Synchronization)

No.	File Name: osLab10_myShm.h
1	#define NOT_READY -1
2	#define FILLED 0
3	#define TAKEN 1
4	struct Memory {
5	int status; // for sync server & client
6	int data[4];
7	};
No.	File Name: osLab10_shmServer02.c
1	#include <sys/shm.h>
2	#include <sys/ipc.h>
3	#include "osLab10_myShm.h"
4	void main(int argc, char *argv[]) {
5	key_t shm_key; int shm_id;
6	struct Memory *shm_ptr;
7	if ((.....2.11.....)) {
8	printf("Use: %s #1 #2 #3 #4\n", argv[0]); exit(1);
9	}
10	shm_key = ftok(".", 111);
11	shm_id = shmget((.....2.12.....));
12	if (shm_id < 0) { printf("*** shmget error ***\n"); exit(1); }
13	printf("Server has received a shared memory of four integers ...\n");
14	shm_ptr = (struct Memory *) shmat(shm_id, NULL, 0);
15	if (shm_ptr == NULL) { printf("*** shmat error ***\n"); exit(1); }
16	printf("Server has attached the shared memory ...\n");
17	shm_ptr->status = NOT_READY; // Sync server and client
18	shm_ptr->data[0] = atoi(argv[1]);
19	shm_ptr->data[1] = atoi(argv[2]);
20	shm_ptr->data[2] = atoi(argv[3]);

21 22 23 24 25 26 27 28 29 30	<pre> shm_ptr->data[3] = atoi(argv[4]); printf("Server has filled %d %d %d %d to shared memory ...\n", shm_ptr->data[0], shm_ptr->data[1], shm_ptr->data[2], shm_ptr->data[3]); shm_ptr->status = (.....2.13.....); // Sync server and client printf("Please start the client in another windows ...\n"); while (shm_ptr->status != TAKEN) sleep(1); printf("Server has detected the completion of client ...\n"); (.....2.14.....) printf("Server has detached its shared memory ...\n"); shmctl(shm_id, IPC_RMID, NULL); printf("Server has removed its shared memory ...\n"); printf("Server exits ...\n"); } </pre>	
No.	File Name: osLab10_shmClient02.c	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	<pre> void main() { key_t shm_key; int shm_id; struct Memory *shm_ptr; shm_key = (.....2.15.....) shm_id = shmget(shm_key, sizeof(struct Memory), 0666); if (shm_id < 0) { printf("*** shmget error ***\n"); exit(1); } printf("\tClient has received a shared memory of four integers\n"); shm_ptr = (struct Memory *) shmat(shm_id, NULL, 0); if (shm_ptr == NULL) { printf("*** shmat error ***\n"); exit(1); } printf("\tClient has attached the shared memory\n"); while (shm_ptr->status != (.....2.16.....)); printf("\nClient found the data is ready\n"); printf("\nClient found %d %d %d %d in shared memory\n", shm_ptr->data[0], shm_ptr->data[1], shm_ptr->data[2], shm_ptr->data[3]); shm_ptr->status = TAKEN; printf("\tClient has informed server data have been taken\n"); shmdt((void *) shm_ptr); printf("\tClient has detached its shared memory\n"); } </pre>	

18 19	<pre>printf("\tClient exits\n"); }</pre>	
2.11.	argc != 5	
2.12.	shm_key, sizeof(struct Memory), IPC_CREAT 0666	
2.13.	FILLED	
2.14.	shmdt((void *)shm_ptr);	
2.15.	ftok(".", 111)	
2.16.	FILLED	
2.17.	<p>ในบรรทัดที่ 5 ของไฟล์ osLab10_shmClient02.c – Parameter ตัวที่ 3 “0666” บ่งบอกถึงอะไร? 0666 บอกสิทธิ์ว่า ทุก process สามารถอ่าน/เขียน shared memory ได้</p>	
2.18.	<p>ทำการรันโปรแกรมและศึกษาผลลัพธ์ – จงอธิบายหลักการจัดจังหวะของทั้งสอง Process ในการเข้าใช้งานพื้นที่ Shared Memory ด้วยการใช้ตัวแปร “status” (Shared Variable) ตามความเข้าใจของนักศึกษา</p>	
นักศึกษา		
	<pre>status = NOT_READY → บอกว่า memory ยังไม่พร้อม</pre>	
	<pre>Server เขียนข้อมูลเสร็จ → เปลี่ยน status = FILLED → บอก Client ว่าข้อมูลพร้อมแล้ว</pre>	
	<pre>Client รอจนเจอ status = FILLED → อ่านข้อมูลเสร็จ → เปลี่ยน status = TAKEN</pre>	
	<pre>Server เห็น status = TAKEN → รู้ว่า Client ใช้นข้อมูลเสร็จแล้ว → สามารถลบ/เขียนใหม่ได้</pre>	