ใบงานที่ 3

เรื่อง **Process Management (Fork)**

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นาย กวีวัธน์ กาญจน์สุพัฒนากุล 65543206003-7

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

การเขียนใบงานการทดลอง

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

1.ตัวอย่างที่ 1

```c
#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( ));
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex1
Hello World!
I am after forking
        I am process 3107.
I am after forking
        I am process 3108.
```

1.พิมพ์ "Hello World!"

2.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process)

3.พิมพ์ "I am after forking" จากการทำงานหลักและการทำงานการย่อย.

4.แสดง Process ID (PID) ของการทำงานทั้งหลักและย่อย.

2.ตัวอย่างที่ 2

```c
#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());
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex2
Hello World!
I am the parent process and pid is : 3228 .
Here i am before use of forking
Here I am just after forking
I am the parent process and pid is: 3228 .
Here I am just after forking
I am the child process and pid is :3229.
```

1.พิมพ์ "Hello World!"

2.พิมพ์ "I am the parent process and pid is : [PID]." โดยใช้ getpid() เพื่อดึง PID ช

3.พิมพ์ "Here I am before use of forking."

4.ทำการ fork() เพื่อสร้างกระบวนการย่อย.

5.พิมพ์ "Here I am just after forking."

6.ตรวจสอบค่าของ pid ที่ได้จาก fork():

ถ้า pid เป็น 0 แสดงว่าตอนนี้อยู่ในกระบวนการย่อย (child process) จึงพิมพ์ "I am the child process and pid is : [PID]."

ถ้า pid ไม่เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการหลัก (parent process) จึงพิมพ์ "I am the parent process and pid is : [PID]."

ดังนั้น, ผลลัพธ์ชจะได้จากการทำงานของโปรแกรมนี้จะแสดงข้อความช Process ID ของ parent และ child processes หลังจากที่ fork ไป.

3.ตัวอย่างที่ 3

```c
#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());
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex3
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 3334!
Here I am just after first forking statement
Here I am just after second forking statement
                Hello World from process 3335!
Here I am just after second forking statement
                Hello World from process 3336!
Here I am just after second forking statement
                Hello World from process 3338!
```

1.พิมพ์ "Here I am just before the first forking statement."

2.ทำการ fork() ครั้งแรก เพื่อสร้างกระบวนการย่อย.

3.พิมพ์ "Here I am just after the first forking statement."

4.ทำการ fork() ครั้งที่สอง จากทั้ง parent และ child processes ที่ได้จากขั้นที่ 2.

5.พิมพ์ "Here I am just after the second forking statement."

6.ทุกกระบวนการ (ทั้ง parent และ child processes) จะพิมพ์ "Hello World from process [PID]!" โดยที่ [PID] คือ Process ID ของแต่ละกระบวนการ.

4.ตัวอย่างที่ 4

```c
#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");
}
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex4
Hello World!
I am the child process.
I am the parent process. _
```

1.พิมพ์ "Hello World!" จากกระบวนการหลัก.

2.ทำการ fork() เพื่อสร้างกระบวนการย่อย.

3.ตรวจสอบค่าที่ได้จาก fork():

ถ้า pid เป็น -1 แสดงว่ามีข้อผิดพลาดในการ fork จึงพิมพ์ "bad fork" และจบโปรแกรม.

ถ้า pid เป็น 0 จึงพิมพ์ "I am the child process."

ถ้า pid ไม่เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการหลัก จึงทำงานในส่วนของ parent process.

4.ใน parent process, ใช้ wait(&status) เพื่อรอให้ child process ทำงานเสร็จสิ้น.

5.พิมพ์ "I am the parent process." จาก parent process.

6.การใช้ wait(&status) ทำให้ parent process รอ child process ทำงานเสร็จสิ้น และ status จะเก็บสถานะการทำงานของ child process ที่สามารถใช้ในการตรวจสอบว่า child process ทำงานเสร็จสิ้นได้

5.ตัวอย่างที่ 5

```c
#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());
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex5
3696: I am the parent. Remember my number!
3696: I am now going to fork ...
3696: My child's pid is 3697
3696: like father like son.
3697: Hi! I am the child.
3697: like father like son.
```

1.พิมพ์ "I am the parent. Remember my number!" พร้อมกับแสดง PID ของกระบวนการหลัก.

2.พิมพ์ "I am now going to fork ..." พร้อมกับแสดง PID ของกระบวนการหลัก.

3.ทำการ fork() เพื่อสร้างกระบวนการย่อย.

4.ตรวจสอบค่าที่ได้จาก fork():

ถ้า forkresult ไม่เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการหลัก (parent process) จึงพิมพ์ "My child's pid is [child PID]" โดยใช้ค่าที่ได้จาก forkresult.

ถ้า forkresult เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการย่อย (child process) จึงพิมพ์ "Hi! I am the child."

5.พิมพ์ "like father like son." ทั้งใน parent และ child processes.

การทำงานทั้งหมดที่ถูกสร้างขึ้นจาก fork() และสังเกตได้ว่า child process จะมี PID ต่างจาก parent process. นอกจากนี้, สังเกตว่าคำสั่ง "like father like son." จะถูกพิมพ์ทั้งใน parent และ child processes แสดงถึงการทำซ้ำของโปรแกรมทั้งสองกระบวนการ.

6.ตัวอย่างที่ 6

```c
#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()) ;
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex6
I'am the original process with PID 3833 and PPID 2764.
I'am the parent with PID 3833 and PPID 2764.
My child's PID is 3834
PID 3833 terminates.
[iiuuuu@localhost lab3]$ I'm the child with PID 3834 and PPID 1.
PID 3834 terminates.
```

1.พิมพ์ "I'am the original process with PID [PID] and PPID [PPID]." โดยใช้ getpid() และ getppid() เพื่อดึง PID และ PPID ของกระบวนการหลัก.

2.ทำการ fork() เพื่อสร้างกระบวนการย่อย.

3.ตรวจสอบค่าที่ได้จาก fork():

ถ้า pid ไม่เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการหลัก (parent process) จึงพิมพ์ "I'am the parent with PID [PID] and PPID [PPID]." และแสดง PID ของ child process.

ถ้า pid เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการย่อย (child process) จึงให้ child process หลับไปเป็นเวลา 4 วินาที (โดยใช้ sleep(4)) เพื่อให้ parent process ทำงานและสิ้นสุด.

4.พิมพ์ "PID [PID] terminates." ทั้งใน parent และ child processes.

ผลลัพธ์ที่ได้จะแสดง Process ID (PID) และ Parent Process ID (PPID) ของทั้ง parent และ child processes ทั้งสอง. นอกจากนี้, จะมีการพิมพ์ "PID [PID] terminates." ทั้งใน parent และ child processes แสดงถึงลำดับการสิ้นสุดของกระบวนการ.

7.ตัวอย่างที่ 7

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(){
int pid;
pid = fork();
if(pid != 0){
        while (1){
                sleep(12)
                printf("Kaweewat \n");
        }
}else{
        exit (42);
        }
}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./ex7 &
[1] 4478
[iiuuuu@localhost lab3]$ ps
   PID TTY          TIME CMD
  4285 pts/0    00:00:00 bash
  4478 pts/0    00:00:00 ex7
  4480 pts/0    00:00:00 ex7 <defunct>
  4486 pts/0    00:00:00 ps
[iiuuuu@localhost lab3]$ kill4478Kaweewat

bash: kill4478: command not found...
[iiuuuu@localhost lab3]$ kill 4478
[1]+  Terminated              ./ex7
[iiuuuu@localhost lab3]$ ps
   PID TTY          TIME CMD
  4285 pts/0    00:00:00 bash
  4511 pts/0    00:00:00 ps
```

1.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process).

2.ตรวจสอบค่าที่ได้จาก fork():

ถ้า pid ไม่เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการหลัก (parent process) จึงเข้าไปในลูป while(1) และทำการหยุดการทำงานเป็นเวลา 100 วินาที ด้วย sleep(100) ซึ่งหมายถึงว่า parent process จะหยุดทำงานไม่หลับตลอดเวลา.

ถ้า pid เท่ากับ 0 แสดงว่าตอนนี้อยู่ในกระบวนการย่อย (child process) จึงใช้ exit(42) เพื่อส่งค่า 42 ไปยังระบบปฏิบัติการและทำให้ child process จบการทำงาน.

8. ทดลอง 1

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

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

        char *who;
        int i;

        if(fork()){
                who = "Kaweewat";
        } else{
                who = "child";
        }

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

        exit (0);
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./hw1
*frok1: Kaweewat
*frok1: Kaweewat
*frok1: Kaweewat
*frok1: Kaweewat
*frok1: Kaweewat
*frok1: Kaweewat
*frok1: child
*frok1: child
*frok1: child
*frok1: child
*frok1: child
*frok1: child
```

1.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process).

2.ตรวจสอบค่าที่ได้จาก fork():

ถ้าเป็น parent process (ค่าที่ได้จาก fork() ไม่เท่ากับ 0) จะกำหนดค่า who เป็น "Kaweewat."

ถ้าเป็น child process (ค่าที่ได้จาก fork() เท่ากับ 0) จะกำหนดค่า who เป็น "child"

3.ทำการพิมพ์ข้อความใน Loop for ที่ทำงานทั้งใน parent process และ child process.

4. parent process และ child process จะจบการทำงานด้วย exit(0).

9.ทดลอง 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 = "Kaweewat.";
                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);

}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./hw2
* 1. Kaweewat.: my pid =   4722, ppid =   4285
 * 1.   child: my pid =   4723, ppid =   4722
 * 2.   child: my pid =   4723, ppid =   4722
 * 2. Kaweewat.: my pid =   4722, ppid =   4285
 * 3.   child: my pid =   4723, ppid =   4722
 * 4.   child: my pid =   4723, ppid =   4722
 * 3. Kaweewat.: my pid =   4722, ppid =   4285
 * 5.   child: my pid =   4723, ppid =   4722
 * 6.   child: my pid =   4723, ppid =   4722
 * 4. Kaweewat.: my pid =   4722, ppid =   4285
 * 7.   child: my pid =   4723, ppid =   4722
 * 8.   child: my pid =   4723, ppid =   4722
 * 5. Kaweewat.: my pid =   4722, ppid =   4285
 * 9.   child: my pid =   4723, ppid =   4722
 *10.   child: my pid =   4723, ppid =   4722
 * 6. Kaweewat.: my pid =   4722, ppid =   4285
 * 7. Kaweewat.: my pid =   4722, ppid =   4285
 * 8. Kaweewat.: my pid =   4722, ppid =   4285
 * 9. Kaweewat.: my pid =   4722, ppid =   4285
 *10. Kaweewat.: my pid =   4722, ppid =   4285
```

1.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process).

2.ตรวจสอบค่าที่ได้จาก fork():

ถ้าเป็น parent process (ค่าที่ได้จาก fork() ไม่เท่ากับ 0) จะกำหนดค่า who เป็น "Kaweewat." และ n เป็น 2.

ถ้าเป็น child process (ค่าที่ได้จาก fork() เท่ากับ 0) จะกำหนดค่า who เป็น "child" และ n เป็น 1

3.ทำการพิมพ์ข้อความในลูป for ที่ทำงานทั้งใน parent process และ child process โดยให้แสดง PID และ PPID ของกระบวนการ.

4.ทั้ง parent process และ child process จะจบการทำงานด้วย exit(0).

9.ทอลอง 3

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

}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./gw3
pi =3.141593
```

1.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process).

2.ตรวจสอบค่าที่ได้จาก fork():

ถ้าเป็น parent process (ค่าที่ได้จาก fork() ไม่เท่ากับ 0) จะกำหนดค่า who เป็น "Kaweewat" และพิมพ์ค่าของ π (pi) ในลูปโดยใช้ฟังก์ชัน atan(1) (เท่ากับ π/4) และใช้ wait() เพื่อรอให้ child process ทำงานเสร็จสิ้น.

ถ้าเป็น child process (ค่าที่ได้จาก fork() เท่ากับ 0) จะกำหนดค่า who เป็น "child" และใช้ execlp() เพื่อโหลดโปรแกรมใหม่ ("date") แทนที่โปรแกรมที่อยู่ใน child process เดิม.

10.ทดลอง4

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

}
```

ผลลัพธ์

```
[iiuuuu@localhost lab3]$ ./hw4
pi =3.141593
```

1.ทำการ fork() เพื่อสร้างกระบวนการย่อย (child process).

2.ตรวจสอบค่าที่ได้จาก fork():

ถ้าเป็น parent process (ค่าที่ได้จาก fork() ไม่เท่ากับ 0) จะกำหนดค่า who เป็น "Kaweewat" และพิมพ์ค่าของ π (pi) ในลูปโดยใช้ฟังก์ชัน atan(1) (เท่ากับ π/4) และใช้ wait() เพื่อรอให้ child process ทำงานเสร็จสิ้น.

ถ้าเป็น child process (ค่าที่ได้จาก fork() เท่ากับ 0) จะกำหนดค่า who เป็น "child" และใช้ execlp() เพื่อโหลดโปรแกรมใหม่ ("my-script") แทนที่จะดำเนินการต่อไปตามโปรแกรมที่อยู่ใน child process เดิม และส่ง "a" และ "b" ให้เป็น argument ในการเรียก my-script.

สรุปผลการทดลอง

การทดลองการสร้างกระบวนการย่อยทั้ง parent and child การสร้างกระบวนการหลักและย่อย เช่นการ fork parent จะได้ค่าเป็นpid ของ child ถ้า fork child จะได้ = 0 ทำให้การจัดการทำงานได้ง่าย และเป็นขั้นเป็นตอนมากยิ่งขึ้น