204324 ปฏิบัติการระบบคอมพิวเตอร์

ปฏิบัติการที่ 6: Process กับ thread และการโปรแกรมแบบขนาน

ก่อนที่จะเริ่มต้นปฏิบัติการนี้ ขอให้ทุกคนเตรียมเครื่องและ environment ต่างๆให้พร้อมเพื่อให้สามารถรันโปรแกรมที่อยู่ใน zip ไฟล์ทั้งสองที่แนบมาได้

- unzip ไฟล์
- cd เข้าไปที่ directory cpu-api หรือ threads-intro
- พิมพ์ make
- ได้ executable ที่เกี่ยวข้องทั้งหมด
- ทดลองรันตัว executable

ผู้สอนจะมีคำถามเกี่ยวกับการรับโปรแกรมเหล่านี้ ขอให้ทุกคนเตรียมพร้อมตอบคำถาม

หลังจากตอบคำถามผู้สอนเรียบร้อยแล้ว ร่วมกันตอบคำถามต่อไปนี้กับเพื่อนที่ทำปฏิบัติการคู่กัน

1. ให้ว่า PID ของ child process คือ 91122 ผลลัพธ์ทั้งหมดที่เป็นไปได้จากการรันโปรแกรมด้านล่างนี้คืออะไร อธิบายว่าทำไมจึงได้ผลลัพธ์เช่นนี้

```
int main() {
    pid_t pid = fork();
    printf("Hello World: %d\n", pid);
}
```

2. มี process ใหม่ที่เกิดขึ้นกี่ process หลังจากโปรแกรมด้านล่างทำงานสิ้นสุด อธิบายโดยใช้ process graph

```
int main(void) {
  for (int i = 0; i < 3; i++) {
      pid_t pid = fork();
  }
}
```

3. โปรแกรมด้านล่างนี้พิมพ์อะไรออกมา เพราะเหตุใด

```
int main(void)
{
    int loc_var = 7;
    pid_t pid = fork();
    printf("Loc_var is %d\n", loc_var);
    if (pid == 0)
```

```
     loc_var = 6
}
```

4. โปรแกรมด้านล่างนี้พิมพ์อะไรออกมา เพราะเหตุใด

```
int main(void)
{
    int* loc_var = malloc(sizeof(int)*1);
    *loc_var = 7;
    pid_t pid = fork();
    printf("Loc_var is %d\n", *loc_var);
    if (pid == 0)
        *loc_var = 6
}
```

5. โปรแกรมด้านล่างนี้พิมพ์อะไรออกมา เพราะเหตุใด

```
int main(void)
{
    pid_t pid = fork();
    int exit;
    if (pid != 0) {
        wait(&exit);
    }
    printf("Hello World\n: %d\n", pid);
}
```

6. อะไรจะถูกเก็บไว้ในไฟล์ output.txt อธิบายกลไก file descriptor ในระบบปฏิบัติการ UNIX

```
int main(void)
{
int fd;

    fd = open("output.txt", O_CREAT|O_TRUNC|O_WRONLY, 0666);
    if(!fork()) {
        write(fd, "hello ", 6);
        exit(0);
    } else {
        int status;
        wait(&status);
        write(fd, "world\n", 6);
    }
}
```

7. โปรแกรมด้านล่างนี้พิมพ์อะไรออกมาก อธิบายเหตุผลว่าทำไมจึงเป็นเช่นนั้น แจกแจงว่าทำไมรัน for loop ไม่จบ จากนั้นดัดแปลงโปรแกรมนี้ให้รัน (และพิมพ์ผลลัพธ์) for loop ให้จบ อธิบายการทำงานของโปรแกรมที่ถูกดัดแปลง

```
int main(void)
{
    char** argv = (char**) malloc(3*sizeof(char*));
    argv[0] = "/bin/ls";
    argv[1] = ".";
    argv[2] = NULL;
    for (int i = 0; i < 10; i++) {
        printf("%d\n", i);
        if (i == 3)
     execv("/bin/ls", argv);
    }

}
```

8. บอกความเป็นไปได้ทั้งหมดของผลลัพธ์ที่สามารถเกิดได้จากการรันโปรแกรมต่อไปนี้

```
void* task(void* arg) {
    pid_t pid = getpid();
    printf("My pid is %d\n", pid);
    return NULL;
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, &task, NULL);
    task(NULL);
    return 0;
}
```

9. บอกความเป็นไปได้ทั้งหมดของผลลัพธ์ที่สามารถเกิดได้จากการรันโปรแกรมต่อไปนี้

```
void *worker(void *arg) {
    printf("Worker\n");
    return NULL;
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, &worker, NULL);
    pthread_yield();
    printf("Main\n");
    return 0;
```

```
}
```

10. ผลลัพธ์จากการรันโปรแกรมด้านล่างนี้คืออะไร เพราะเหตุใด

```
void *worker(void *arg) {
    int *loc_var = (int*) arg;
    *loc_var = 2;
    return NULL;
}

int main() {
    int i = 0;
    pthread_t thread;
    pthread_create(&thread, NULL, &worker, &i);
    pthread_join(thread, NULL);
    printf("i is %d\n", i);
    return 0;
}
```

11. ผลลัพธ์จากการรันโปรแกรมด้านล่างนี้คืออะไร เพราะเหตุใด

```
void *worker(void *arg) {
    char *message = (char *) arg;
    strcpy(message, "I am the child");
    return NULL;
}

int main() {
    char *message = malloc(100);
    strcpy(message, "I am the parent");
    pthread_t thread;
    pthread_create(&thread, NULL, &worker, message);
    pthread_join(thread, NULL);
    printf("%s\n", message);
    return 0;
}
```

12. บอกความเป็นไปได้ทั้งหมดของผลลัพธ์ที่ได้จากการรันโปรแกรมต่อไปนี้

```
void *worker(void *arg) {
    int *my_var = (int *) arg;
    *my_var = *my_var + 1;
    printf("My_var is %d\n", *my_var);
    return (void *) 42;
}
```

```
int my_var;
int main() {
    int status;
    my_var = 0;
    pthread_t thread;
    pid_t pid = fork();
    if (pid == 0) {
        pthread_create(&thread, NULL, &worker, &my_var);
        pthread_join(thread, NULL);
    } else {
        pthread_create(&thread, NULL, &worker, &my_var);
        pthread_join(thread, NULL);
        pthread_create(&thread, NULL, &worker, &my_var);
        pthread_join(thread, NULL);
        wait(&status);
    }
    return 0;
}
```

สิ่งที่ต้องส่งในชั่วโมง

- แบบฝึกหัดที่ทำในชั้นเรียน

สิ่งที่ต้องสิ่งหลังชั่วโมงปฏิบัติการ

- ไฟล์ lab6_answer.pdf ที่ใส่คำตอบของคำถามในปฏิบัติการนี้
- ไฟล์ README เพื่อระบุสถานะว่าทำเสร็จถึงส่วนใด มีข้อผิดพลาดอะไรบ้าง กรณีที่ทำไม่เสร็จให้ชี้แจงเหตุผลด้วย

การส่งงาน:

- **นำงานที่ต้องส่งหลังจากชั่วโมงปฏิบัติการใส่ไว้ที่โฟลเดอร์ studentID1_firstname1_studentID2_firstname2_lab6 โดย studentID และ firstname คือเลขประจำตัวและชื่อแรกของสมาชิกที่ทำปฏิบัติการร่วมกัน จากนั้น zip โฟลเดอร์นี้แล้วส่ง zip ไฟล์มาที่ Google Classroom ก่อนกำหนดส่ง**