ใบงานที่ 2

เรื่อง Process management(Fork)

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นายอธิศ สุนทโรดม

รหัส 65543206086-2

ใบงานนี้เป็นส่วนหนึ่งของวิชาระบบปฏิบัติการ (ENGCE125)

หลักสูตรวิศกรรมมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

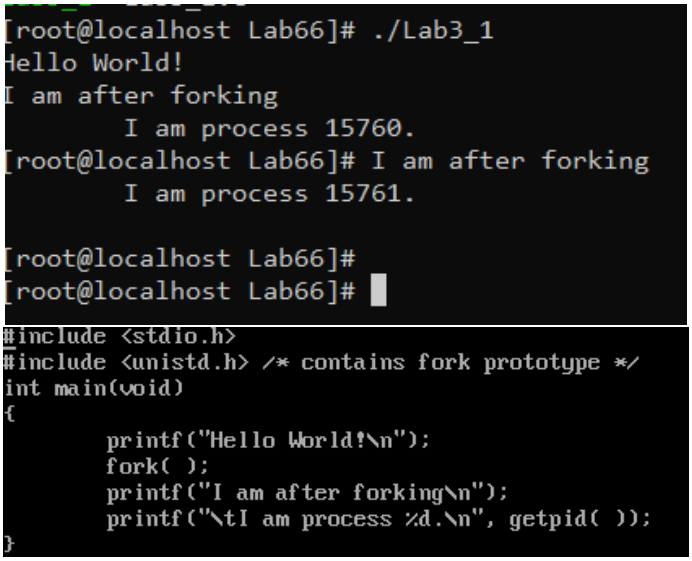
มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา เชียงใหม่

ประจำภาคเรียนที่ 2 ปีการศึกษา 2566

การเขียนใบงานการทดลอง

1. เขียนอธิบายขั้นตอนการทำงานโปรแกรมพร้อมแคปเจอร์รูปภาพขั้นตอนการทำงานสรุปผลการทำงานในแต่ละหัวข้อของการทดลอง
2. ทำเป็นเอกสารใบงานบันทึกงานเป็นไฟล์ Wordและปริ้นส่ง
3. จากการทดลองมีข้อแตกต่างกันอย่างไรบ้างจงอธิบาย
4. สรุปผลการทดลอง

ตัวอย่างที่ 1



```
[root@localhost Lab66]# ./Lab3_1
Hello World!
I am after forking
        I am process 15760.
[root@localhost Lab66]# I am after forking
        I am process 15761.

[root@localhost Lab66]#
[root@localhost Lab66]#
```

```
#include <stdio.h>
#include <unistd.h> /* contains fork prototype */
int main(void)
{
        printf("Hello World!\n");
        fork( );
        printf("I am after forking\n");
        printf("\tI am process %d.\n", getpid( ));
}
```

โค้ดในภาพเป็นโปรแกรมที่แสดงข้อความ "Hello World!" โดยใช้ฟังก์ชัน fork() ของระบบปฏิบัติการ ฟังก์ชัน fork() จะสร้างกระบวนการใหม่ขึ้นมาจากกระบวนการปัจจุบัน Processใหม่นี้จะเรียกว่า " Process ลูก" และกระบวนการปัจจุบันจะเรียกว่า " Process พ่อ" โดย Process ทั้ง 2 จะมีคุณสมบัติเหมือนกันแต่จะแตกต่างกันที่หมายเลข id โดยแสดงออกมากโดย getpid()

ตัวอย่างที่ 2

```
[root@localhost Lab66]# ./Lab3_2
Hello World!
I am the parent process and pid is : 15817 .
Here i am before use of forking
Here I am just after forking
I am the parent process and pid is: 15817 .
[root@localhost Lab66]# Here I am just after forking
I am the child process and pid is :15818.
```

```
#include <stdio.h>
#include <unistd.h> /* contains fork prototype */
int main(void)
{
int pid;
        printf("Hello World!\n");
        printf("I am the parent process and pid is : %d .\n",getpid());
        printf("Here i am before use of forking\n");
        pid = fork();
        printf("Here I am just after forking\n");
        if (pid == 0)
                printf("I am the child process and pid is :%d.\n",getpid());
        else
                printf("I am the parent process and pid is: %d .\n",getpid());
}
```

โค้ดในภาพนี้แสดงถึงการสร้างforkของ parent procees และ child process โดยเริ่มต้นจะทำการพิม process id ของ parent จากนั้นทำการบอกผู้ใช้ว่ากำลังจะทำการสร้าง process ของ child ต่อจากนี้จากนั้นทำการสร้าง fork() โดยเก็บค่าไว้ใน pid จากนั้นเช็คค่าของ pid เมื่อมีค่าเท่ากับ 0 ให้แสดง process id ของ child

ตัวอย่างที่ 3

```
[root@localhost Lab66]# ./Lab3_3
Here I am just before first forking statement
Here I am just after first forking statement
Here I am just after second forking statement
                Hello World from process 15848!
[root@localhost Lab66]# Here I am just after first forking statement
Here I am just after second forking statement
                Hello World from process 15849!
Here I am just after second forking statement
                Hello World from process 15850!
Here I am just after second forking statement
                Hello World from process 15851!
```

```
#include <stdio.h>
#include <unistd.h> /* contains fork prototype */
main(void)
{
        printf("Here I am just before first forking statement\n");
        fork();
        printf("Here I am just after first forking statement\n");
        fork();
        printf("Here I am just after second forking statement\n");
        printf("\t\tHello World from process %d!\n", getpid());
}
```

โค้ดในภาพนี้แสดงถึงการสร้าง fork หลาย process จาก process หลักโดยจะทำงานตาม fork ที่ถูกสร้างขึ้นโดยทั้งหมดมีการสร้าง 2 process โดยและมีการบอกผู้ใช้ถึงการสร้าง process ลูก โดยเมื่อมีการรัน Hello World จะมีการบอก id ของ process นั้นๆเพื่อระบุว่าและแยกแยะความแตกต่างระหว่าง process

ตัวอย่างที่ 4

```
[root@localhost Lab66]# ./Lab3_4
Hello World!
I am the child process.
I am the parent process.
```

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h> /* contains prototype for wait */
int main(void)
{
        int pid;
        int status;
        printf("Hello World!\n");
        pid = fork();
        if(pid == -1) /* check for error in fork */
        {
                perror("bad fork");
                exit(1);
        }
        if(pid == 0)
                printf("I am the child process.\n");
        else
        {
                wait(&status); /* parent waits for child to finish */
                printf("I am the parent process.\n");
        }
}
```

โค้ดในภาพนี้แสดงถึงการตอบโต้ระหว่าง process พ่อ และ ลูก โดยจะมีการเก็บค่าของ pid และ status ไว้จากนั้นทำการสร้าง process ลูกและทำการตรวจสอบโดย if (pid == -1) ตรวจสอบว่าfork()เกิดข้อผิดพลาดหรือไม่ หากเกิดข้อผิดพลาด ให้พิมพ์ข้อความแสดงข้อผิดพลาด "bad fork" และ exit(1) เพื่อยุติโปรแกรมหากว่าไม่เกิดข้อผิดพลาดจะทำการเช็กว่าเป็น process ลูกหรือไม่โดยถ้า pid == 0 คือเมื่อส่งค่า 0 จะแสดงว่าเป็น process ลูก แต่ถ้าไม่จะเรียกใช้waitฟังก์ชันเพื่อรอให้กระบวนการลูกเสร็จสิ้นและ&statusอาร์กิวเมนต์ได้รับสถานะการออกของกระบวนการลูกจากนั้นให้แสดงว่าเป็น parent process

ตัวอย่างที่ 5

```
[root@localhost Lab66]# ./Lab3_5
1170: I am the parent. Remember my number!
1170: I am now going to fork ...
1170: My child's pid is 1171
1170: like father like son.
[root@localhost Lab66]# 1171: Hi! I am the child.
1171: like father like son.
```

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
main()
{
        int forkresult;
        printf("%d: I am the parent. Remember my number!\n", getpid());
        printf("%d: I am now going to fork ... \n", getpid());
        forkresult = fork();
        if (forkresult != 0)
        { /* the parent will execute this code */
                printf("%d: My child's pid is %d\n", getpid(), forkresult);
        }
        else /* forkresult == 0 */
        { /* the child will execute this code */
                printf("%d: Hi! I am the child.\n", getpid());
        }
        printf("%d: like father like son. \n", getpid());
}
```

โค้ดในภาพนี้แสดงถึงการแยกกันทำงานของ process ลูกโดยจะเก็บค่าไว้ใน forkresult และเมื่อ forkresult != 0 จะเป็นการทำงานของ process พ่อ โดยจะแสดง process id ของ Child แต่เมื่อไม่ตรงเงื่อนไขนี้ หรือ forkresult = 0 จะทำงานอีกเงื่อนไขโดยพิมพ์ข้อความที่ระบุถึงการดำเนินการของ process ลูก

ตัวอย่างที่ 6

```
[root@localhost Lab66]# ./Lab3_6
I'am the original process with PID 1178 and PPID 1142.
I'am the parent with PID 1178 and PPID 1142.
My child's PID is 1179
PID 1178 terminates.
```

```
#include <stdio.h>
main()
{
        int pid ;
        printf("I'am the original process with PID %d and PPID %d.\n", getpid(),
        getppid()) ;
        pid = fork ( ) ; /* Duplicate. Child and parent continue from here */
        if ( pid != 0 ) /* pid is non-zero,so I must be the parent*/
        {
                printf("I'am the parent with PID %d and PPID %d.\n", getpid(), getppid()) ;
                printf("My child's PID is %d\n", pid ) ;
        }
        else /* pid is zero, so I must be the child */
        {
                sleep(4); /* make sure that the parent terminates first */
                printf("I'm the child with PID %d and PPID %d.\n", getpid(), getppid()) ;
        }
        printf ("PID %d terminates.\n", getpid()) ;
}
```

โค้ดในภาพนี้แสดงถึง id ของ process เป็นขั้นๆ โดยเมื่อเริ่มต้นจะประกาศตัวแปรจำนวนเต็มpidเพื่อจัดเก็บ ID กระบวนการของกระบวนการลูก และทำการแสดง getpid(): ID กระบวนการปัจจุบันและgetppid():ID กระบวนการหลักของกระบวนการปัจจุบันจากนั้นจะทำการ fork และทำตามเงื่อนไขโดยเมื่อ pid != 0 จะแสดงว่าเป็น parent process และแสดง id ของ parent และ child และถ้าเป็นอีกเงื่อนไขจะแสดงว่าเป็น child process จะรอเป็นเวลา 4 วินาทีเพื่อให้แน่ใจว่ากระบวนการหลักเสร็จสิ้นก่อนและพิมพ์ข้อความ " I'm the child with…”

ตัวอย่างที่ 7

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    int pid;
    pid = fork(); /* Duplicate. Child and parent continue from here */

    if (pid != 0) /* pid is non-zero, so I must be the parent */
    {
        while (1) /* Never terminate and never execute a wait() */
        {
            sleep(10); /* stop executing for 10 seconds */

            // Print the student's name every 10-15 seconds
            printf("Student Name: 65543206086-2 Athit Suntalodom\n");

        }
    }
    else /* pid is zero, so I must be the child */
    {
        exit(42); /* exit with any number */
    }

    return 0;
}
```

```
[root@localhost Lab3]# ./a.out &
[1] 1268
[root@localhost Lab3]# Student Name: 65543206086-2 Athit Suntalodom

[root@localhost Lab3]# Student Name: 65543206086-2 Athit Suntalodom

[root@localhost Lab3]# kill 1268
[root@localhost Lab3]# ps
  PID TTY          TIME CMD
 1107 pts/0    00:00:00 bash
 1270 pts/0    00:00:00 ps
[1]+  Terminated              ./a.out
```

โค้ดในภาพนี้แสดงถึงการทำงานของ process และการทำงานแบบ background process โดยทำการสร้าง fork ออกมาและจากนั้นทำการวนลูปการทำงาน 1 อย่างเพื่อทดสอบการทำงานของ background process จากนั้นทำการ run โดย a.out เพื่อดูการทำงานของ process โดย 1268 เป็น process ที่ทำงานแบบ background process โดยทุก 10 วิจะพิมชื่อออกมาและสามารถทำการสั่งการทำงานอื่นๆได้โดยในที่นี้จะทำการสั่ง kill 1268 เพื่อจะให้หยุดการทำงาน

แบบฝึกหัดที่ 1

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[]) {

        char *who;
        int i;

        if(fork()){
                who = "athit suntalodom.";
        } else{
                who = "child";
        }

        for (i = 0; i<6;i++){
                printf("*frok1: %s\n", who);
        }

        exit (0);

}
```

```
[root@localhost Lab3]# ./ex1
*frok1: athit suntalodom.
*frok1: athit suntalodom.
*frok1: athit suntalodom.
*frok1: athit suntalodom.
*frok1: athit suntalodom.
*frok1: athit suntalodom.
[root@localhost Lab3]# *frok1: child
*frok1: child
*frok1: child
*frok1: child
*frok1: child
*frok1: child
```

โค้ดในภาพนี้แสดงถึงการทำงานของ process โดยจะระบุเป็นชื่อกับ childโดยเมื่อเริ่มต้นจะประกาศตัวแปรเก็บค่าของ int และ char เพื่อใช้ในการแสดงผล เก็บข้อมูล และเช็คเงื่อนไขโดยเมื่อทำการ fork ครั้งแรกจะการรับ who เป็นชื่อและนำมาแสดง 6 รอบโดยจะเป็นการทำงานของ parent process ก่อนและทำการ fork จากนั้นทำการแสดงชื่อของ child เมื่อเราทำการ fork ไปแล้ว 6 รอบในการทำงานยี้เป็นส่วนของ child process

แบบฝึกหัดที่ 2

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

int main (int argc, char *argv[]) {

        int i;
        char *who;
        int n;

        if(fork()){
                who = "athit suntalodom";
                n = 2;
        } else{
                who = "child";
                n = 1;
        }

        for (i = 1; i<=10 ;i++){
                fprintf(stdout,"*%2d. %7s: my pid = %6d, ppid = %6d\n ",i,who,getpid(),getppid());
                fflush (stdout);
                sleep (n);
        }

        exit (0);

}
```

```
[root@localhost Lab3]# ./ex2
* 1. athit suntalodom: my pid =   9250, ppid =   1114
 * 1.    child: my pid =   9251, ppid =   9250
 * 2.    child: my pid =   9251, ppid =   9250
 * 2. athit suntalodom: my pid =   9250, ppid =   1114
 * 3.    child: my pid =   9251, ppid =   9250
 * 4.    child: my pid =   9251, ppid =   9250
 * 3. athit suntalodom: my pid =   9250, ppid =   1114
 * 5.    child: my pid =   9251, ppid =   9250
 * 6.    child: my pid =   9251, ppid =   9250
 * 4. athit suntalodom: my pid =   9250, ppid =   1114
 * 7.    child: my pid =   9251, ppid =   9250
 * 8.    child: my pid =   9251, ppid =   9250
 * 5. athit suntalodom: my pid =   9250, ppid =   1114
 * 9.    child: my pid =   9251, ppid =   9250
 *10.    child: my pid =   9251, ppid =   9250
```

โค้ดในภาพนี้แสดงถึง pid ของ process ทั้งหมดโดยเริ่มจากการกำหนดให้ชื่อเป็น parent process และ child process จากนั้นกำหนดการทำงานโดยจะทำการวนลูป print ชื่อ และ parent id และ ppid โดย child จะ print ออกมาเป็นจำนวน 10 ครั้งทุกๆ 1วิ และชื่อและเป็น 10 ครั้ง ทุกๆ 2 วิ

แบบฝึกหัดที่ 3

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

int main (int argc, char *argv[]) {

        char *who;
        int status;

        if(fork()){
                who = "athit";
                printf("pi =%f\n", 4*atan(1));
                wait (&status);
                exit(0);
        } else{
                who = "child";
                execlp ("/usr/bin/date","date",(char*)0);
        }

}
```

```
[root@localhost Lab3]# ./ex3
pi =3.141593
Tue Dec  5 15:11:28 2023
```

โค้ดในภาพนี้แสดงถึงการทำงานของ child process โดนจะทำการประการ who และทำการ fork จากนั้นให้แสดงค่าของ pi และทำการ wait เพื่อเก็บค่าและรอ child หยุดการทำงานจากนั้นเมื่อทำงานอีกครั้งจะทำงานโดย child จะเข้าไปค้นหาตาม path และ print ค่าออกมาเป็น วัน/เวลา

แบบฝึกหัดที่ 4

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

int main (int argc, char *argv[]) {

        char *who;
        int status;

        if(fork()){
                who = "athit";
                printf("pi =%f\n", 4*atan(1));
                wait (&status);
                exit(0);
        } else{
                who = "child";
                execlp ("/bin/my-script","my-script","a","b",(char*)0);
        }

}
```

```
[root@localhost Lab3]# ./ex4
pi =3.141593
Tue Dec  5 15:11:50 2023
```

โค้ดในภาพนี้แสดงถึงการทำงานของ child process โดยจะประกาศ who เพื่อให้การทำงานของ Processแยกกันและทำการ fork จากนั้นจะแสดง pic และทำการ wait เพื่อเก็บค่าและรอ child หยุดการทำงาน จากนั้นจะทำการเรียกใช้งาน script เพื่อแสดงวัน/เวลา

จากการทดลอง มีข้อแตกต่างกันอย่างไร

โดยการใช้ fork() จะทำในส่วน parent process ส่วนของ child process จะทำงานหลังจากที่สร้างขึ้น parent process ก็จะทำต่อโดยการทำงานและทำตามคำสั่งเหมือนกันโดยอยู่ที่เราจะกำหนดแต่ id ของ process จะไม่เหมือนกัน

สรุปผลการทดลอง

ฟังก์ชัน fork() ในภาษา C เป็นฟังก์ชันพื้นฐานสำหรับการสร้างกระบวนการใหม่ กระบวนการใหม่นี้เรียกว่า (child process) และกระบวนการเดิมเรียกว่า (parent process)

การทำงานของฟังก์ชัน fork() มีดังนี้

ระบบปฏิบัติการจะสร้างกระบวนการใหม่ขึ้นมา โดยกระบวนการใหม่นี้จะมีลักษณะเหมือนกับ parent process ทุกประการ ยกเว้นค่าตัวแปร pid ซึ่งจะแตกต่างกันฟังก์ชัน

fork() จะคืนค่ากลับไปยัง parent process ในกรณีที่การสร้างกระบวนการใหม่สำเร็จ ฟังก์ชัน fork() จะคืนค่าเป็นค่ากระบวนการลูก ในกรณีที่การสร้างกระบวนการใหม่ล้มเหลว ฟังก์ชัน fork() จะคืนค่าเป็นค่า -1 ใน parent process เราสามารถตรวจสอบค่าที่คืนกลับมาจากฟังก์ชัน fork() เพื่อดูว่าการสร้างกระบวนการใหม่สำเร็จหรือไม่ ในกรณีที่การสร้างกระบวนการใหม่สำเร็จ เราสามารถเรียกใช้ฟังก์ชัน wait() เพื่อรอให้กระบวนการลูกทำงานเสร็จก่อนจึงจะดำเนินการต่อได้สำหรับกระบวนการลูก

กระบวนการลูกจะเริ่มต้นการทำงานที่จุดที่ฟังก์ชัน fork() คืนค่ากลับไปยัง parent process กระบวนการลูกสามารถทำงานแยกจาก parent process ได้อย่างอิสระ ตัวอย่างเช่น กระบวนการลูกสามารถทำงานต่าง ๆ ร่วมกัน ทำงานต่างเวลา หรือทำงานต่าง ๆ กันกับ parent process ก็ได้