การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

1. วัตถุประสงค์

- เพื่อให้มีความรู้ความเข้าใจ กระบวนการมีอะไรบ้าง
- เพื่อให้มีความรู้ความเข้าใจการสร้างกระบวนการ
- เพื่อให้นักศึกษาสามารถใช้คำสั่งพื้นฐานของ CentOS ได้
- เพื่อให้นักศึกษาสามารถใช้คำสั่งพื้นฐานเกี่ยวกับกระบวนการ Unix

2. อุปกรณ์ที่ใช้งาน

- เครื่องคอมพิวเตอร์ 1 เครื่องที่ติดตั้ง Virtualbox และมีระบบปฏิบัติการ CentOS ใน Virtualbox

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ทฤษฎี

สำหรับปฏิบัติการทดลองนี้จะอธิบายถึงวิธีการที่โปรแกรมสามารถ สร้างและยุติกระบวนการที่ ควบคุม child process มีการดำเนินงานที่เกี่ยวข้องดังนี้ การสร้าง child process ใหม่ที่ก่อให้เกิดกระบวนการใหม่ในการรันโปรแกรม และการประสานงานของโปรแกรมต้นฉบับ child process

แนวคิดการสร้างกระบวนการ

หลักพื้นฐานในการจัดโปรเซสของ UNIX จะแบ่งเป็น 2 ปฏิบัติการคือการสร้างโปรเซส และการรันโปรแกรมใหม่ ในการสร้างโปรเซสใหม่จะสร้างจาก systen call ที่ชื่อ fork และโปรแกรมจะมีการรันหลังจากเรียก execve ทั้งสองเป็นฟังก์ชันที่สำคัญ การสร้างโปรแซสจาก fork นี้อาจจะสร้างในขณะที่ไม่ได้รันโปรแกรมก็ได้ โปรเซสย่อยหรือโปรเซสลูกนี้จะเอ็กซิคิวซ์โดยตรงในโปรแกรมที่โปรเซสแม่รับอยู่ ส่วนการรันโปรแกรมใหม่ไม่จำเป็นต้องสร้างโปรเซสใหม่ก็ ได้ โปรเซสจะเรียก exeve เวลาใดก็ได้ โปรแกรมที่กำลังรันจะหยุดทันที และโปรแกรมจะเริ่มเอ็กซิคิวซ์ในภาวะแวดล้อมของโปรเซสที่มีอยู่ถ้าโปรเซสแม่ต้องการจะแก้ไขสภาวะแวดล้อมที่โปรแกรมใช้รันก็สามรถทำได้ หลังจากนั้นโปรแกรมเดิมยังคงรันได้ต่อไปในโปรเซสลูก จำเป็นต้องมีการ system call เพื่อแก้ไขโปรเซสลูกนั้นก่อนที่จะจบการเอ็กซิคิวต์โปรแกรมนั้น

สำหรับ UNIX โปรเซสจะมีข่าวสารทั้งหมดที่ระบบปฏิบัติการดูแลเพื่อติดตาม context ของการเอ็กซิคิวต์ในโปรแกรมหนึ่งส่วนใน Linux เราสามารถแบ่ง context เป็นส่วนต่าง ๆ ได้สำหรับคุณสมบัติของโปรเซสแบ่งเป็นสามกลุ่มได้คือ identity ของโปรเซส environment ของโปรเซสและ context ของโปรเซส

identity ของโปรเซส

Identity ของโปรเซสประกอบด้วยไอเท็มหลัก ๆ ดังนี้

หมายเลขโปรเซส Process ID: แต่ละโปรเซสมีหมายเลขเฉพาะที่เรียกว่า PID โดย PID ใช้เพื่อกำหนดลักษณะเฉพาะของโปรเซสให้กับระบบปฏิบัติการเมื่อแอปพริเคชันส่งสัญญาณ systeme call ออกมา, แก้ไขหรือคอยโปรเซสอื่น ยิ่งกว่านั้นหมายเลขโปรเซสยังสัมพันธ์กับกลุ่มของโปรเซส และล็อกอินเซสชัน

Credential : แต่ละโปรเซสจะต้องสัมพันธ์กับหมายเลขผู้ใช้ในกลุ่ม ID หนึ่งกลุ่มหรือมากกว่าที่พิจารณาสิทธิของโปรเซส ในการเอ็กเซสรีซอร์ส และไฟล์

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

Personality : บุคลิกของโปรเซสไม่มีใน UINX แต่พบใน Linux ซึ่งใน Linux แต่ละโปรเ.เซสจะสัมพันธ์กับหมายเลข Personality ที่ใช้ในไลบรารีเสมือนเพื่อร้องขอ system call ที่เข้ากันได้กับ UNIX

หมายเลขเหล่านี้ส่วนมากอยู่ในการควบคุมที่จำกัดของโปรเซสเอง กลุ่มของโปรเซสและหมายเลขฟังก์ชั่นสามารถเปลี่ยนแปลงได้ถ้าโปรเซสต้องการเริ่มต้นกลุ่มหรือเซสชันใหม่ ส่วน credential สามารถเปลี่ยนแปลงตามการรักษาความปลอดภัยที่เลือก อย่างไรก็ตามหมายเลข PID หลักของโปรเซสไม่สามารถเปลี่ยนแปลงได้ เป็นหมายเลขเฉพาะของโปรเซสนั้นจนกว่าจะสิ้นสุด

Environment ของโปรเซส

environment ของโปรเซส ได้รับช่วงต่อมาจากโปรเซสแม่ ที่ประกอบด้วยสองส่วน คือ argument vector และ environment vector โดยที่ argument vector เป็นลิสต์ของอาร์กิวเมนต์คำสั่ง ที่ใช้ร้องขอให้โปรแกรมที่กำลังทำงานอยู่นั้นรันต่อไป ด้วยชื่อของโปรแกรมเอง ส่วน environment vector เป็นลิสต์ของคู่ “ NAME=VALUE” ที่สัมพันธ์กับชื่อ environment vector กับค่าของต้นฉบับเดิม(arbitrary textual values) เป็นสิ่งแวดล้อมไม่อยู่ในหน่วยความจำ kernel แต่เก็บอยู่ในเนื้อที่ภายในของ user mode ของโปรเซสที่อยู่ชั้นบนสุดสแต็กของโปรเซส ทั้ง argument และenvironment vector ไม่มีการเปลี่ยนแปลงเมื่อมีการสร้างโปรเซสใหม่ โปรเซสสุดท้ายของโปรเซสจะรับช่วง environment จากโปรเซสแม่ อย่างไรก็ตาม environment ใหม่จะเซ็ตอัพใหม่ เมื่อมีการร้องขอโปรแกรมใหม่ เมื่อมีการเรียก execute โปรเซสจะต้องมี environment ให้โปรแกรมใหม่ kernel จะส่งผ่าน environment เหล่านี้ไปยังโปรแกรมใหม่โดยการเปลี่ยน environment เดิมของโปรเซส นอกนั้น kernel จะปล่อยให้ environment และ vector ของคำสั่งไว้เฉย ๆ การแปลความหมายจะเก็บไว้ในไลบรารี และแอปพิลเคชันของ user mode ในการส่งผ่านค่าตัวแปร environment จากโปรเซสหนึ่งไปยังอีกโปรเซสหนึ่ง และการรับช่วงของตัวแปลเหล่านี้ของโปรเซสลูกมีหลายวิธี ที่ส่งผ่านข้อมูลไปยังคอโพเนนต์ของซอฟต์แวร์ใน user node ตัวแปร environment ที่หลากหลายมีความหมายและสัมพันธ์กับสิ่งต่าง ๆ ของซอฟต์แวร์ เช่น ตัวแปร TERM จะเป็นชื่อประเภทของการเชื่อมต่อเทอร์มินอลไปยังล็อกอินเซสชัน ของผู้ใช้ มีหลายโปรแกรมที่ใช้ตัวแปรเหล่านี้ตัดสินวิธีการปฏิบัติในการแสดงผลของผู้ใช้ เช่น การย้ายเคอร์เซอร์ หรือ เลื่อนขอบเขตของข้อความ ในโปรแกรมที่สนับสนุนหลายภาษาจะใช้ตัวแปร LANG เพื่อตัดสินเลือกภาษาเพื่อแสดงข้อความ

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

Context ของโปรเซส

คุณสมบัติของโปรเซสทั้ง identity และ environment ปกติจะมีการเซ็ตอัพเมื่อมีการสร้างโปรเซสมา และจะไม่มีการเปลี่ยนแปลงจนกว่าโปรเซสนั้นจะออกไป โปรเซสอาจจะเลือกเพื่อเปลี่ยน identity ก็สามารถทำได้ หรือแม้แต่จะเปลี่ยน environment ก็ได้เช่นกัน สำหรับ context ของโปรเซสจะเป็นสถานะของการรันโปรแกรม ณ เวลาใดเวลาหนึ่งซึ่งมีการเปลี่ยนแปลงสม่ำเสมอ ส่วนประกอบ context ของโปรแกรมมีดังนี้

Scheduling context : สิ่งนี้เป็นส่วนสำคัญที่สุดของ context ของโปรเซส ซึ่ง Scheduling context เป็นข้อมูลที่ Scheduler ต้องการหยุด หรือ เริ่มโปรเซส ข้อมูลเหล่านี้รวมถึงสำเนาทั้งหมดของรีจีสเตอร์ของโปรเซส นอกจากนี้ Scheduling context ยังรวมข้อมูลเกี่ยวกับ Scheduling context และสัญญาณพิเศษที่จะส่งไปยังโปรเซสหลักของ Scheduling context คือ สแต็กของ kernel ซึ่งเป็นหน่วยความจำที่แยกต่างหาก ที่สงวนไว้ใช้เฉพาะโค้ดใน kernel mode ทั้ง system call และอินเทอร์รัพต์ที่เกิดขึ้นขณะที่โปรเซสกำลังเอ็กซิคิวต์จะใช้สแต็ก

Accounting : kernel จะรักษาข้อมูลเกี่ยวกับรีซอร์สที่ใช้ในแต่ละโปรเซส รวมทั้งรีซอร์สทั้งหมดที่ใช้ในโปรเซสนั้น

File table : File table เป็นอาร์เรย์ของพอยเตอร์โครงสร้างไฟล์ kernel เมื่อมีการสร้าง system call ที่เป็นไฟล์ อินพุต/เอาต์พุต โปรเซสอ้างอิงไฟล์โดยใช้อินเด็กซ์ของ table นี้

File-system context : ในขณะที่ File table จะเป็นลิสต์ของไฟล์ที่เปิดอยู่ ส่วน File-system context ใช้สำหรับการร้องขอเพื่อเปิดไฟล์ใหม่ ส่วนนี้จะเก็บไดเร็กทอรีราก และดีฟอลต์ไดทอรีที่ใช้ในการค้นหาไฟล์ใหม่

Signal-handler table : ใน UNIX สามารถส่งสัญญาณ asynchronous ไปยังโปรเซสเพื่อโต้ตอบอีเวนต์ภายนอก Signal-handler table กำหนดรูทีนในแอ็ดเดรสของโปรเซสที่ถูกเรียกสัญญาณที่กำหนดนั้นส่งมาถึง

Virtual-memory context : Virtual-memory context จะอธิบายสารบัญของแอ็กเดรสส่วนตัวของโปรเซส

กระบวนการตรวจสอบ

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ในการตรวจสอบสถานะของกระบวนการของคุณภายใต้ระบบปฏิบัติการยูนิกซ์ใช้คำสั่ง ps

ps [-option]

ที่ใช้โดยไม่มีตัวเลือกนี้ผลิตรายการของทุกกระบวนการที่เป็นของคุณและเกี่ยวข้องกับ terminal ของคุณ ข้อมูลที่แสดงโดยคำสั่ง ps แตกต่างกันไปตามตัวเลือกคำสั่งที่ (s) ที่คุณใช้และชนิดของยูนิกซ์ที่คุณกำลังใช้ เหล่านี้คือบางส่วนของหัวคอลัมน์ที่ปรากฏขึ้นโดยรุ่นที่แตกต่างกันของคำสั่งนี้

PID SZ (size in Kb) TTY (controlling terminal) TIME (used by CPU) COMMAND

ตัวอย่าง

1. เมื่อต้องการแสดงข้อมูลเกี่ยวกับกระบวนการของผู้ที่กำลังทำงานอยู่

% ps

2. เมื่อต้องการแสดงข้อมูลเกี่ยวกับกระบวนการทั้งหมดของคุณ

% ps -u Username

3. เมื่อต้องการสร้างรายการยาวของกระบวนการทั้งหมดที่กำลังทำงานอยู่

% ps –ly

การระบุหมายเลขของกระบวนการ

ชนิดข้อมูลที่แสดงถึงรหัส pid_t จะได้รับข้อมูลกระบวนการที่เป็นชนิดจำนวนเต็ม (int) คือProcess ID ของกระบวนการ โดยการเรียก getpid () ส่วนฟังก์ชัน getppid () จะส่งค่า Process ID ของ parent process (Process ID ของผู้ปกครอง) โปรแกรมควรจะรวมส่วนหัวของไฟล์ 'unistd.h' และ 'sys / types.h' เพื่อใช้ฟังก์ชันเหล่านี้

Function: pid_t getpid (void)

The **getpid()** function returns the process ID of the current process.

Function: pid_t getppid (void)

The **getppid()** function returns the process ID of the parent of the current process.

การสร้างกระบวนการหลายกระบวนการ

ฟังก์ชั่นที่เป็นพื้นฐานสำหรับการสร้างกระบวนการ จะมีการประกาศในส่วนหัวของไฟล์ "unistd.h"

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

Function: pid_t fork (void)

ถ้าการดำเนินการสำเร็จแล้วจะมี parent และกระบวนการที่ fork ทั้งสอง และเห็นผลลัพธุ์ ที่มีค่าที่แตกต่างกัน ถ้าส่งกลับค่ามาเป็น 0 ของ child process ID ในการ fork parent process ถ้าการสร้างกระบวนการที่ล้มเหลวจะส่งกลับค่าของ -1 ในการ parent process และ dki fork จะไม่มีการถูกสร้างขึ้น คุณลักษณะเฉพาะของกระบวนการ fork ที่แตกต่างจากการ parent คือ child process มี Process ID ของตัวเองที่ไม่ซ้ำกัน

ตัวอย่างที่ 1

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

เมื่อโปรแกรมนี้จะถูกดำเนินการก่อนจะพิมพ์ Hello World! . เมื่อ fork ถูกดำเนินการขั้นตอนเหมือนกันที่เรียกว่า child จะถูกสร้างขึ้น จากนั้นทั้งแม่และ child process เริ่มต้นดำเนินการที่คำสั่งต่อไป หมายเหตุต่อไปนี้

เมื่อ fork จะถูกดำเนินการทุกอย่างในการปกครองจะถูกคัดลอกไปที่กระบวนการเด็ก ซึ่งรวมถึงค่าตัวแปร, รหัสและอธิบายไฟล์ต่อไปนี้การแยกforkและกระบวนการปกครองเป็นอิสระอย่างสมบูรณ์

มีการรับประกันที่ดำเนินกระบวนการจะพิมพ์ผมกระบวนการแรกคือ กระบวนการ child process จะเริ่มดำเนินการที่คำสั่งในทันทีหลังจากที่ทางแยกที่ไม่จุดเริ่มต้นของโปรแกรม

สามารถแตกต่างไปจาก child process โดยการตรวจสอบ valueOf ตอบแทน Fork ส่งกลับเป็นศูนย์ให้กระบวนการ fork และกระบวนการ id จาก childprocess ไปยัง parent

กระบวนการสามารถดำเนิน Fork ให้มากที่สุดเท่าที่ต้องการ แต่ต้องระวังลูปอนันต์ของ Fork (มีจำนวนสูงสุดของกระบวนการอนุญาตให้ผู้ใช้คนเดียว)

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ตัวอย่างที่ 2

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

ตัวอย่างที่ 3 (Multiple forks):

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

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

เสร็จสิ้นกระบวนการ (Process Completion)

ฟังก์ชั่นที่อธิบายไว้ในส่วนนี้จะใช้ในการรอให้ child process ในการยุติหรือหยุดและตรวจสอบสถานะของมัน ฟังก์ชั่นเหล่านี้จะถูกประกาศในแฟ้มส่วนหัว "sys / wait.h"

Function: pid_t wait (int *status_ptr)

wait()จะบังคับให้การปกครองที่จะรอให้กระบวนการเด็กที่จะหยุดหรือยุติ wait () กลับ pid ของเด็กหรือ -1 สำหรับข้อผิดพลาด สถานะออกจากเด็กที่ถูกส่งกลับไป status_ptr

Function: void exit (int status)

exit() สิ้นสุดกระบวนการที่เรียกฟังก์ชันนี้และส่งกลับค่าสถานะออก ทั้ง UNIX และ C โปรแกรม (forked) สามารถอ่านค่าสถานะ

โดยการ convention สถานะของ 0 หมายถึงการเลิกจ้างตามปกติ ค่าอื่น ๆ บ่งชี้ข้อผิดพลาดหรือผิดปกติเกิดขึ้น หลายสายมาตรฐานห้องสมุดมีข้อผิดพลาดที่กำหนดไว้ในแฟ้มส่วนหัว sys / stat.h เราสามารถได้อย่างง่ายดายได้รับมาจัดประชุมของเราเอง

หาก child process จะต้องดำเนินการในการดำเนินการก่อนที่ยังคงรอคอยการเรียกระบบที่ใช้ การเรียกร้องให้ฟังก์ชันนี้ทำให้เกิดการที่จะรอจนกว่าจะเป็นหนึ่งในทางออกของกระบวนการ โทรรอส่งกลับ id กระบวนการของกระบวนการซึ่งจะช่วยให้สามารถที่จะรอให้กระบวนการโดยเฉพาะอย่างยิ่งที่จะเสร็จสิ้น

กระบวนการอาจระงับสำหรับระยะเวลาหนึ่งโดยใช้คำสั่ง sleep command

Function: unsigned int sleep (seconds)

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ตัวอย่างที่ 4

: Guarantees the child process will print its message before the parent process.

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

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ตัวอย่างที่ 5 :

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

Orphan processes

When a parent dies before its child, the child is automatically adopted by the original “init” process whose PID is 1. To, illustrate this insert a sleep statement into the child’s code. This ensured that the parent process terminated before its child.

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ตัวอย่างที่ 6

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

The output is:

```
I'am the original process with PID 5100 and PPID 5011.
I'am the parent process with PID 5100 and PPID 5011.
My child's PID is 5101
PID 5100 terminates. /* Parent dies */
I'am the child process with PID 5101 and PPID 1.
/* Orphaned, whose parent process is “init” with pid 1 */
PID 5101 terminates.
```

Zombie processes

กระบวนการที่ยุติไม่สามารถออกจากระบบจนกว่าของรับรหัสกลับของมัน อยู่แล้วตายแล้วมันจะได้รับการรับรองโดยกระบวนการ "init" ซึ่งมักจะยอมรับรหัสกลับ child ของ อย่างไรก็ตามหากของกระบวนการมีชีวิตอยู่ แต่ไม่เคยดำเนินการรอ (), รหัสกลับของกระบวนการจะไม่ได้รับการยอมรับและกระบวนการจะยังคงอยู่ผีดิบ

โปรแกรมต่อไปนี้สร้างกระบวนการผีดิบที่ถูกระบุในผลลัพธ์จากยูทิลิตี้ PS เมื่อการปกครองถูกฆ่าตายเด็กเป็นลูกบุญธรรมโดย "init" และอนุญาตให้พักผ่อนในความสงบ

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ตัวอย่างที่ 7

```
#include <stdio.h>
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 (100) ;           /* stop executing for 100 seconds */
        }
        else    /* pid is zero, so I must be the child */
        {
                exit (42) ;     /* exit with any number */
        }
}
```

The output is:

```
root@CentOS> a.out &            /* execute the program in the background */
[1] 5186
root@CentOS > ps                /* obtain process status  */
PID TT STAT TIME COMMAND
5187 p0 Z 0:00 <exiting> the zombie child process
5149 p0 S 0:01 -csh (csh) the shell
5186 p0 S 0:00 a.out the parent process
5188 p0 R 0:00 ps
root@CentOS > kill 5186         /* kill the parent process  */
[1] Terminated a.out
root@CentOS > ps                /* notice that the zombie is gone now  */
PID TT STAT TIME COMMAND
5149 p0 S 0:01 -csh (csh)
5189 p0 R 0:00 ps
```

***เพิ่มเติมโปรแกรม**

- ให้เพิ่มชื่อนักศึกษาเข้าไปในโปรแกรม
- ให้โปรแกรมทำงานแบบ background process
- กำหนดการแสดงผลโดยพิมพ์ชื่อนักศึกษาออกทางหน้าจอ ทุกๆ 10-15 วินาที

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

การทดลองที่ 4

ลำดับขั้นการทดลอง

ให้นักศึกษา Print Screen ขั้นตอนการทดลอง ด้วย พร้อมกับอธิบาย วิธีการทำ

1. Examples of Processes 1 ให้ทำการโปรแกรม fork1.c แล้วเปรียบกับ กับผลของการ RUN เหมือนกันหรือไม่

program code fork1.c ผลจากการรันจะต้องได้

program code

fork1.c

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

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

     char *who;
     int i;

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

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

     exit (0);
}
```

ให้เปลี่ยนเป็นชื่อนักศึกษา

fork returns twice!

program behavior

```
% fork1
*fork1: parent
*fork1: parent
*fork1: child
*fork1: parent
*fork1: child
*fork1: parent
*fork1: child
*fork1: parent
*fork1: child
*fork1: parent
*fork1: child
*fork1: child
```

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

2. การดำเนินการเป็น Asynchronous

program code

fork2.c

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

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

   int i;              /* loop counter */
   char *who;          /* name of process */
   int n;              /* seconds to sleep */

   if (fork ()) {
      who = "parent";
      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);
}
```

sleep n seconds

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

ผลจากการรันจะต้องได้

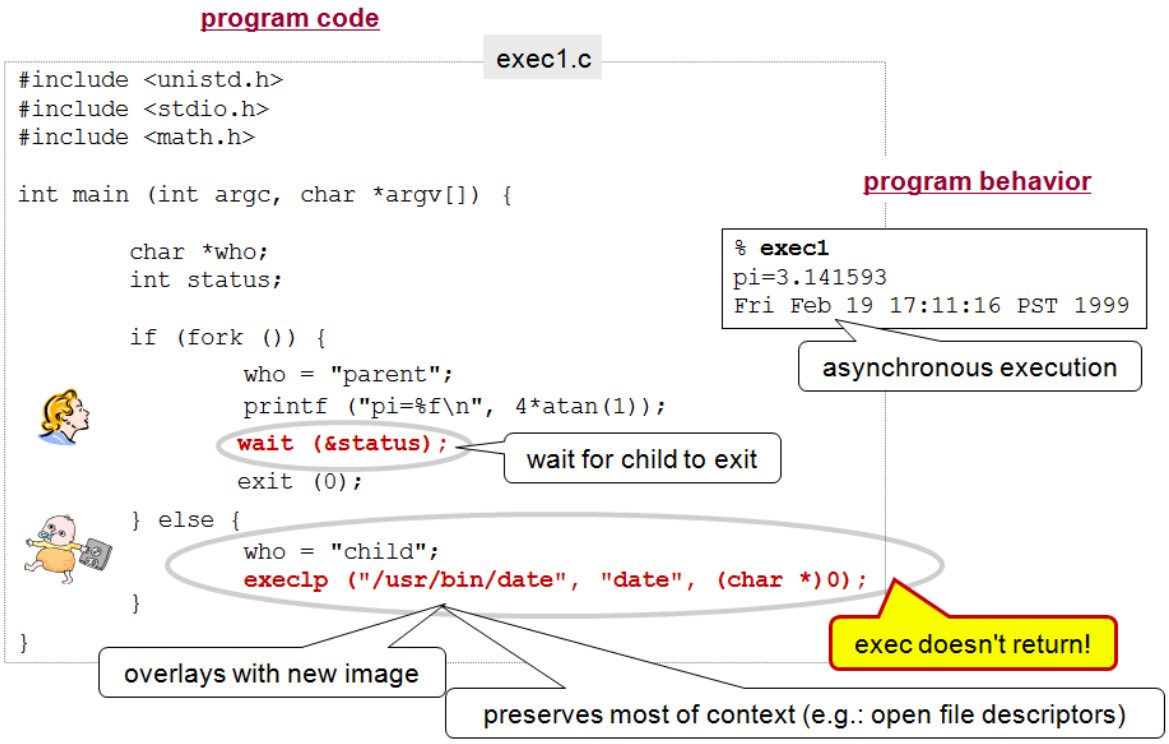
program behavior

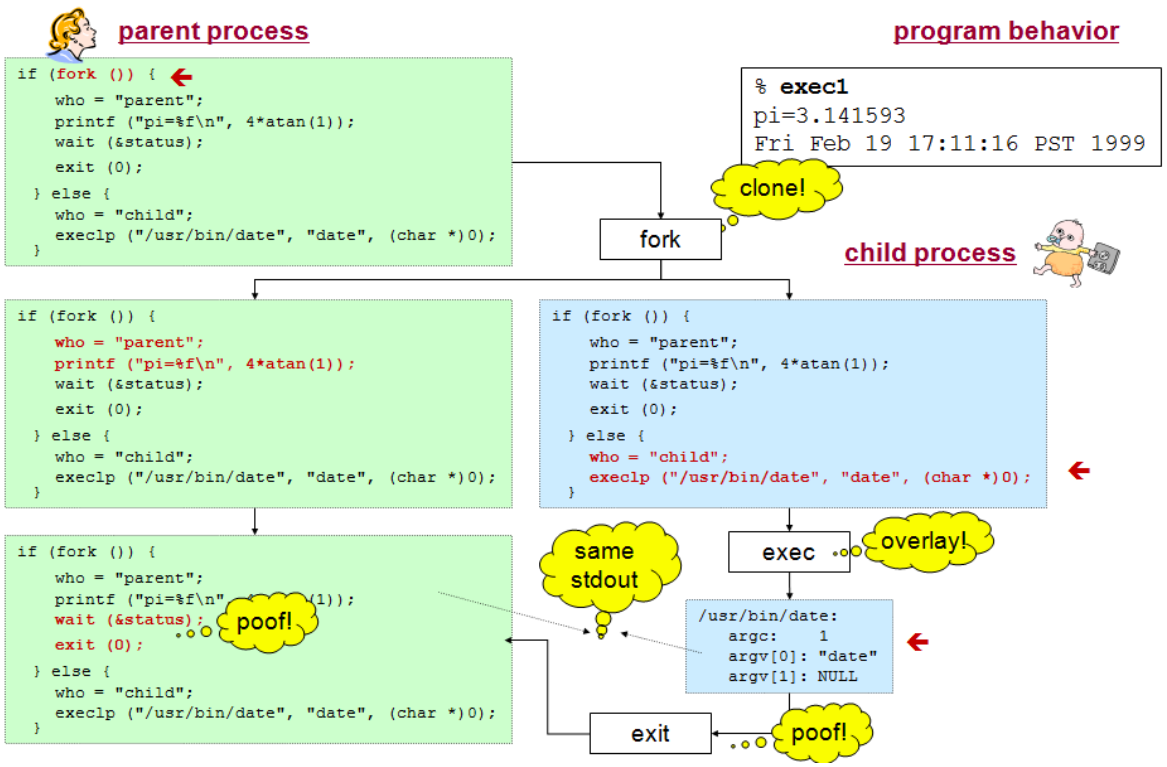
```
% fork2
* 1.  parent: my pid =  11597, ppid =   7125
* 1.   child: my pid =  10843, ppid =  11597
* 2.   child: my pid =  10843, ppid =  11597
* 2.  parent: my pid =  11597, ppid =   7125
* 3.   child: my pid =  10843, ppid =  11597
* 4.   child: my pid =  10843, ppid =  11597
* 3.  parent: my pid =  11597, ppid =   7125
* 5.   child: my pid =  10843, ppid =  11597
* 6.   child: my pid =  10843, ppid =  11597
* 4.  parent: my pid =  11597, ppid =   7125
* 7.   child: my pid =  10843, ppid =  11597
* 8.   child: my pid =  10843, ppid =  11597
* 5.  parent: my pid =  11597, ppid =   7125
* 9.   child: my pid =  10843, ppid =  11597
*10.   child: my pid =  10843, ppid =  11597
* 6.  parent: my pid =  11597, ppid =   7125
* 7.  parent: my pid =  11597, ppid =   7125
* 8.  parent: my pid =  11597, ppid =   7125
* 9.  parent: my pid =  11597, ppid =   7125
*10.  parent: my pid =  11597, ppid =   7125
```

unpredictable interleaving

การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

3. You Can Exec a Program





การสอนครั้งที่ 4	ใบงานที่ 3	รหัสวิชา TEDEE305
หน่วยที่ 3 การแบ่งปันทรัพยากร		เวลา 3 ชั่วโมง
เรื่อง การจัดการโปรเซส		

5. You Can Also Exec a Script

program code

exec2.c

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

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

        char *who;
        int status;

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

program behavior

```
% exec2
pi= 3.141593
*my-script: argv[1]=a argv[2]=b
```

pathname of script

which shell to use

/bin/my-script

```
#! /bin/ksh
echo "*my-script: argv[1]=$1 argv[2]=$2"
```

การเขียนใบงานการทดลอง

1. เขียนอธิบายขั้นตอนการทำงานโปรแกรม พร้อม แคปเจอร์ รูปภาพขั้นตอนการทำงาน สรุปผลการทำงานในแต่ละหัวข้อของการทดลอง
2. ทำเป็นเอกสารใบงานบันทึกงานเป็นไฟล์ Word และปริ้นส่ง
3. จากการทดลอง มีข้อแตกต่างกันอย่างไรบ้าง จงอธิบาย
4. สรุปผลการทดลอง

.....

.....

.....

.....