ใบงานที่ 3

เรื่อง Process management (Fork)

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

จัดทำโดย

นางสาวรัชนีกร เชื้อดี 65543206077-1

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

ใบงานที่ 3

Process management (Fork)

บันทึกผลการทดลอง

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());
}
```

บรรทัดที่ 1: นำเข้าไลบรารี stdio.h ซึ่งจำเป็นสำหรับการใช้คำสั่ง printf()

บรรทัดที่ 2: นำเข้าไลบรารี unistd.h ซึ่งจำเป็นสำหรับการใช้ฟังก์ชัน fork()

บรรทัดที่ 3: จุดเริ่มต้นของโปรแกรม

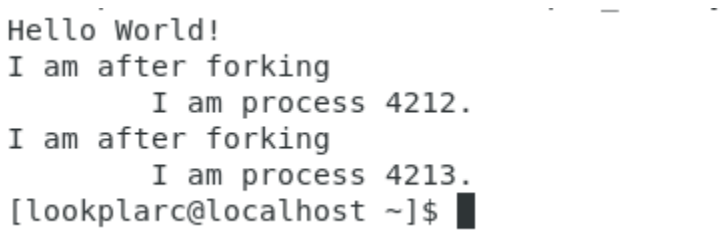
บรรทัดที่ 4: แสดงข้อความ "Hello World!" บนจอภาพ

บรรทัดที่ 5: แสดงข้อความ "I am after forking" บนจอภาพ

บรรทัดที่ 6: เรียกใช้ฟังก์ชัน fork() ซึ่งจะสร้างกระบวนการลูกขึ้นมาใหม่

บรรทัดที่ 7: แสดงข้อความ "I am process %d." บนจอภาพ โดยที่ %d จะแทนด้วยค่า ID ของกระบวนการ

ผลลัพธ์



กระบวนการหลักและกระบวนการลูกจะแสดงข้อความ "Hello World!" บนจอภาพ

กระบวนการหลักและกระบวนการลูกจะแสดงข้อความ "I am after forking" บนจอภาพ

กระบวนการหลักและกระบวนการลูกจะแสดงข้อความ "I am process %d." บนจอภาพ โดยที่ค่า ID ของกระบวนการจะแตกต่างกัน

ตัวอย่างที่ 2

```c
#include<stdio.h>
#include<unistd.h>
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());
}
```

บรรทัดที่ 1-2: นำเข้าไลบรารี stdio.h และ unistd.h ซึ่งจำเป็นสำหรับการใช้คำสั่ง printf() และ fork()

บรรทัดที่ 3: ประกาศตัวแปร pid ของชนิด int ซึ่งจะใช้เก็บค่า ID ของกระบวนการลูก

บรรทัดที่ 4: แสดงข้อความ "Hello World!" บนจอภาพ

บรรทัดที่ 5: แสดงข้อความว่ากระบวนการนี้เป็นกระบวนการหลักและแสดงค่า ID ของกระบวนการหลักโดยใช้ฟังก์ชัน getpid()

บรรทัดที่ 6: แสดงข้อความว่าขณะนี้อยู่ระหว่างการเรียกใช้ฟังก์ชัน fork()

บรรทัดที่ 7: เรียกใช้ฟังก์ชัน fork() ซึ่งจะสร้างกระบวนการลูกขึ้นมาใหม่ ค่า ID ของกระบวนการลูกจะถูกเก็บไว้ในตัวแปร pid

บรรทัดที่ 8: แสดงข้อความว่าขณะนี้เสร็จสิ้นการเรียกใช้ฟังก์ชัน fork() แล้ว

บรรทัดที่ 9-12: ตรวจสอบค่าของตัวแปร pid หาก pid เท่ากับ 0 หมายความว่าขณะนี้กำลังทำงานอยู่ในกระบวนการลูก จึงแสดงข้อความว่ากระบวนการนี้เป็นกระบวนการลูกและแสดงค่า ID ของกระบวนการลูก หาก pid ไม่เท่ากับ 0 หมายความว่าขณะนี้ยังคงทำงานอยู่ในกระบวนการหลัก จึงแสดงข้อความว่ากระบวนการนี้เป็นกระบวนการหลักและแสดงค่า ID ของกระบวนการหลัก

บรรทัดที่ 13: แสดงข้อความว่าขณะนี้เสร็จสิ้นการประมวลผลในกระบวนการหลัก

ผลลัพธ์

```
Hello World!
I am the parent process and pid is : 4363.
Here i am before use of forking
Here I am just after forking
I am the parent process and pid is : 4363.
Here I am just after forking
I am the child process and pid is : 4364.
[lookplarc@localhost ~]$
```

หากเรารันโค้ดนี้ ผลลัพธ์ที่ได้จะแตกต่างกันไปตามค่าที่คืนโดยฟังก์ชัน fork() หากฟังก์ชัน fork() คืนค่าเป็นค่าบวก หมายความว่ากระบวนการที่เรียกใช้ฟังก์ชัน fork() จะกลายเป็นกระบวนการหลัก และกระบวนการลูกจะมี ID เป็นค่าบวกที่คืนโดยฟังก์ชัน fork()

ตัวอย่างที่ 3

```c
#include<stdio.h>
#include<unistd.h>
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());
}
```

บรรทัดที่ 1-2: นำเข้าไลบรารี stdio.h และ unistd.h ซึ่งจำเป็นสำหรับการใช้คำสั่ง printf() และ fork()

บรรทัดที่ 3: กำหนดฟังก์ชัน main() ซึ่งจุดเริ่มต้นของโปรแกรม

บรรทัดที่ 4: แสดงข้อความว่าขณะนี้อยู่ระหว่างการเรียกใช้ฟังก์ชัน fork() ครั้งแรก

บรรทัดที่ 5: เรียกใช้ฟังก์ชัน fork() เพื่อสร้างกระบวนการลูกขึ้นมาใหม่ กระบวนการลูกและกระบวนการหลักจะยังคงดำเนินการตามโค้ดหลังจากบรรทัดนี้

บรรทัดที่ 6: แสดงข้อความว่าขณะนี้เสร็จสิ้นการเรียกใช้ฟังก์ชัน fork() แล้ว

บรรทัดที่ 7: เรียกใช้ฟังก์ชัน fork() อีกครั้งเพื่อสร้างกระบวนการลูกขึ้นมาใหม่ กระบวนการลูกและกระบวนการหลักจะยังคงดำเนินการตามโค้ดหลังจากบรรทัดนี้

บรรทัดที่ 8: แสดงข้อความว่าขณะนี้เสร็จสิ้นการเรียกใช้ฟังก์ชัน fork() แล้ว

บรรทัดที่ 9: เรียกใช้ฟังก์ชัน getpid() เพื่อรับค่า ID ของกระบวนการปัจจุบัน และแสดงข้อความพร้อมค่า ID ของกระบวนการ สัญลักษณ์ \t\t ใช้เพื่อเพิ่มการเยื้องให้กับผลลัพธ์

ผลลัพธ์

```
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 4526!
Here I am just after first forking statement
Here I am just after second forking statement
                Hello World from process 4527!
Here I am just after second forking statement
                Hello World from process 4528!
Here I am just after second forking statement
                Hello World from process 4530!
[lookplarc@localhost ~]$
```

ผลลัพธ์ที่ได้จะแตกต่างกันไปตามค่าที่คืนโดยฟังก์ชัน fork() หากฟังก์ชัน fork() คืนค่าเป็นค่าบวก หมายความว่า กระบวนการที่เรียกใช้ฟังก์ชัน fork() จะกลายเป็นกระบวนการหลัก และกระบวนการลูกจะมี ID เป็นค่าบวกที่คืนโดยฟังก์ชัน fork()

ตัวอย่างที่ 4

```
#include<stdio.h>
#include<sys/wait.h>

void exit(int status);

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");
}
}
```

บรรทัดที่ 1-2: นำเข้าไลบรารี stdio.h และ sys/wait.h ซึ่งจำเป็นสำหรับการใช้คำสั่ง printf() และ fork()

บรรทัดที่ 3: ประกาศตัวแปร pid และ status ของชนิด int

บรรทัดที่ 4: แสดงข้อความ "Hello World!" บนจอภาพ

บรรทัดที่ 5: สร้างกระบวนการลูกโดยใช้ฟังก์ชัน fork()

บรรทัดที่ 6: ตรวจสอบค่าที่คืนโดยฟังก์ชัน fork() หากค่าที่คืนเป็น -1 แสดงว่าเกิดข้อผิดพลาดในการเรียกใช้ฟังก์ชัน fork() ในกรณีนี้ ฟังก์ชัน perror() จะถูกเรียกใช้เพื่อแสดงข้อความแสดงข้อผิดพลาด และโปรแกรมจะออกโดยแสดงรหัสสถานะเป็น 1

บรรทัดที่ 7-9: ตรวจสอบค่าของตัวแปร pid หากค่าของตัวแปร pid เท่ากับ 0 แสดงว่าปัจจุบันกำลังทำงานอยู่ในกระบวนการลูก ในกรณีนี้ กระบวนการลูกจะแสดงข้อความ "I am the child process." และสิ้นสุดการทำงาน

ผลลัพธ์

```
Hello World!
I am the child process.
I am the parent process.
[lookplarc@localhost ~]$
```

โค้ดเริ่มต้นด้วยการพิมพ์ข้อความ "Hello World!" จากนั้นจะสร้างกระบวนการลูกโดยใช้ฟังก์ชัน fork() หากเกิดข้อผิดพลาดในระหว่างการดำเนินการ fork() โปรแกรมจะออกโดยแสดงข้อความแสดงข้อผิดพลาด

หากไม่มีข้อผิดพลาด โค้ดจะดำเนินต่อไปในทั้งกระบวนการหลักและกระบวนการลูก กระบวนการหลักจะเรียกใช้ฟังก์ชัน wait() เพื่อรอจนกว่ากระบวนการลูกจะสิ้นสุดการทำงาน จากนั้นจะพิมพ์ข้อความ "I am the parent process."

กระบวนการลูกจะพิมพ์ข้อความ "I am the child process." และสิ้นสุดการทำงาน

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)
        {
                printf("%d: My child's pid is %d\n", getpid(), forkresult);
        }
        else
        {
                printf("%d: Hi! I am the child. \n",getpid());
        }
        printf("%d: like father like son. \n", getpid());
}
```

บรรทัดที่ 1-4: นำเข้าไลบรารีที่จำเป็นสำหรับฟังก์ชันต่างๆ ที่ใช้ในโค้ด ไลบรารี stdio.h จำเป็นสำหรับฟังก์ชัน printf() ไลบรารี unistd.h จำเป็นสำหรับฟังก์ชัน fork() ไลบรารี stdlib.h จำเป็นสำหรับฟังก์ชัน exit() และไลบรารี sys/wait.h จำเป็นสำหรับฟังก์ชัน wait()

บรรทัดที่ 5: ประกาศฟังก์ชันหลัก ซึ่งเป็นจุดเริ่มต้นของโปรแกรม

บรรทัดที่ 6: ประกาศตัวแปร forkresult ของชนิด int เพื่อเก็บค่าผลลัพธ์ของฟังก์ชัน fork()

บรรทัดที่ 7-8: พิมพ์ข้อความจากกระบวนการหลัก ข้อความแรกระบุว่ากระบวนการหลักคือกระบวนการที่เรียกใช้ฟังก์ชัน main() และข้อความที่สองระบุว่ากระบวนการหลักกำลังจะเรียกใช้ฟังก์ชัน fork() ฟังก์ชัน getpid() ใช้ในการรับค่า ID ของกระบวนการปัจจุบัน

บรรทัดที่ 9: เรียกใช้ฟังก์ชัน fork() เพื่อสร้างกระบวนการลูก ค่าผลลัพธ์ของฟังก์ชัน fork() จะเก็บไว้ในตัวแปร forkresult

บรรทัดที่ 10-11: ตรวจสอบค่าผลลัพธ์ของฟังก์ชัน fork() หากค่าผลลัพธ์ไม่เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการหลัก และจะข้ามไปบล็อก else หากค่าผลลัพธ์เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการลูก และจะข้ามไปบล็อก if

บรรทัดที่ 12: พิมพ์ข้อความจากกระบวนการหลัก ข้อความนี้ระบุค่า ID ของกระบวนการหลักและค่า ID ของกระบวนการลูก

บรรทัดที่ 13-14: พิมพ์ข้อความจากกระบวนการลูก ข้อความแรกระบุว่ากระบวนการลูกกำลังทำงานอยู่ และข้อความที่สองเป็นข้อความเล่นๆ

ผลลัพธ์

```
5237: I am the parent. Remember my number!
5237: I am now going to fork ...
5237: My child's pid is 5238
5237: like father like son.
5238: Hi! I am the child.
5238: like father like son.
```

โค้ดเริ่มต้นด้วยการนำเข้าไลบรารีที่จำเป็นและประกาศฟังก์ชันหลัก ภายในฟังก์ชันหลัก จะประกาศตัวแปรเพื่อเก็บค่าผลลัพธ์ของฟังก์ชัน fork() และพิมพ์ข้อความจากกระบวนการหลัก

ตัวอย่างที่ 6

```c
#include <stdio.h>
main()
{
        int pid;
        printf("I'm the original process with PID %d and PPID %d.\n",getpid(),getppid()
);
        pid = fork();
        if(pid != 0)
        {
                printf("I'm the parent with PID %d and PPID %d\n",getpid(),getppid());
                printf("My child's PID is %d\n",pid);
        }
        else
        {
                sleep(4);
                printf("I'm the child with PID %d and PPID %d.\n",getpid(),getppid());
        }
        printf("PID %d terminates.\n",getpid());
}
~
```

บรรทัดที่ 1: นำเข้าไลบรารี stdio.h ซึ่งจำเป็นสำหรับฟังก์ชัน printf()

บรรทัดที่ 2: ประกาศฟังก์ชันหลัก ซึ่งเป็นจุดเริ่มต้นของโปรแกรม

บรรทัดที่ 3: ประกาศตัวแปร pid ของชนิด int เพื่อเก็บค่าผลลัพธ์ของฟังก์ชัน fork()

บรรทัดที่ 4: พิมพ์ข้อความระบุว่ากระบวนการต้นฉบับคือกระบวนการที่เรียกใช้ฟังก์ชัน main() ข้อความนี้ระบุค่า ID ของกระบวนการต้นฉบับและค่า ID ของกระบวนการแม่ (PPID) ฟังก์ชัน getpid() ใช้ในการรับค่า ID ของกระบวนการปัจจุบัน และฟังก์ชัน getppid() ใช้ในการรับค่า ID ของกระบวนการแม่

บรรทัดที่ 5: เรียกใช้ฟังก์ชัน fork() เพื่อสร้างกระบวนการลูก ฟังก์ชัน fork() จะคืนค่า 0 ให้กับกระบวนการลูก และค่า ID ของกระบวนการลูกให้กับกระบวนการแม่

บรรทัดที่ 6: ตรวจสอบค่าผลลัพธ์ของฟังก์ชัน fork() หากค่าผลลัพธ์ไม่เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการหลัก และจะพิมพ์ข้อความเกี่ยวกับกระบวนการหลัก หากค่าผลลัพธ์เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการลูก และจะพิมพ์ข้อความเกี่ยวกับกระบวนการลูก

บรรทัดที่ 7: สิ้นสุดฟังก์ชันหลัก

ผลลัพธ์

```
I'm the original process with PID 3085 and PPID 2822.
I'm the parent with PID 3085 and PPID 2822
My child's PID is 3086
PID 3085 terminates.
[lookplarc@localhost ~]$ I'm the child with PID 3086 and PPID 1.
PID 3086 terminates.
```

โค้ดเริ่มต้นด้วยการนำเข้าไลบรารีที่จำเป็นและประกาศฟังก์ชันหลัก ภายในฟังก์ชันหลัก จะประกาศตัวแปรเพื่อเก็บค่าผลลัพธ์ของฟังก์ชัน fork() และพิมพ์ข้อมูลเกี่ยวกับกระบวนการต้นฉบับ

จากนั้น จะเรียกใช้ฟังก์ชัน fork() เพื่อสร้างกระบวนการลูก

กระบวนการหลัก

หากค่าผลลัพธ์ของฟังก์ชัน fork() ไม่เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการหลัก กระบวนการหลักจะพิมพ์ข้อมูลเกี่ยวกับกระบวนการหลักและกระบวนการลูก

กระบวนการลูก

หากค่าผลลัพธ์ของฟังก์ชัน fork() เท่ากับ 0 แสดงว่าขณะนี้กำลังทำงานอยู่ในกระบวนการลูก กระบวนการลูกจะรอ 4 วินาที จากนั้นจะพิมพ์ข้อมูลเกี่ยวกับกระบวนการลูกและสิ้นสุดการทำงาน

หากเรารันโค้ดนี้ ผลลัพธ์ที่ได้จะแตกต่างกันไปตามค่าที่คืนโดยฟังก์ชัน fork() หากฟังก์ชัน fork() คืนค่าเป็นค่าบวก หมายความว่ากระบวนการที่เรียกใช้ฟังก์ชัน fork() จะกลายเป็นกระบวนการหลัก และกระบวนการลูกจะมี ID เป็นค่า

ตัวอย่างที่ 7

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

char *name = "Ratchaneekorn Chuadee";

main()
{
  int pid;
  pid = fork();
  if (pid != 0) {
    while (1);
    sleep(100);
  } else {
    while (1) {
      printf("%s\n", name);
      sleep(10 + (rand() % 5));
    }
  }
}
```

บรรทัดที่ 1: นำเข้าไลบรารี stdio.h ซึ่งประกอบด้วยฟังก์ชันสำหรับการรับและแสดงผลข้อมูล

บรรทัดที่ 2: นำเข้าไลบรารี stdlib.h ซึ่งประกอบด้วยฟังก์ชันพื้นฐานต่างๆ เช่น fork() และ exit()

บรรทัดที่ 3: สร้างตัวแปร name เป็นค่าคงที่ของสตริงชื่อนักศึกษา

บรรทัดที่ 4: ใช้ฟังก์ชัน fork() เพื่อแยกกระบวนการหลักออกเป็นกระบวนการลูก

บรรทัดที่ 5: ตรวจสอบว่ากระบวนการปัจจุบันเป็นกระบวนการลูกหรือไม่

บรรทัดที่ 6: ในลูป while(1) ของกระบวนการลูก จะมีการวนซ้ำการแสดงผลชื่อนักศึกษาบนหน้าจอทุกๆ 10-15 วินาที

บรรทัดที่ 7: ใช้ฟังก์ชัน sleep() เพื่อหน่วงเวลา 10-15 วินาที

ผลลัพธ์

```
[lookplarc@localhost ~]$ ./a.out &
[1] 4296
Ratchaneekorn Chuadee
[lookplarc@localhost ~]$ Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
^C
[lookplarc@localhost ~]$ ps
   PID TTY          TIME CMD
  4076 pts/2    00:00:00 bash
  4296 pts/2    00:01:41 a.out
  4298 pts/2    00:00:00 a.out
  4327 pts/2    00:00:00 ps
[lookplarc@localhost ~]$ Ratchaneekorn Chuadee
^C
[lookplarc@localhost ~]$ kill 4296
[1]+  Terminated              ./a.out
[lookplarc@localhost ~]$ Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
^C
[lookplarc@localhost ~]$ ps
   PID TTY          TIME CMD
  4076 pts/2    00:00:00 bash
  4298 pts/2    00:00:00 a.out
  4352 pts/2    00:00:00 ps
[lookplarc@localhost ~]$ Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
Ratchaneekorn Chuadee
```

โปรแกรมจะแสดงชื่อนักศึกษา "Ratchaneekorn Chuadee" บนหน้าจอทุกๆ 10-15 วินาที คุณสามารถทำงานอื่นบนเทอร์มินัลได้ในขณะที่โปรแกรมกำลังทำงานอยู่

การทดลองที่ 4

1. Examples of Process 1

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

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

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

        for (i = 0; i < 6; i++) {
                printf("*fork1: %s\n", who);
        }

        exit (0);
}
```

บรรทัดที่ 1 และ 2: นำเข้าไลบรารีมาตรฐาน stdlib.h และ stdio.h

บรรทัดที่ 3: ประกาศตัวแปร who เป็นสตริง

บรรทัดที่ 4: ประกาศตัวแปร i เป็นจำนวนเต็ม

บรรทัดที่ 5: ใช้ฟังก์ชัน fork() เพื่อแยกกระบวนการออกเป็นสองกระบวนการ

- ถ้ากระบวนการหลัก (parent process) คืนค่าค่าเป็น 0 แสดงว่ากระบวนการย่อย (child process) สำเร็จ
- ถ้ากระบวนการหลัก (parent process) คืนค่าค่าเป็นค่าอื่น แสดงว่ากระบวนการย่อย (child process) ล้มเหลว

บรรทัดที่ 6: ถ้ากระบวนการหลัก (parent process) คืนค่าค่าเป็น 0 แสดงว่าตัวแปร who จะถูกตั้งค่าเป็น "Ratchaneekorn Chuadee"

บรรทัดที่ 7: ถ้ากระบวนการหลัก (parent process) คืนค่าค่าเป็นค่าอื่น แสดงว่าตัวแปร who จะถูกตั้งค่าเป็น "child"

บรรทัดที่ 8: วนลูป 6 รอบ

- ในรอบที่ 1 ถึง 5 ตัวแปร who จะขึ้นอยู่กับค่าที่คืนกลับมาจากฟังก์ชัน fork()
- ในรอบที่ 6 ตัวแปร who จะยังคงเป็นค่าเดิมที่ประกาศไว้ในบรรทัดที่ 3

บรรทัดที่ 9: ยุติการทำงานของกระบวนการ

ผลลัพธ์

```
*fork1: Ratchaneekorn Chuadee
*fork1: Ratchaneekorn Chuadee
*fork1: Ratchaneekorn Chuadee
*fork1: Ratchaneekorn Chuadee
*fork1: Ratchaneekorn Chuadee
*fork1: Ratchaneekorn Chuadee
*fork1: child
*fork1: child
*fork1: child
*fork1: child
*fork1: child
*fork1: child
```

กระบวนการแม่จะแสดงข้อความ "fork1: Ratchaneekorn Chuadee" ออกมาก่อน จากนั้นกระบวนการลูกทั้งสองจะแสดงข้อความ "fork1: child" ออกมาทีละตัว

2. **การดำเนินการเป็น Asynchronous**

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

int main (int argc, char *argv[]) {
        int i;
        char *who;
        int n;

        if (fork ()) {
                who = "Ratchaneekorn Chuadee";
                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);
"fork2.c" 24L, 384C
```

บรรทัดที่ 1-3: เป็นการนำเข้าไลบรารี unistd.h, stdio.h, และ stdlib.h ซึ่งจำเป็นสำหรับฟังก์ชัน fork(), printf(), และ exit() ตามลำดับ

บรรทัดที่ 4: เป็นจุดเริ่มต้นของฟังก์ชัน main() ซึ่งเป็นฟังก์ชันหลักของโปรแกรม

บรรทัดที่ 5: ประกาศตัวแปร i เป็นจำนวนเต็มเพื่อใช้วนลูป

บรรทัดที่ 6: ประกาศตัวแปร who เป็นสตริงเพื่อเก็บชื่อของนักศึกษา

บรรทัดที่ 7: เรียกใช้ฟังก์ชัน fork() ซึ่งจะสร้างกระบวนการใหม่ขึ้นมา

- หากฟังก์ชัน fork() คืนค่าเป็น 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการลูก
- ในบรรทัดนี้ กระบวนการลูกจะตั้งค่าตัวแปร who เป็นชื่อของผู้เขียนโปรแกรม

บรรทัดที่ 8: หากฟังก์ชัน fork() คืนค่าเป็นค่าอื่นที่ไม่ใช่ 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการแม่

- ในบรรทัดนี้ กระบวนการแม่จะตั้งค่าตัวแปร who เป็นคำว่า "child"

บรรทัดที่ 9: เริ่มต้นการวนลูป for ซึ่งจะวนซ้ำ 10 ครั้ง

บรรทัดที่ 10: เรียกใช้ฟังก์ชัน printf() เพื่อแสดงข้อความ "fork1: [ชื่อของเด็ก]" บนหน้าจอ

บรรทัดที่ 11: เรียกใช้ฟังก์ชัน fflush() เพื่อบังคับให้ข้อความที่แสดงผลจะถูกส่งไปยังอุปกรณ์แสดงผลทันที

บรรทัดที่ 12: เรียกใช้ฟังก์ชัน sleep() เพื่อพักการทำงานชั่วคราวเป็นเวลา 2 วินาที

บรรทัดที่ 13: เรียกใช้ฟังก์ชัน exit() เพื่อสิ้นสุดการทำงานของกระบวนการ

ผลลัพธ์

```
[lookplarc@localhost ~]$ gcc -o fork2 fork2.c
[lookplarc@localhost ~]$ ./fork2
* 1. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 1.   child: my pid =   4649, ppid =   4648
* 2.   child: my pid =   4649, ppid =   4648
* 2. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 3.   child: my pid =   4649, ppid =   4648
* 4.   child: my pid =   4649, ppid =   4648
* 3. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 5.   child: my pid =   4649, ppid =   4648
* 6.   child: my pid =   4649, ppid =   4648
* 4. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 7.   child: my pid =   4649, ppid =   4648
* 8.   child: my pid =   4649, ppid =   4648
* 5. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 9.   child: my pid =   4649, ppid =   4648
*10.   child: my pid =   4649, ppid =   4648
* 6. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 7. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 8. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
* 9. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
*10. Ratchaneekorn Chuadee: my pid =   4648, ppid =   2822
[lookplarc@localhost ~]$
```

กระบวนการแม่จะแสดงข้อความ "fork1: [ชื่อของนักศึกษา]" ออกมาก่อน จากนั้นกระบวนการลูกทั้งสองจะแสดงข้อความ "fork1: child" ออกมาทีละตัว กระบวนการลูกแต่ละตัวจะแสดงข้อความ "fork1: child" ออกมา 5 ครั้ง

3. You Can Exec a Program

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

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

        if (fork ()) {
                who = "Ratchaneekorn Chuadee";
                printf ("pi=%f\n", 4*atan(1));
                wait (&status);
                exit (0);
        } else {
                who = "child";
                execlp ("/usr/bin/date", "date", (char *)0);
        }
}
~
~
~
~
~
                                                        4,19          All
```

บรรทัดที่ 1-4: เป็นการนำเข้าไลบรารี unistd.h, stdio.h, math.h, และ stdlib.h ซึ่งจำเป็นสำหรับฟังก์ชัน fork(), printf(), atan(), และ execlp() ตามลำดับ

บรรทัดที่ 5: เป็นจุดเริ่มต้นของฟังก์ชัน main() ซึ่งเป็นฟังก์ชันหลักของโปรแกรม

บรรทัดที่ 6: ประกาศตัวแปร who เป็นสตริงเพื่อเก็บชื่อของนักศึกษา

บรรทัดที่ 7: ประกาศตัวแปร status เป็นจำนวนเต็มเพื่อเก็บสถานะการสิ้นสุดของกระบวนการลูก

บรรทัดที่ 8: เรียกใช้ฟังก์ชัน fork() ซึ่งจะสร้างกระบวนการใหม่ขึ้นมา

- หากฟังก์ชัน fork() คืนค่าเป็น 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการลูก
- ในบรรทัดที่ 8 กระบวนการลูกจะตั้งค่าตัวแปร who เป็นคำว่า "child"

บรรทัดที่ 9: หากฟังก์ชัน fork() คืนค่าเป็นค่าอื่นที่ไม่ใช่ 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการแม่

- ในบรรทัดที่ 9 กระบวนการแม่จะตั้งค่าตัวแปร who เป็นชื่อของผู้เขียนโปรแกรม

บรรทัดที่ 10: เรียกใช้ฟังก์ชัน execlp() เพื่อรันคำสั่ง date ในโหมดเบื้องหลัง

บรรทัดที่ 11: เรียกใช้ฟังก์ชัน printf() เพื่อแสดงค่า pi บนหน้าจอ

บรรทัดที่ 12: เรียกใช้ฟังก์ชัน wait() เพื่อรอให้กระบวนการลูกสิ้นสุดลง

บรรทัดที่ 13: เรียกใช้ฟังก์ชัน exit() เพื่อสิ้นสุดการทำงานของกระบวนการแม่

ผลลัพธ์

```
pi=3.141593
Tue Dec  5 10:51:20 PST 2023
```

กระบวนการแม่จะแสดงค่า pi บนหน้าจอก่อน จากนั้นกระบวนการลูกจะรันคำสั่ง date ในโหมดเบื้องหลัง ซึ่งจะแสดงวันที่และเวลาปัจจุบันบนหน้าจอ

4. You Can Also Exec a Script

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

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

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

~
```

บรรทัดที่ 1-4: เป็นการนำเข้าไลบรารี unistd.h, stdio.h, math.h, และ stdlib.h ซึ่งจำเป็นสำหรับฟังก์ชัน fork(), printf(), atan(), และ execlp() ตามลำดับ

บรรทัดที่ 5: เป็นจุดเริ่มต้นของฟังก์ชัน main() ซึ่งเป็นฟังก์ชันหลักของโปรแกรม

บรรทัดที่ 6: ประกาศตัวแปร who เป็นสตริงเพื่อเก็บชื่อของนักศึกษา

บรรทัดที่ 7: ประกาศตัวแปร status เป็นจำนวนเต็มเพื่อเก็บสถานะการสิ้นสุดของกระบวนการลูก

บรรทัดที่ 8: เรียกใช้ฟังก์ชัน fork() ซึ่งจะสร้างกระบวนการใหม่ขึ้นมา

- หากฟังก์ชัน fork() คืนค่าเป็น 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการลูก
- ในบรรทัดที่ 8 กระบวนการลูกจะตั้งค่าตัวแปร who เป็นคำว่า "child"

บรรทัดที่ 9: หากฟังก์ชัน fork() คืนค่าเป็นค่าอื่นที่ไม่ใช่ 0 แสดงว่ากระบวนการปัจจุบันเป็นกระบวนการแม่

- ในบรรทัดที่ 9 กระบวนการแม่จะตั้งค่าตัวแปร who เป็นชื่อของผู้เขียนโปรแกรม

บรรทัดที่ 10: เรียกใช้ฟังก์ชัน execlp() เพื่อรันสคริปต์ my-script ในโหมดเบื้องหลัง

บรรทัดที่ 11: เรียกใช้ฟังก์ชัน printf() เพื่อแสดงค่า pi บนหน้าจอ

บรรทัดที่ 12: เรียกใช้ฟังก์ชัน wait() เพื่อรอให้กระบวนการลูกสิ้นสุดลง

บรรทัดที่ 13: เรียกใช้ฟังก์ชัน exit() เพื่อสิ้นสุดการทำงานของกระบวนการแม่

ผลลัพธ์

```
pi=3.141593
```

กระบวนการแม่จะแสดงค่า pi บนหน้าจอก่อน จากนั้นกระบวนการลูกจะรันสคริปต์ my-script ในโหมดเบื้องหลัง ซึ่งจะแสดงผลลัพธ์ของสคริปต์บนหน้าจอ

จากการทดลอง มีข้อแตกต่างกันอย่างไรบ้าง จงอธิบาย

1. กระบวนการหลักและกระบวนการลูก ทำงานพร้อมกันโดยอิสระจากกัน หมายความว่า กระบวนการหลักและกระบวนการลูกสามารถทำงานพร้อมกันได้โดยไม่ต้องรอให้กันและกันทำงานเสร็จก่อน
2. กระบวนการลูก จะสิ้นสุดลงก็ต่อเมื่อถูกบังคับให้สิ้นสุดลงโดยคำสั่ง kill() หมายความว่า กระบวนการลูกจะทำงานต่อไปเรื่อยๆ จนกว่าจะถูกบังคับให้สิ้นสุดลงด้วยคำสั่ง kill()

สรุปผลการทดลอง

ฟังก์ชัน fork() สามารถสร้างกระบวนการใหม่ที่มีโค้ดและทรัพยากรทั้งหมดของกระบวนการหลัก จากนั้นกระบวนการใหม่จะทำงานต่อไปโดยอิสระจากกระบวนการหลัก กระบวนการลูก จะถูกสร้างขึ้นด้วยหมายเลข PID (Process ID) ใหม่ที่แตกต่างจากกระบวนการหลัก กระบวนการลูก สามารถเข้าถึงทรัพยากรของกระบวนการหลักได้ เช่น หน่วยความจำ ไฟล์ และอุปกรณ์ กระบวนการลูก สามารถสื่อสารกับกระบวนการหลักได้โดยใช้สัญญาณ (signal)

ฟังก์ชัน fork() เป็นฟังก์ชันที่สำคัญของระบบปฏิบัติการ Unix/Linux ใช้ในการสร้างกระบวนการใหม่ได้อย่างรวดเร็วและมีประสิทธิภาพ