

ใบงาน 4

วัตถุประสงค์ ศึกษา system call fork() และอื่นๆ ที่เกี่ยวข้อง

4.1 fork()

System call คือ library สำหรับ
นักพัฒนาโปรแกรมเรียกให้ kernel
ให้บริการตามความสามารถของ
system call นั้นๆ

fork() เป็นคำสั่งสร้างโปรเซส โปรเซสในระบบคอมพิวเตอร์มักมีความสัมพันธ์แม่ลูก สำหรับ โปรเซสที่สร้าง และ โปรเซสที่ถูกสร้าง

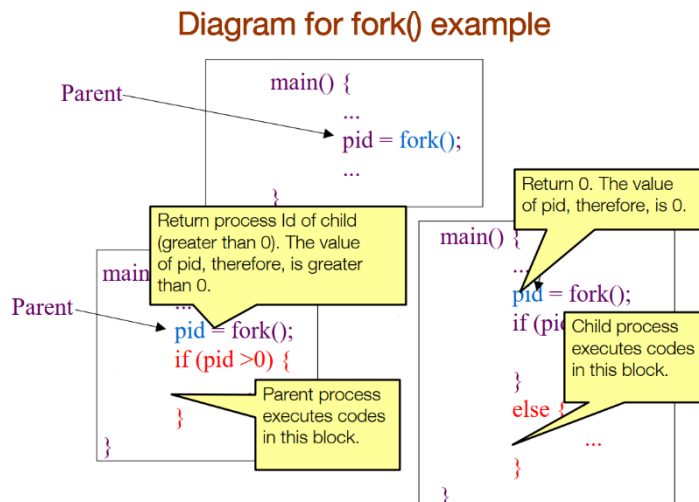
การเรียก fork() ต้องใช้ unistd.h และ sys/types.h

fork() ทำสำเนาของ code ของแม่ และคืน process id ของลูกที่สร้างมาให้แม่ ทั้ง 2 โปรเซสจะทำงานต่อใน statement ถัดไป การที่ copy ของลูกไม่ได้ fork() ทำให้ pid ใน copy ของลูกเป็น 0 เราจึงใช้ค่านี้เป็นตัวแยก

code ของแม่ และ ลูก หลังจากการ fork()

เนื่องจากการเลือกโปรเซสเข้าไปครอบครองซีพียูเป็น
หน้าที่ของโอเอส กล่าวคือลำดับการทำงานระหว่างโพรเซสแม่ กับ โปรเซสลูกไม่จำเป็นต้องแม่ได้ครอบครองซีพียูก่อนลูกเสมอ

อนึ่ง ชื่อ pid ถือว่ากำกวม เพราะ fork() คืน process id ของลูก ดังนั้น pid ไม่ใช่ pid ของผู้เรียก fork() แต่เป็นของลูกของผู้เรียก



```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4
5
6  int main(/* int argv, char** argc */) {
7      pid_t pid;
8      int i = 0;      //prevent garbage
9
10     pid = fork();
11     if (pid > 0 ) { //parent
12         i = 1;
13         printf("my copy of i is %d\n",i)
14     } else {        //child
15         i = 2;
16         printf("my copy of i is %d\n",i)
17     }
18
19
20     return 0;
21 }

```

Q1 จาก code ที่ให้ เขียน output 3 แบบที่เป็นไป

ได้ (แบบที่ 3 ถือว่าพบได้ไม่บ่อย ค่อยกลับมาหาก็ได้ แบบที่ 4 อยู่หน้าถัดไป)

1. My copy of i is 1
My copy of i is 2

2. My copy of i is 2
My copy of i is 1

3. My copy of i is 1

ภาพประกอบ

output กรณีที่ 4 เกิดจากเสียเวลาที่
prompt กำลังเขียน (prompt สีเขียว)
ออกจอ (โดยมากเป็น) โพรเซสลูกเขียน
ออกจอพอดี cursor จะอยู่หลังสุด ...

กล่าวคือหากไม่รู้เรื่องนี้ก่อนอาจสับสนได้ว่า prompt หายไปไหน หรือ ทำไมการแสดงผลไม่มีการจัดจ้งหะ
(เพราะเราไม่ได้จัดจ้งหะ)

```
mirage@mirage-VirtualBox: ~
mirage@mirage-VirtualBox:~$ gcc -o lab lab4.c
mirage@mirage-VirtualBox:~$ ./lab
=====
my copy of i is 1 <0>
mirage@mirage-VirtualBox:~$ my copy of i is 2 <0>*
```

4.2 fork() and wait()

จึงมี system call wait() เพื่อให้นักพัฒนาโปรแกรมสร้างกลไกเพื่อคุมจ้งหะการทำงานได้ระดับหนึ่ง โดยแม่จะ
เป็นผู้เรียก เมื่อเรียก wait() แล้ว แม่จะรอให้ลูกจบการ
ทำงานจึงจะไปทำงานต่อที่ส่วนของโปรแกรมหลัง wait()

ในการเรียกแต่ละ system call ลองไม่ #include ก่อน
เพื่อศึกษาว่า system call นั้นๆ ต้อง #include อะไรบ้าง
เป็นวิธีที่สะดวกวิธีหนึ่ง

โปรแกรมที่สมบูรณ์ควรตรวจกรณี fork() ไม่สำเร็จ (บรรทัด
ที่ 11) ด้วย

Q2 จาก code ที่ให้ output มีได้ กี่ แบบ อะไรบ้าง

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5
6 int main(void) {
7     pid_t pid;
8     int i;
9
10    pid = fork();
11    //if (pid < 0) exit(-1);
12    if (pid > 0) { //parent
13        i = 1;
14        printf("my copy of i is %d\n",i)
15    } else { //child
16        i = 2;
17        printf("my copy of i is %d\n",i)
18    }
19
20    wait(NULL);
21    return 0;
22 }
```

2 แบบ คือ

1. my copy of i is 1
my copy of i is 2
2. my copy of i is 2
my copy of i is 1

4.3 fork() wait() and exit()

เมื่อใช้ if ในการแยก code ส่วนของแม่กับลูก หากมีโค้ดต่อจาก if ลูกย่อมไปทำด้วย ซึ่งปกติไม่ใช่สิ่งที่นักพัฒนาโปรแกรมต้องการ จึงเรียก exit() เพื่อให้ลูกจบการทำงาน เมื่อทำงานของตนเสร็จ

```
lab4_code > C lab4_31.c
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5
6  int main(void) {
7      pid_t pid;
8      int i; int sum = 3;
9
10     pid = fork();
11     if (pid > 0) { //parent
12         i = 1; sum += i;
13         printf("my copy of i is %d\n", i);
14     } else { //child
15         i = 2; sum += i;
16         printf("my copy of i is %d\n", i);
17     }
18     print("my sum = %d\n", sum);
19     wait(NULL);
20     return 0;
21 }
```

```
ab4_code > C lab4_32.c
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5
6  int main(void) {
7      pid_t pid;
8      int i; int sum = 3;
9      for (i = 0; i < 3; i++) {
10         pid = fork();
11         if (pid == 0) { //child
12             printf("my copy of i is %d\n", i);
13             exit(0);
14             printf("should not be executed\n");
15         }
16     }
17     while (wait(NULL) != -1); //empty loop
18     print("bye from main \n", sum);
19     return 0;
20 }
```

RETURN VALUE top

wait(): on success, returns the process ID of the terminated child; on error, -1 is returned.

waitpid(): on success, returns the process ID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned. On error, -1 is returned.

waitid(): returns 0 on success or if WNOHANG was specified and no child(ren) specified by id has yet changed state; on error, -1 is returned.

ศึกษาว่า child หลุดจาก wait() ได้อย่างไร ที่

<https://man7.org/linux/man-pages/man2/waitpid.2.html>

Q3.1 4_31 บรรทัดที่ 18 ทำงานกี่ครั้ง
2 ครั้ง

Q3.2 4_32 บรรทัดที่ 18 ทำงานกี่ครั้ง
1 ครั้ง

4.4 fork() exit(), waitpid(), WEXITSTATUS(status)

exit() เป็น system call สำหรับจบการทำงาน การใส่ไว้ใน code ลูก จะรับประกันไม่ให้ลูกสามารถออกมาทำงานนอก if ได้

มี system call สำหรับ wait() แบบระบุ pid หรือ รับค่าจากลูกได้ WEXITSTATUS(status)

ตัวอย่าง code นี้ยังแสดงวิธีการรับค่าทางคีย์บอร์ดด้วย getchar() ซึ่งโปรแกรมเมอร์ต้องตัด \n เอง ปัจจุบันใช้ scanf();

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6 #define clear_buffer() while(getchar() != '\n')
7
8 int main() {
9     pid_t pid[3], w;
10    int num, sum = 0; int i, status;
11    printf("Enter a positive number ");
12    num = getchar() - 48; //scanf("%d",&num);
13    clear_buffer();
14    //ignore if (num <= 0)
15    for (i = 0; i < 3; i++) {
16        if ((pid[i] = fork()) == 0) {
17            printf("I am child no %d. my copy of num %d\n", i, num);
18            exit(i);
19        }
20    }
21    for (i = 0; (w = waitpid( pid[i], &status, 0)) && w != -1; ++i) {
22        printf("Wait on PID: %d returns value of : %d\n", w,
23            WEXITSTATUS(status));
24    }
25 }
```

ดูอ่านรายละเอียด คีย์บอร์ดคั่นทุกใน buffer ของการรับค่า(ทิ้งไป)

4.5 getpid(), getppid()

getpid() เป็น system call แสดงค่า pid ของผู้เรียก ผู้เรียกสามารถใช้ค่านี้ตามวัตถุประสงค์ ทำนองเดียวกัน getppid() แสดงค่า pid ของแม่

นอกจากนี้ตัวอย่างนี้แสดงให้เห็นว่า แม้ตัวแปร sum จะเป็น global แม้ลูกก็เห็นเป็นคนละตัว เพราะอยู่กันคนละ code (เพียงแค่ copy มาจึงเหมือนกันเป๊ะ)

```
lab4_code > C lab4_5.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 int sum = 0;
8 int main() {
9     pid_t pid;
10    int i;
11    if (pid = fork()) > 0) {
12        i = 1;
13        sum += i;
14        printf("I am parent. My id is %d\n", getpid());
15    } else {
16        i = 2;
17        sum += i;
18        printf("I am child. My id is %d ", getpid());
19        printf("My parent id is %d\n", getppid());
20        exit(0);
21    }
22    wait(NULL);
23    printf("(parent) sum = %d\n", sum);
24    return 0;
25 }
```

อนึ่ง zombie process หมายถึง โพรเซสลูกที่จบการ

ทำงานไปแล้ว แต่ยังครองทรัพยากร

ไว้ (แม้ไม่รู้วาลูกจบไปแล้ว)

สมัยก่อนโอเอสอาจพลาดในการ

จัดการ สามารถพบโพรเซสสถานะนี้

<defunct> ได้

```
pid_t pid;
int i;
pid = fork();
if (pid > 0) { /* parent */
    i = 1;
    sleep(5);
    printf("I am parent my pid = %d, i = %d\n", getpid(), i);
}
else { /* child */
    i = 2;
    printf("I am child my pid = %d, i = %d\n", getpid(), i);
}
```

```
suntana@DESKTOP-IQOCR48:~/lab3$ ps -u suntana
PID TTY      TIME CMD
26 tty1    00:00:00 bash
49 tty2    00:00:00 bash
70 tty2    00:00:00 ps
suntana@DESKTOP-IQOCR48:~/lab3$ ps -u suntana
PID TTY      TIME CMD
26 tty1    00:00:00 bash
49 tty2    00:00:00 bash
71 tty1    00:00:00 demo
72 tty1    00:00:00 demo <defunct>
73 tty2    00:00:00 ps
suntana@DESKTOP-IQOCR48:~/lab3$ ps -u suntana
PID TTY      TIME CMD
26 tty1    00:00:00 bash
49 tty2    00:00:00 bash
74 tty2    00:00:00 ps
```

ศัพท์อีกคำหนึ่งคือ orphan process หมายถึงโพรเซสที่แม่จบการทำงานไปแล้ว โอเอสจะหาโพรเซสเหนือแม่มา

จัดการโพรเซสลูกเมื่อลูกทำงานเสร็จ แต่มีกรณีที่น่าสนใจให้ลูกเป็น daemon process คอยให้บริการ เช่น httpd

4.6 จากตัวอย่างโปรแกรม

โปรแกรมรับค่าเต็มบวกจากผู้ใช้

Process แม่ คำนวณผลบวกจาก 1 ถึง จำนวนนั้น

Process ลูก คำนวณผลบวกจาก 1 ถึง 2 เท่าของ
จำนวนนั้น

ให้พิมพ์ผลลัพธ์จาก child ก่อน

Q6 ถ้าย้ายบรรทัดที่ 19-20 ออกมาหลัง if (pid = fork() > 0) ได้ผลเหมือนกันหรือไม่

เหมือนกัน

4.7 โปรแกรม Lab4_7_skel.c

A ให้โปรเซสแม่วนลูปโดยใช้ for loop เพื่อ fork
โปรเซสลูก 5 โปรเซส

B โดยลูกคนที่ถูก fork จากค่า index ของลูปที่
เป็นเลขคู่ (0 2 4) จะวนลูป fork ลูกของตัวเอง 3
โปรเซส ส่วนลูกที่เกิดจาก index ที่เป็นเลขคี่ (1 3
) จะวนลูป fork ลูกของตัวเอง 4 โปรเซส

C โปรเซสแต่ละตัวจะทำงานโดยพิมพ์ข้อความ
แสดงตัวออกมาหนึ่งบรรทัดเท่านั้น (ภาพประกอบ
ลูกเลขคู่ fork 2 โปรเซส และ ลูกเลขคี่ fork 3 โปร
เซส)

Q7.9 ทำไมตรงนี้ไม่ต้องมี exit(0)

Q7.10 โปรแกรมนี้จะแสดงผลกี่บรรทัด

Q7.11 โปรแกรมนี้จะสร้าง process ทั้งหมดกี่ตัว

Q7.12 ตอบ yes หรือ no ว่า ลำดับการแสดงผล
เสียงที่เกิดขึ้น เหมือนกันทุกรอบหรือไม่ (ศึกษา
ว่าเพราะเหตุใด)

```

7 int main() {
8     pid_t pid;
9     int i = 0;
10    printf("Enter a positive number : ");
11    scanf("%d", &num); //num = getchar() - 48; while(getchar() != '\n');
12    if (num <= 0) {
13        print("You did not enter a positive number\n");
14        exit(1); //exit(-1)
15    }
16    if (pid = fork() > 0) { //parent
17        for (i = 1; i <= num; i++)
18            sum += i;
19        wait(NULL);
20        printf("I am parent my sum = %d\n", sum);
21    } else {
22        for (i = 1; i <= 2 * num; i++)
23            sum += i;
24        printf("I am child my sum = %d\n", sum);
25        exit(0);
26    }
27    return 0;
28 }

```

```

7 int main() {
8     pid_t pid, pidsub;
9     int i, j;
10    printf("only parent befor fork\n");
11    for (i = 0; i < 5; i++) {
12        pid = fork();
13        if (pid == 0) {
14            if ((i % 2) == 0) {
15                printf("I am the child no %d\n", i);
16                int num_gc = ; /* 7.1 */
17                for (j = 0; j < num_gc; j++) {
18                    /* 7.2 */
19                    if (pidsub == 0) {
20                        printf("I am grandchild num %d of even child no %d\n", j, i);
21                        /* 7.3 */
22                    }
23                } //for j
24                /* 7.4 */
25                exit(0);
26            } /*even child */ else { //odd child {
27                printf("I am the child no %d\n", i);
28                int num_gc = ; /* 7.5 */
29                for (j = 0; j < num_gc; j++) {
30                    pidsub = fork();
31                    if ( /* 7.6 */ ) {
32                        printf("I am grandchild num %d of even child no %d\n", j, i);
33                        /* 7.7 */
34                    }
35                } //for j
36                while(wait(NULL) != -1);
37                /* 7.8 */
38            }
39            // exit(0) /* 7.9 */
40        } //if child
41    } //i
42    while (wait(NULL) != -1);
43    return 0;
44 }

```

```

child 0 forked grandchild 1
child 1 forked grandchild 2
child 2 forked grandchild 1
child 0 forked grandchild 0
child 2 forked grandchild 0
child 1 forked grandchild 1
child 1 forked grandchild 3
child 1 forked grandchild 0
parent is terminating! Bye

```

ตอบคำถาม 12 ข้อ

พิมพ์คำตอบในไฟล์ Lab4_xxyyy.txt

กำหนดส่ง TBA

7.9 มันไป exit in if ไม่ก็ else condition อยู่แล้ว

7.10 23 lines

7.11 22

7.12 no