ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

วิชา 01076115 ปฏิบัติการองค์ประกอบคอมพิวเตอร์

ภาคการศึกษาที่ 2/2565

กลุ่ม _____

การทดลองที่ 7 System Call

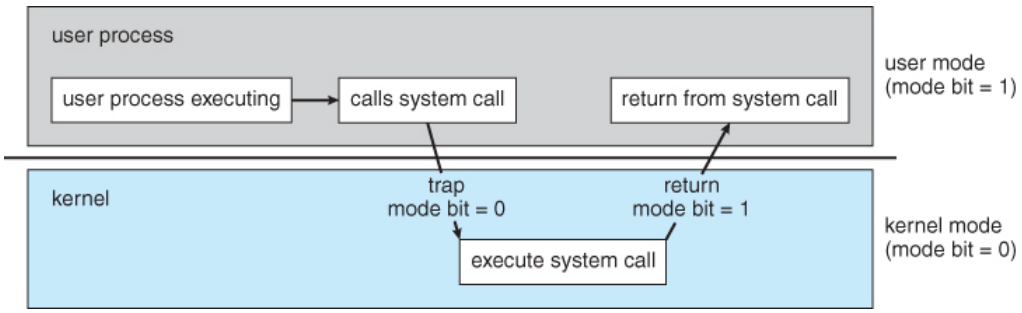
1. System Call

หน้าที่อย่างหนึ่งของระบบปฏิบัติการ คือ การสร้างบริการพื้นฐานในการติดต่อกับ Hardware เนื่องจากระบบปฏิบัติการในปัจจุบัน มักจะไม่ยอมให้โปรแกรมติดต่อกับ Hardware โดยตรง ทั้งนี้เพื่อป้องกันปัญหาเรื่องการแย่งใช้ Hardware กันระหว่างโปรแกรม และเพื่อรักษาความปลอดภัยให้กับระบบปฏิบัติการเอง

การที่ระบบปฏิบัติการให้บริการพื้นฐานนี้ ก็ช่วยให้การเขียนโปรแกรมง่ายขึ้นอีกด้วย เพราะผู้เขียนโปรแกรมไม่จำเป็นจะต้องลงไปศึกษารายละเอียดในระดับ Hardware เพียงแค่เรียก System Call ก็จะได้การทำงานตามที่ต้องการ ซึ่งที่ผ่านมาก็ได้ทดลองใช้ไปบางส่วน เช่น การแสดงผล การรับคีย์บอร์ด

ระบบปฏิบัติการตระกูล UNIX รวมถึง Macintosh (ซึ่งใช้ Kernel เป็น Unix) จะใช้ System Call แบบเดียวกันนี้ สำหรับระบบปฏิบัติการ Windows อาจจะต่างออกไปบ้าง แต่ก็ไม่ต่างกันมากนัก เนื่องจากการออกแบบระบบปฏิบัติการ Windows ก็ได้รับอิทธิพลไปจากระบบปฏิบัติการ UNIX อยู่หลายส่วน

สำหรับการทำงานของ System Call พื้นฐาน แสดงตามรูปด้านล่าง โดยเมื่อมีการเรียก System Call จะเกิดการ Trap เพื่อเปลี่ยนโหมดการทำงานจาก User Process ไปสู่โหมดการทำงานของระบบปฏิบัติการเอง (เรียกว่า Kernel Mode) ทั้งนี้เนื่องจากการติดต่อในระดับ Hardware ต้องใช้ Privilege ระดับ Kernel จากนั้นระบบปฏิบัติการจะทำงานใน System Call ที่เรียกไป เช่น ให้รับคีย์บอร์ด ให้แสดงผลบนจอภาพ เมื่อเสร็จจึง return เข้าสู่ User Process เพื่อทำงานต่อไป



ในระบบปฏิบัติการ UNIX จะมี System Call จำนวนมาก โดยหากต้องการดูรายละเอียดสามารถดูได้จาก http://www.tutorialspoint.com/unix_system_calls/

สำหรับในการทดลองนี้ ไม่สามารถนำ System Call ทั้งหมดมาทดลองได้ แต่จะนำเฉพาะส่วนของ I/O หรือ File I/O มาแนะนำ โดยให้ศึกษาจากโปรแกรมต่อไปนี้

```
   /* -- create.s -- */

         .text
         .global _start
   _start:
         push  {r4, lr}

/* Open (Create) File      */
         ldr   r0, =newfile
         mov   r1, #0x42         @ create R/W
         mov   r2, #384          @ = 600 (octal)
         mov   r7, #5            @ open (create)
         svc   0

         cmp   r0, #-1           @ open error?
         beq   err

         mov   r4, r0            @ save file descriptor

/* Show Input String */
         mov   r0, #1            @ stdout - 1 = monitor
         ldr   r1, =input        @ input string
         mov   r2, #(ip_end-input)     @ len
         mov   r7, #4
         svc   0

/* Read Input String */
         mov   r0, #0            @ stdin - 0 = keyboard
         ldr   r1, =buffer       @ address of input buffer
         mov   r2, #26           @ max. len. of input
         mov   r7, #3            @ read
         svc   0
         mov   r5, r0            @ save no. of character

/* Write to File */
         mov   r0, r4            @ file descriptor
         ldr   r1, =buffer       @ address of buffer to write
         mov   r2, r5            @ length of data to write
         mov   r7, #4
         svc   0

/* Close File */
         mov   r7, #6            @ close
         svc   0
         mov   r0, r4            @ return file descriptor

exit:    pop   {r4, lr}
         mov   r7, #1            @ exit
         svc   0

err:     mov   r4, r0
         mov   r0, #1
         ldr   r1, =errmsg
         mov   r2, #(errmsgend-errmsg)
```

```
        mov   r7, #4
        svc   0

        mov   r0, r4
        b exit

        .data
errmsg:  .asciz      "create failed"
errmsgend:

newfile: .asciz      "/home/pi/newfile"
input:   .asciz      "Input a string: \n"
ip_end:
buffer:  .byte       100
```

ในส่วน **Open (Create) File** จะเป็นการสร้างไฟล์ใหม่ โดยใช้ system call #5 โดยมีพารามิเตอร์ดังนี้

r0 : ชื่อไฟล์ที่จะเปิดหรือสร้าง

r1 : โหมดของการทำงาน 0=read only, 1=write only, 2=R/W, 0x40=create (if not exist)
ดังนั้น 0x42 คือ สร้างแบบ R/W

r2 : permission bit คือ ค่า permission ของไฟล์ที่จะสร้างขึ้นตามแบบ UNIX (rwx) ฐาน 8
โดยค่า 600 ในโปรแกรมจะหมายถึง rw- --- ---

เมื่อเรียก system call นี้ หากมี error จะ return -1 มาที่ r0 แต่ถ้าไม่มี error จะ return หมายเลขของ file descriptor กลับคืนมา โดย file descriptor จะเป็นโครงสร้างข้อมูลที่อยู่ภายในระบบปฏิบัติการ จะเก็บรายละเอียดของไฟล์เอาไว้ เช่น ตำแหน่งที่อยู่ของไฟล์ ตำแหน่งที่กำลังอ่านไฟล์ ฯลฯ

ในส่วน Show Input String และ Read Input String จะไม่อธิบาย เนื่องจากเคยใช้มาแล้ว โดยทั้งสองส่วนจะทำหน้าที่รับข้อมูลจากคีย์บอร์ด โดยเก็บไว้ในหน่วยความจำ address buffer

สำหรับ ส่วน Write to File จะ write ข้อมูลใน buffer ลงในไฟล์ โดยใช้ system call #4 โดยมีพารามิเตอร์ดังนี้

r0 : หมายเลข ของ file descriptor

r1 : ตำแหน่งของข้อมูลที่จะนำไปเขียนลงไฟล์

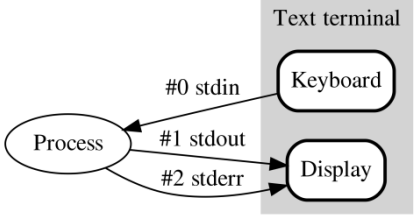
r2 : ความยาวข้อมูลที่จะเขียน

หากสังเกตุให้ดี จะเห็นว่าในส่วนนี้ จะคล้ายกับส่วนของการ Write String ออกทางจอภาพ โดยเรียก Function Call เดียวกัน ที่เป็นเช่นนี้ เนื่องจาก Concept ของ UNIX คือ มองทุกอย่างเป็น File นั่นคือ มองว่า Monitor เป็นไฟล์หนึ่ง โดยไฟล์ของระบบที่ไม่ต้องเปิดหรือปิดก็สามารถเขียนได้เลยนั้นประกอบด้วย

standard input หมายเลข 0

standard output หมายเลข 1

standard error หมายเลข 2



ดังนั้นการรับคีย์บอร์ดและแสดงผลจะใช้เหมือนกับการอ่านเขียนไฟล์

สำหรับส่วน /* Close File */ ใช้ system call #6 เพื่อปิดไฟล์ โดย r0 จะเก็บเลขของ file descriptor ที่ต้องการปิด

คำสั่ง

ให้นำไฟล์ข้างต้นมาทดลองรัน แล้วดูเนื้อหาในไฟล์ที่สร้างขึ้น (ชื่อ new file)

Function Call ข้างต้นนั้น ในภาษาซี ก็มีการนำมาสร้างเป็นภาษา C ซึ่งมีรูปแบบพารามิเตอร์คล้ายๆ กันดังนี้

```
int open (char* full_path_name, int flags, int permissions);
int write (int file_descriptor, char* buffer, int length);
int read (int file_descriptor, char* buffer, int length);
void close (int file_descriptor);
```

ยังมี Function Call ที่น่าสนใจเกี่ยวกับไฟล์อีกอันหนึ่ง คือ lseek ซึ่งจะทำหน้าที่เลื่อน file pointer ไปตำแหน่งที่ต้องการ เพื่อจะได้อ่านหรือเขียนในตำแหน่งที่ต้องการ โดยมีการใช้งานดังนี้

r0 : file descriptor

r1 : offset หมายถึงตำแหน่งที่จะเลื่อนไป

r2 : mode มีทั้งหมด 3 โหมด คือ 0=seek_set หมายถึงอ้างอิงจากตำแหน่งต้นไฟล์ 1=seek_cur หมายถึงตำแหน่งจะอ้างอิงจากตำแหน่งล่าสุดของไฟล์ และ 2=seek_end หมายถึงอ้างอิงจากตำแหน่งท้ายสุดของไฟล์

r7 : เป็นค่า 19 หมายถึง lseek

คำสั่ง

ให้ทดลองป้อนโปรแกรมต่อไปนี้ แทรกเข้าไปตำแหน่งหลังจาก write file แต่ก่อน close file

```
* lseek */
        mov   r0, r4            @ file descriptor
        mov   r1, #3            @ position
        mov   r2, #0            @ seek_set
        mov   r7, #19
        svc   0

/* Write to File */
        mov   r0, r4            @ file descriptor
        ldr   r1, =test         @ address of buffer to write
        mov   r2, #4            @ length of data to write
        mov   r7, #4
        svc   0
```

โดยกำหนดข้อมูล

```
test:    .asciz      "abcd"
```

ให้ทดลองรันและดูผลการทำงาน

การส่ง Lab 7

1. ทดลอง run โค้ด create file 2 ครั้ง capture หน้าจอส่ง
2. เพิ่มโค้ด lseek รัน 2 ครั้ง capture หน้าจอส่ง
3. ภายใน 17 เม.ย.

Assignment 1

ให้เขียนโปรแกรมหนึ่งมี 3 ฟังก์ชัน

1. ฟังก์ชันที่ 1 **รับข้อมูล** ดังต่อไปนี้เข้าไปเก็บในไฟล์
   - รหัสนักศึกษา มีขนาด 8 ตัวอักษร
   - ชื่อนักศึกษา (ภาษาอังกฤษ) ความยาว ไม่เกิน 80 ตัวอักษร
   - หลังจากรับ 1 คนให้ปิดไฟล์ เพื่อป้องกันข้อมูลหาย
   - หากป้อนคนที่ 2 ให้ต่อจากคนแรก
2. หาก cat ไฟล์ จะต้องเห็น 1 คนต่อ 1 บรรทัด
3. ฟังก์ชันที่ 2 ค้นหาข้อมูล โดยรับรหัสนักศึกษา จากนั้นให้แสดงชื่อของนักศึกษาออกมา
4. ฟังก์ชันที่ 3 ให้ออกจากโปรแกรม
5. ส่งเป็นรายงานมี code, อธิบายแนวคิดของแต่ละฟังก์ชัน และผลการทำงานแยกตามฟังก์ชัน
6. ภายใน 24 เม.ย.

คำแนะนำ : ให้เก็บข้อมูลแบบ Fix size ในแต่ละบรรทัด เพื่อที่จะใช้ lseek ไปที่ต้นบรรทัดของแต่ละบรรทัดได้

[option] สามารถเลือกทำเพิ่มเติมได้

7. ฟังก์ชันที่ 4 ให้ลบข้อมูลนักศึกษาบางคนได้ โดยรับรหัสนักศึกษา โดยเมื่อลบแล้วเมื่อ cat file จะไม่พบรายชื่อนักศึกษาคนนั้นอีก และไม่มีบรรทัดใดที่ว่าง