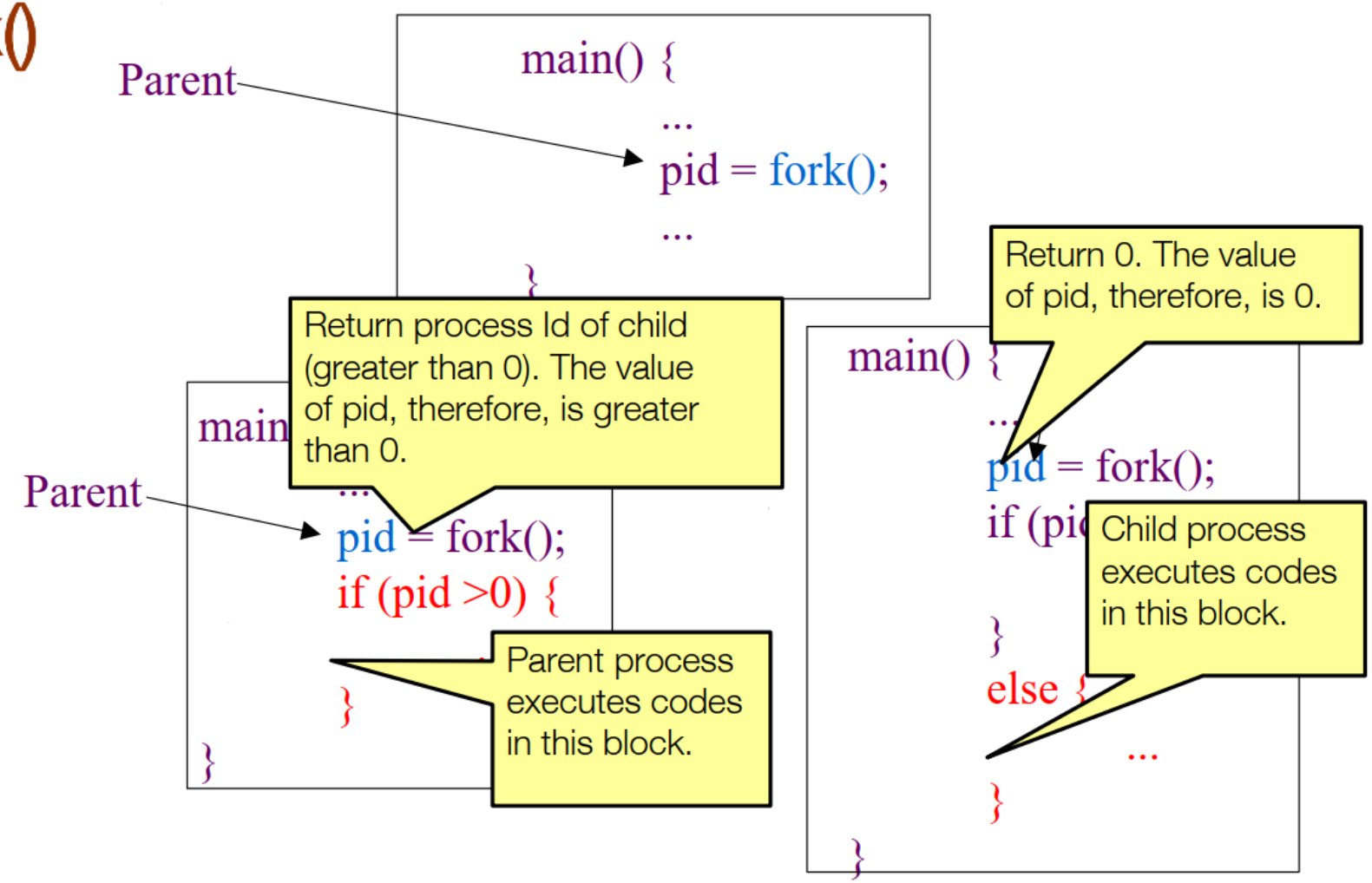
fork()

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

Creates a child process by making a copy of the parent process

Diagram for fork() example



4.1 fork()

- output 3 แบบ ได้แก่อะไรบ้าง

```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;
}
```

4.2 fork() wait()

- output มีได้ กี่แบบ อะไรบ้าง

- ศึกษาว่า child หลุดจาก wait() ได้อย่างไร ที่ https://man7.org/linux/man-pages/man2/waitpid.2.html

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.

```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 ) { //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;
}
```

4.3 fork(), wait(), exit()

- 4_31 บรรทัดที่ 18 ทำงานกี่ครั้ง
- 4_32 บรรทัดที่ 18 ทำงานกี่ครั้ง

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;
}
```

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

```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 คีย์บอร์ด อินพุต ใน **buffer** ของการรับค่า(ทิ้งไป)

4.5 getpid(), getppid()

- บรรทัดที่ sum ที่ บรรทัดที่ 23 มีค่าเท่าไหร่

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;
}
```

4.6 zombie process (defunct process)

Zombie Process

A **Zombie process** (or *defunct process*) is a process that has completed execution but hasn't been reaped by its **parent process**. As result it holds a **process entry** in the **process table** and the **PID**.

Orphan Process

> An orphan process is a computer **process whose parent process has finished or terminated**, though it remains running itself.

A process can be *orphaned unintentionally*, such as when the parent process terminates or crashes. The process group mechanism in most Unix-like operation systems can be used to help protect against accidental orphaning, where in coordination with the user's shell will try to terminate all the child processes with the `SIGHUP` process signal, rather than letting them continue to run as orphans.

A process may also be *intentionally orphaned* so that it becomes detached from the user's session and left running in the background; usually to allow a long-running job to complete without further user attention, or to start an indefinitely running service. Under Unix, the latter kinds of processes are typically called daemon processes. The Unix `nohup` command is one means to accomplish this.

https://www.gmarik.info/blog/2012/orphan-vs-zombie-vs-daemon-processes/

https://www.gmarik.info/blog/2012/orphan-vs-zombie-vs-daemon-processes/

```c
/* zombie.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
  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);
  }
  return 0;
}
```

```
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
```

4.7

- โจทย์

- โปรแกรมค่าเต็มบวกจากผู้ใช้
- Process แม่ คำนวณผลบวกจาก 1 ถึง จำนวนนั้น
- Process ลูก คำนวณผลบวกจาก 1 ถึง 2 เท่า ของจำนวนนั้น
- ให้พิมพ์ผลลัพธ์จาก child ก่อน

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

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <sys/types.h>
4   #include <unistd.h>
5   #include <sys/wait.h>
6
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      }
26      return 0;
27  }
```

task

```c
#include <stdio.h>
#include <sys/types.h>  //fork()
#include <unistd.h>     //fork()
#include <sys/wait.h>   //wait()
#include <stdlib.h>     //exit()

int main() {
    pid_t pid, pidsub;
    int i,j;
```

- เติมโปรแกรม Lab4_8_skel.c ตามข้อกำหนด และ**ตอบคำถาม**
- 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 (ภาพประกอบ ลูกเลขคู่ fork 2 โพรเซส และลูกเลขคี่ fork 3 โพรเซส)

```c
    printf("only parent before 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 =    ;       /* 8.1 */
                for (j = 0; j < num_gc; j++) {
                                        /* 8.2 */
                    if (pidsub == 0) {
                        printf("I am grandchild num %d of even child no %d\n",j,i);
                                        /* 8.3 */
                    }
                } //for j
                                        /* 8.4 */
                exit(0);
            } /*even child */ else { //odd chiled {
                printf("I am the child no %d\n", i);
                int num_gc =     ;      /* 8.5 */
                for (j = 0; j < num_gc; j++) {
                    pidsub = fork();
                    if (                /* 8.6 */ ) {
                        printf("I am grandchild num %d of even child no %d\n",j,i);
                                        /* 8.7 */
                    }
                } //for j
                while(wait(NULL) != -1);
                exit(0);
            }
            //exit(0);                  /* 8.8 */
        } //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
```

8.8 ทำไมตรงนี้ไม่ต้องมี exit(0)

8.9 โปรแกรมนี้จะแสดงผลกี่บรรทัด

8.10 โปรแกรมนี้จะสร้าง process ทั้งหมดกี่ตัว

8.11 ตอบ yes หรือ no ว่า ลำดับการแสดงผลเรียงที่เกิดนั้น เหมือนกันทุกรอบหรือไม่ (ศึกษาว่าเพราะเหตุใด)

Online gcc editor

- https://www.onlinegdb.com/online_c_compiler
- https://repl.it/languages/c
- https://ideone.com/
- https://www.codechef.com/ide