SCI19 2243 Operating Systems (1/2568)

สาขาวิชาคณิตศาสตร์และภูมิสารสนเทศ สำนักวิชาวิทยาศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

รหัสนักศึกษา B6638146 ชื่อ-สกุล นายเจษฎา ธัญญะ

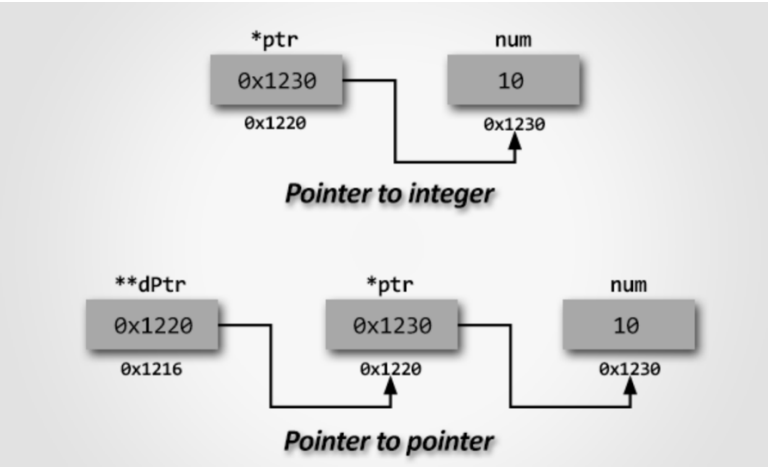
Assignment ครั้งที่ 4 – POSIX Thread (pthread)

วัตถุประสงค์ของการปฏิบัติการ

- อธิบายการทำงานของ Thread ได้
- ใช้งาน pthread ได้
- สามารถส่งค่า (Argument Passing) และรับค่ากลับ (Return Data Through) จากการใช้ pthread ได้

1. ทบทวน Pointer in C

Pointer เป็นเครื่องมือที่ใช้ในการอ้างถึง Memory ในภาษา C ทำให้เราสามารถเข้าถึงและจัดการข้อมูลที่อยู่บน Memory ได้ รูปแบบการใช้งาน Pointer จะใช้สัญญาลักษณ์ “ * ” (dereferencing operator) เป็นการระบุตัวแปร Pointer เช่น “int *prt” และอีกหนึ่งสัญลักษณ์ที่ใช้คู่กับ Pointer คือ “ & ” (address of operator) ใช้สำหรับแสดงค่าตำแหน่งของบน Memory ที่ตัวแปรถือครองอยู่ เช่น “&num” แสดงหลักการทำงานของ Pointer ในภาษา C ดังรูปที่ 1



รูปที่ 1 แสดงการอ้างถึง Memory ของ Singer Pointer และ Double Pointer

แก้ไข Code ที่ 1 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 1.1 - 1.4

Code ที่ 1

No.	File Name: osLab7-01.c
1	void runner() {
2	int num = 10;
3	int *ptr; ptr = #
4	int **dPtr; dPtr = &ptr;
5	printf("Value at num = %d\n", num);
6	printf("Address of num = %p\n\n", &num);
7	printf("Value at ptr = ...**1.1**.....\n", ptr);
8	printf("Address of ptr = %p\n", &ptr);
9	printf("Value at *ptr = %d\n\n", *ptr);
10	printf("Value at dPtr = %p\n", dPtr);
11	printf("Address of dPtr = %p\n", ...**1.2**.....);
12	printf("Value at **dPtr = %d\n\n", **dPtr);
13	}
14	void main() {
15	.....**1.3**.....
16	}

1.1. printf("Value at ptr = %p\n", ptr);

1.2. printf("Address of dPtr = %p\n", &dPtr);

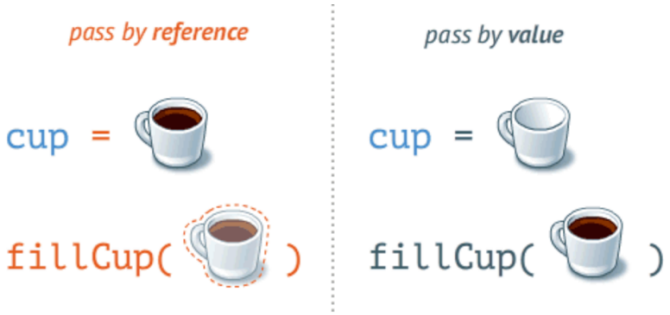
1.3. runner();
return0;

1.4. จงแสดงผลลัพธ์ที่ได้จากการรันโปรแกรม

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./pthread01
Value at num = 10
Address of num = 0x7fff94e9b564
Value at ptr = 0x7fff94e9b564
Address of ptr = 0x7fff94e9b568
Value at *ptr = 10
Value at dPtr = 0x7fff94e9b568
Address of dPtr = 0x7fff94e9b570
Value at **dPtr = 10
jeadsada@Salmon:/mnt/c/Users/fewku$
```

2. ทบทวนการส่งค่า Pass-by-Value & Pass-by-Reference

การส่ง Parameter ใน Structure Programming เช่น ภาษา C มีการส่ง 2 แบบคือ ส่งแบบ “Pass-by-Value” และส่งแบบ “Pass-by-Reference” ในการส่งข้อมูลที่มีโครงสร้างแบบ Primitive เป็น Parameter จะนิยมส่งแบบ “Pass-by-Value” ซึ่งหมายถึงการทำสำเนา (Local Variable) ส่งไป ทำให้หากมีการเปลี่ยนแปลงค่าก็จะไม่ส่งผลต่อค่าเดิม แต่หากส่งแบบ “Pass-by-Reference” จะหมายถึงเป็นการส่ง Address ของตัวแปร ทำให้หากมีการเปลี่ยนค่าที่ตัวแปรที่ส่งไปค่าในตัวแปรเดิมก็จะถูกเปลี่ยนด้วย เพราะคือตัวแปรเดียวกัน แสดงการส่งค่าทั้งสองแบบโดยเปรียบเทียบได้ดังรูปที่ 2 สังเกตว่า การส่ง Array (Non-primitive) จะเป็นการส่งแบบ “Pass-by-Reference” โดยอัตโนมัติ



รูปที่ 2 เปรียบเทียบการส่งค่าแบบ Pass-by-reference กับ Pass-by-value

แก้ไข Code ที่ 2 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 2.1 - 2.4

Code ที่ 2

No.	File Name: osLab7-02.c
1	void passByValue(int argv) {
2	argv += 1;
3	printf("During pass-by-value: %d\n", argv);
4	}
5	void passByReference(int .....**2.1**.....) {
6	*argv += 1;
7	printf("During pass-by-reference: %d\n", *argv);
8	}
9	void swap(char *argv1, char *argv2) {
10	char tmp = *argv1;
11	.....**2.2**.....
12	*argv2 = tmp;
13	}

```
14  void passByArray(int argv[]) {
15      argv[0] = 999;
16  }
17  void main() {
18      int num = 10;
19      printf("Before pass-by-value: %d\n", num);
20      passByValue(num);
21      printf("After pass-by-value: %d\n\n", num);
22      printf("Before pass-by-reference: %d\n", num);
23      passByReference(.....2.3.....);
24      printf("After pass-by-reference: %d\n\n", num);
25      char var1 = 'a'; char var2 = 'b';
26      printf("Before swap: var1 = %c, var2 = %c\n", var1, var2);
27      swap(&var1, &var2);
28      printf("After swap: var1 = %c, var2 = %c\n\n", var1, var2);
29      int arr[] = {777, 888};
30      printf("Before pass:\t {%d, %d}\n", arr[0], arr[1]);
31      passByArray(arr);
32      printf("After pass:\t {%d, %d}\n", arr[0], arr[1]);
33  }
```

2.1. void passByReference(int *argv)

2.2. *argv1 = *argv2;

2.3. passByReference(&num);

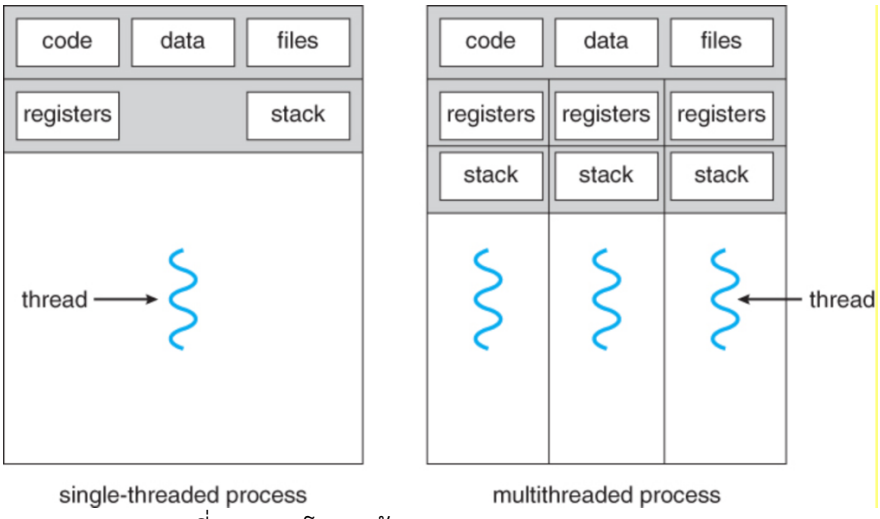
2.4. จงแสดงผลลัพธ์ที่ได้จากการรันโปรแกรม

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./pthread02
Before pass-by-value: 10
During pass-by-value: 11
After pass-by-value: 10
Before pass-by-reference: 10
During pass-by-reference: 11
After pass-by-reference: 11
Before swap: var1 = a, var2 = b
After swap: var1 = b, var2 = a
Before pass:     {777, 888}
After pass:      {999, 888}
jeadsada@Salmon:/mnt/c/Users/fewku$
```

.....

.....

pthread (POSIX threading interface) เป็นเครื่องมือหรือ Library ที่รองรับใน OS ตระกูล UNIX และ Linux ซึ่งใช้ในการสร้าง Thread เรียกอีกชื่อว่า Light-weight process ซึ่ง Thread ใช้ทรัพยากรน้อยกว่า Process ปกติ เพราะเมื่อเรียกใช้งาน Thread จะมีทรัพยากรบางส่วนที่ใช้งานร่วมกันกับ Process หลัก แสดงดังรูปที่ 3 ข้อดีของ Thread คือสามารถสร้างโปรแกรมแบบขนาดได้ (Parallelism) ทำให้การทำงานของโปรแกรมมีประสิทธิภาพ ทำงานได้เร็วและเกิดการใช้งานทรัพยากรได้คุ้มประสิทธิภาพ



รูปที่ 3 แสดงโครงสร้างของ Process และ Thread

Common pthread functions

- Create Thread:

```
int pthread_create(pthread_t *tid, pthread_attr_t *attr, void *function, void *argv)
```

- Joining (reaping) Thread

```
Int pthread_join(pthread_t tid, void **return_value)
```

- Terminate Thread

```
void pthread_exit(void *return_value)
```

3. pthread: Argument Passing

แก้ไข Code ที่ 3 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 3.1 - 3.4

Code ที่ 3

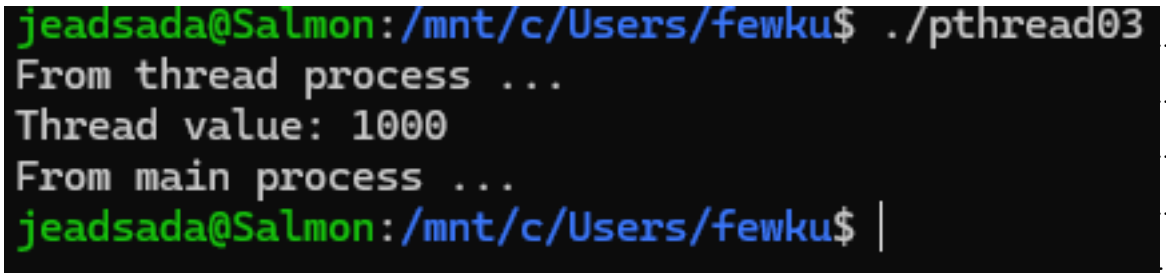
No.	File Name: osLab7-03.c
1	static int input = 999;
2	void *myfunc(void *argv) {
3	int value = *((int *)argv);
4	printf("From thread process ...\n");
5	printf("Thread value: %d\n", value);
6	.....**3.1**.....
7	}
8	pthread_t launch_thread(void) {
9	input += 1;
10	pthread_t tid;
11	pthread_attr_t tattr;
12	pthread_attr_init(&tattr);
13	pthread_create(&tid, &tattr, myfunc, &input);
14	return .....**3.2**.....;
15	}
16	void main(void) {
17	pthread_t tid = ...**3.3**.....;
18	pthread_join(tid, NULL);
19	printf("From main process ...\n");
20	}

3.1. pthread_exit(NULL);

3.2. return tid;

3.3. pthread_t tid = launch_thread();

3.4. จงแสดงผลลัพธ์ที่ได้จากการรันโปรแกรม



แก้ไข Code ที่ 4 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 3.5 - 3.8

Code ที่ 4

No.	File Name: osLab7-04.c
1	#define NUM_TASKS 4
2	char *messages[NUM_TASKS];
3	void *printHello(void *argv) {
4	sleep(1);
5	int taskID = ...**3.5**.....argv);
6	printf("Task %d: %s\n", taskID, messages[taskID]);
7	pthread_exit(0);
8	}
9	void main(int argc, char *argv[]) {
10	pthread_t threads[NUM_TASKS];
11	int *input;
12	int i, t;
13	for (i = 0; i < NUM_TASKS; i++) {
14	messages[i] = "Hello KMITL";
15	}
16	for (t = 0; t < NUM_TASKS; t++) {
17	input = ...**3.6**.....;
18	printf("Create thread %d\n", t);
19	pthread_create(&threads[t], 0, printHello, (void *) input);
20	pthread_join(.....**3.7**.....);
21	}
22	}

3.5. int taskID = (int)argv;

3.6. input = (int *)t;

3.7. pthread_join(threads[t], NULL);

3.8. จงแสดงผลลัพธ์ที่ได้จากการรันโปรแกรม

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./pthread04
Create thread 0
Create thread 1
Create thread 2
Create thread 3
Task 0: Hello KMITL
Task 2: Hello KMITL
Task 1: Hello KMITL
Task 3: Hello KMITL
jeadsada@Salmon:/mnt/c/Users/fewku$
```

.....

.....

.....

.....

4. pthread: Return Data

แก้ไข Code ที่ 5 ให้สามารถ Compile ได้ ทดสอบรัน ศึกษาผลลัพธ์และตอบคำถาม 4.1 - 4.5

Code ที่ 5

No.	File Name: osLab7-05.c
1	typedef struct mydata {
2	double d;
3	int i;
4	} data;
5	.....**4.1**..... *myfunc(void *argv) {
6	data *my = (data *)malloc(sizeof(data));
7	my->d = 3.14;
8	my->i = 40;
9	pthread_exit(.....**4.2**.....);
10	}
11	void main() {
12	pthread_t tid;
13	data my;
14	void *retval;
15	pthread_create(&tid, NULL, myfunc, ...**4.3**.....);
16	pthread_join(tid, &retval);
17	my = *((.....**4.4**.....)retval);
18	free(retval);
19	printf("%f, %d\n", my.d, my.i);
20	}

4.1. void *myfunc(void *argv) .....

4.2. pthread_exit(my); .....

4.3. pthread_create(&tid, NULL, myfunc, NULL); .....

4.4. my = *((data *)retval); .....

4.5. จงแสดงผลลัพธ์ที่ได้จากการรันโปรแกรม

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./pthread05
3.140000, 40
jeadsada@Salmon:/mnt/c/Users/fewku$
```

5. pthread your program

จงพัฒนาโปรแกรมภาษา C โดยใช้ pthread สร้าง Thread ในการทำงาน โดยมีเงื่อนไขดังต่อไปนี้

1. รับเลขจำนวนเต็มบวกจากผู้ใช้หนึ่งจำนวน โดยใช้การส่งผ่าน argument ขณะรันโปรแกรม
2. ให้สร้าง Thread เพื่อคำนวณผลรวมจาก 1 ถึงสองเท่าของค่าจำนวนเต็มบวกที่รับจากผู้ใช้
3. ให้ Main process คำนวณผลรวมจาก 1 ถึงค่าจำนวนเต็มบวกที่รับจากผู้ใช้
4. แสดงผลต่างระหว่างผลของการคำนวณทั้งสอง โดยให้ผลรวมที่คำนวณได้จาก Thread ลบผลรวมที่คำนวณได้จาก Main process
5. สามารถใช้วิธีคืนค่าจากการคำนวณของ Thread ผ่าน “pthread_exit(void *retval)” หรือใช้ตัวแปรแบบ Global Variable
6. ตัวอย่างการรันโปรแกรมและการแสดงผลลัพธ์ ดังรูปที่ 4

```
anupong@cms:~/os_lab/lab07$ ./a.out 5
From thread process ...
csum = 55
From main process ...
msum = 15
Difference csum and msum = 40
anupong@cms:~/os_lab/lab07$ ./a.out 3
From thread process ...
csum = 21
From main process ...
msum = 6
Difference csum and msum = 15
anupong@cms:~/os_lab/lab07$ ./a.out 100
From thread process ...
csum = 20100
From main process ...
msum = 5050
Difference csum and msum = 15050
anupong@cms:~/os_lab/lab07$
```

รูปที่ 4 ตัวอย่างการรัน และผลลัพธ์จากการรันโปรแกรม

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h> // For error checking with strtoll

// A structure to pass arguments to the thread.
// This is a robust way to pass multiple or large data types.
typedef struct {
    long long limit;
} thread_args_t;

/**
 * @brief Calculates the sum of integers from 1 to n.
 * Uses the arithmetic series formula for O(1) efficiency.
 * @param n The upper limit of the summation.
 * @return The sum from 1 to n.
 */
long long calculate_sum(long long n) {
    // Using the formula n * (n + 1) / 2 to avoid slow loops
    // This is much more efficient for large numbers.
    return n > 0 ? (n * (n + 1) / 2) : 0;
}

/**
 * @brief The thread's main function.
 * Calculates the sum up to 2 * limit.
 * @param arg A pointer to thread_args_t.
 * @return A pointer to the calculated sum on the heap.
 */
void *sum_runner(void *arg) {
    thread_args_t *args = (thread_args_t *)arg;
    long long limit = args->limit;
    free(arg); // Free the argument structure as soon as we've copied the data.

    long long thread_sum = calculate_sum(limit * 2);

    printf("From thread process ...\n");
    printf("csum = %lld\n", thread_sum);

    // To safely return a value from a thread, we must allocate memory on the heap.
    // The calling thread (main) is responsible for freeing this memory.
    long long *result_ptr = malloc(sizeof(long long));
    if (result_ptr == NULL) {
        // Not much we can do here, but in a real app, you might log this.
        pthread_exit(NULL);
    }
    *result_ptr = thread_sum;

    pthread_exit((void *)result_ptr);
}

int main(int argc, char **argv) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <positive_integer>\n", argv[0]);
        return 1; // Use stderr for errors and return non-zero status
    }

    // Use strtoll for robust number parsing and error checking
    char *endptr;
    errno = 0; // Reset errno before the call
    long long limit = strtoll(argv[1], &endptr, 10);

    // Check for conversion errors
    if (errno != 0 || *endptr != '\0' || limit <= 0) {
        fprintf(stderr, "Error: Invalid input. Please provide a single positive integer.\n");
        return 1;
    }

    // Prepare arguments for the thread
    thread_args_t *thread_args = malloc(sizeof(thread_args_t));
    if (thread_args == NULL) {
        perror("Failed to allocate memory for thread arguments");
        return 1;
    }
    thread_args->limit = limit;

    pthread_t tid;
    // We don't need custom attributes, so passing NULL is cleaner.
    if (pthread_create(&tid, NULL, sum_runner, thread_args) != 0) {
        perror("Failed to create thread");
        free(thread_args); // Clean up memory on failure
        return 1;
    }

    // Main process calculates its sum
    long long main_sum = calculate_sum(limit);

    // Wait for the thread to finish and get its result
    long long *thread_result_ptr = NULL;
    if (pthread_join(tid, (void **)&thread_result_ptr) != 0) {
        perror("Failed to join thread");
        // If join fails, we might not be able to free thread_result_ptr
        // as it might not have been set.
        return 1;
    }

    // Check if the thread returned a valid result
    if (thread_result_ptr == NULL) {
        fprintf(stderr, "Thread failed to return a result.\n");
        return 1;
    }
    long long thread_sum = *thread_result_ptr;
    free(thread_result_ptr); // IMPORTANT: Free the memory allocated by the thread.

    printf("From main process ...\n");
    printf("msum = %lld\n", main_sum);
    printf("Difference csum and msum = %lld\n", thread_sum - main_sum);

    return 0;
}
```

```
jeadsada@Salmon:/mnt/c/Users/fewku$ ./a.out 5
From thread process ...
csum = 55
From main process ...
msum = 15
Difference csum and msum = 40
jeadsada@Salmon:/mnt/c/Users/fewku$ ./a.out 3
From thread process ...
csum = 21
From main process ...
msum = 6
Difference csum and msum = 15
jeadsada@Salmon:/mnt/c/Users/fewku$ ./a.out 100
From thread process ...
csum = 20100
From main process ...
msum = 5050
Difference csum and msum = 15050
jeadsada@Salmon:/mnt/c/Users/fewku$
```