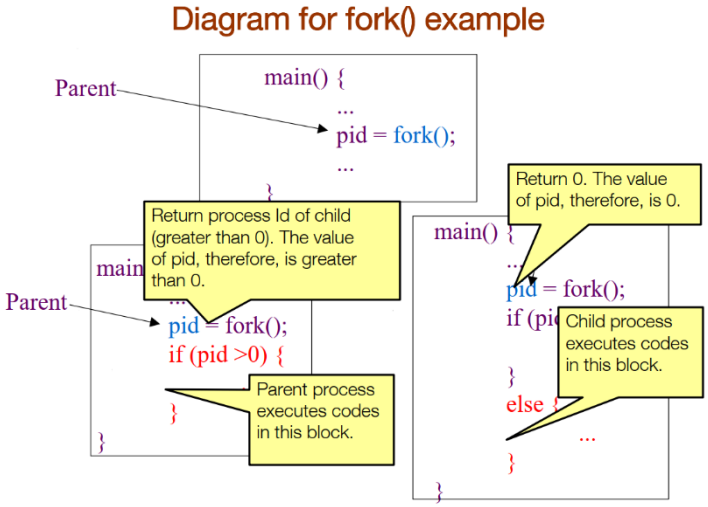
ชื่อ ณัฐพัชร์ สงศรีอินทร์ รหัสนักศึกษา 65050301

ใบงาน 4

วัตถุประสงค์ ศึกษา system call fork() และ อื่นๆ ที่เกี่ยวข้อง

4.1 fork()

System call คือ library สำหรับนักพัฒนาโปรแกรมเรียกให้ kernel ให้บริการตามความสามารถของ system call นั้นๆ



fork() เป็นคำสั่งสร้างโพรเซส โพรเซสในระบบคอมพิวเตอร์มักมีความสัมพันธ์แม่ลูก สำหรับ โพรเซสที่สร้าง และ โพรเซสที่ถูกสร้าง

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

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

เนื่องจากการเลือกโพรเซสเข้าไปครอบครองซีพียูเป็นหน้าที่ของโอเอส กล่าวคือลำดับการทำงานระหว่างโพรเซสแม่ กับ โพรเซสลูกไม่จำเป็นต้องแม่ได้ครอบครองซีพียูก่อนลูกเสมอ

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

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>


int main(/* int argv, char** argc */) {
    pid_t pid;
    int i = 0;      //prevent garbage

    pid = fork();
    if (pid > 0 ) { //parent
        i = 1;
        printf("my copy of i is %d\n",i)
    } else {        //child
        i = 2;
        printf("my copy of i is %d\n",i)
    }


    return 0;
}
```

Q1 จาก code ที่ให้ เขียน output 3 แบบที่เป็นไปได้ (แบบที่ 3 ถือว่าพบได้ไม่ง่าย ค่อยกลับมาหาก็ได้ แบบที่ 4 อยู่หน้าถัดไป)

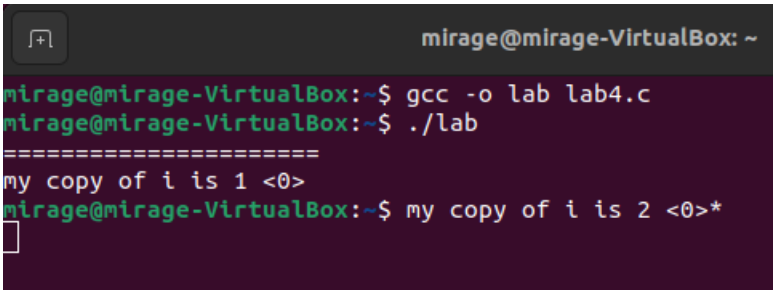
1. My copy of i is 1
My copy of i is 2

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

3. My copy of i is 1

ชื่อ ..... รหัสนักศึกษา .....

ภาพประกอบ



output กรณีที่ 4 เกิดจากเสี้ยวเวลาที่ prompt กำลังเขียน (prompt สีเขียว) ออกจอ (โดยมากเป็น)โพรเซสลูกเขียนออกจอพอดี cursor จะอยู่หลังสุด ... กล่าวคือหากไม่รู้เรื่องนี้ก่อนอาจสับสนได้ว่า prompt หายไปไหน หรือ ทำไมการแสดงผลไม่มีการจัดจังหวะ (เพราะเราไม่ได้จัดจังหวะ)

4.2 fork() and wait()

จึงมี system call wait() เพื่อให้นักพัฒนาโปรแกรมสร้างกลไกเพื่อคุมจังหวะการทำงานได้ระดับหนึ่ง โดยแม่จะเป็นผู้เรียก เมื่อเรียก wait() แล้ว แม่จะรอให้ลูกจบการทำงานจึงจะไปทำงานต่อที่ส่วนของโปรแกรมหลัง wait()

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    int i;

    pid = fork();
    //if (pid < 0) exit(-1);
    if (pid > 0 ) { //parent
        i = 1;
        printf("my copy of i is %d\n",i)
    } else {        //child
        i = 2;
        printf("my copy of i is %d\n",i)
    }

    wait(NULL);
    return 0;
}
```

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

โปรแกรมที่สมบูรณ์ควรตรวจกรณี fork() ไม่สำเร็จ (บรรทัดที่ 11) ด้วย

Q2 จาก code ที่ให้ output มีได้ กี่ แบบ อะไรบ้าง

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

ชื่อ ..... รหัสนักศึกษา .....

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

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    int i; int sum = 3;

    pid = fork();
    if (pid > 0 ) { //parent
        i = 1;  sum += i;
        printf("my copy of i is %d\n",i)
    } else {        //child
        i = 2;  sum += i;
        printf("my copy of i is %d\n",i)
    }
    print("my sum = %d\n",sum)
    wait(NULL);
    return 0;
}
```

ab4_code > C lab4_32.c

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    int i; int sum = 3;
    for (i = 0; i < 3; i++) {
        pid = fork();
        if (pid == 0 ) { //child
            printf("my copy of i is %d\n",i)
            exit(0);
            printf("should not be executed\n")
        }
    }
    while (wait(NULL) != -1) ;      //empty loop
    print("bye from main \n",sum)
    return 0;
}
```

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

ชื่อ ..... รหัสนักศึกษา .....

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();

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#define clear_buffer() while(getchar() != '\n')

int main() {
    pid_t pid[3], w;
    int num, sum = 0, int i, status;
    printf("Enter a positive number ");
    num = getchar() - 48; //scanf(%d,&num);
    clear_buffer();
    //ignore if (num <= 0)
    for (i = 0; i < 3; i++) {
        if ((pid[i] = fork()) == 0) {
            printf("I am child no %d. my copy of num %d\n", i, num);
            exit(i)
        }
    }
    for (i = 0; (w = waitpid( pid[i], &status, 0)) && w != -1; ++i) {
            printf("Wait on PID: %d returns value of  : %d\n", w,
                WEXITSTATUS(status));
    }
}
```

ลูปอ่านขยะหลัง คีย์บอร์ดอินพุต ใน 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 แม่ลูกก็เห็นเป็นคนละตัว เพราะอยู่กันคนละ code (เพียงแค่ copy มาจึงเหมือนกันเป๊ะ)

lab4_code > C lab4_5.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int sum = 0;
int main() {
    pid_t pid;
    int i;
    if ((pid = fork()) > 0) {
        i = 1;
        sum += i;
        printf("I am parent. My id is %d\n", getpid());
    } else {
        i = 2;
        sum += i;
        printf("I am child. My id is %d ", getpid());
        printf("My parent id is %d\n", getppid());
        exit(0);
    }
    wait(NULL);
    printf("(parent) sum = %d\n", sum);
    return 0;
}
```

อนึ่ง zombie process หมายถึง โพรเซสลูกที่จบการทำงานไปแล้ว แต่ยังครองทรัพยากรไว้ (แม่ไม่รู้ว่าลูกจบไปแล้ว) สมัยก่อนโอเอสอาจพลาดในการจัดการ สามารถพบโพรเซสสถานนี้ <defunct> ได้

```c
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 โอเอสจะหาโพรเซสเหนือแม่มาจัดการโพรเซสลูกเมื่อลูกทำงานเสร็จ แต่มีกรณีที่แม่จงใจให้ลูกเป็น daemon process คอยให้บริการ เช่น httpd

ชื่อ ..... รหัสนักศึกษา .....

4.6 จากตัวอย่างโปรแกรม

โปรแกรมรับค่าเต็มบวกจากผู้ใช้

Process แม่ คำนวณผลบวกจาก 1 ถึง จำนวนนั้น

Process ลูก คำนวณผลบวกจาก 1 ถึง 2 เท่าของจำนวนนั้น

ให้พิมพ์ผลลัพธ์จาก child ก่อน

```c
int main() {
    pid_t pid;
    int i = 0;
    printf("Enter a positive number : ");
    scanf(" %d", &num); //num = getchar() - 48; while(getchar() != '\n');
    if (num <= 0) {
        print("You did not enter a positive number\n");
        exit(1); //exit(-1)
    }
    if (pid = fork() > 0) { //parent
        for (i = 1; i <= num; i++)
            sum += i;
        wait(NULL);
        printf("I am parent my sum = %d\n", sum);
    } else {
        for (i = 1; i <= 2 * num; i++)
            sum += i;
        printf("I am child my sum = %d\n", sum);
        exit(0);
    }
    return 0;
}
```

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

เหมือนกัน

4.7 โปรแกรม Lab4_7_skel.c

A ให้โพรเซสแม่วนลูปโดยใช้ for loop เพื่อ fork โพรเซสลูก 5 โพรเซส

B โดยลูกคนที่ถูก fork จากค่า index ของลูปที่เป็นเลขคู่ (0 2 4) จะวนลูป fork ลูกของตัวเอง 3 โพรเซส ส่วนลูกที่เกิดจาก index ที่เป็นเลขคี่ (1 3) จะวนลูป fork ลูกของตัวเอง 4 โพรเซส

C โพรเซสแต่ละตัวจะทำงานโดยพิมพ์ข้อความแสดงตัวออกมาหนึ่งบรรทัดเท่านั้น (ภาพประกอบลูกเลขคู่ fork 2 โพรเซส และ ลูกเลขคี่ fork 3 โพรเซส)

```c
int main() {
    pid_t pid, pidsub;
    int i,j;
    printf("only parent befor fork\n");
    for (i = 0; i < 5; i++) {
        pid = fork();
        if (pid == 0) {
            if ((i % 2) == 0) {
                printf("I am the child no %d\n", i);
                int num_gc =    ;       /* 7.1 */
                for (j = 0; j < num_gc; j++) {
                                        /* 7.2 */
                    if (pidsub == 0) {
                        printf("I am grandchild num %d of even child no %d\n",j,i);
                                        /* 7.3 */
                    }
                } //for j
                                        /* 7.4 */
                exit(0);
            } /*even child */ else { //odd chiled {
                printf("I am the child no %d\n", i);
                int num_gc =    ;       /* 7.5 */
                for (j = 0; j < num_gc; j++) {
                    pidsub = fork();
                    if (                /* 7.6 */ ) {
                        printf("I am grandchild num %d of even child no %d\n",j,i);
                                        /* 7.7 */
                    }
                } //for j
                while(wait(NULL) != -1);
                //                      /* 7.8 */
            }
            // exit(0)                  /* 7.9 */
        } //if child
    } //i
    while (wait(NULL) != -1) ;
    return 0;
}
```

```
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

Q7.9 ทำไมตรงนี้ไม่ต้องมี **exit(0)**

Q7.10 โปรแกรมนี้จะแสดงผลกี่บรรทัด

Q7.11 โปรแกรมนี้จะสร้าง process ทั้งหมดกี่ตัว

Q7.12 ตอบ yes หรือ no ว่า ลำดับการแสดงผลเรียงที่เกิดนั้น เหมือนกันทุกรอบหรือไม่ (ศึกษาว่าเพราะเหตุใด)

พิมพ์คำตอบในไฟล์ Lab4_xxyyy.txt

กำหนดส่ง TBA