

# Activity 4 : Simple Shell

## วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจหลักการทำงานของ shell
2. เพื่อให้นิสิตเขียนโปรแกรม shell อย่างง่าย
3. เพื่อให้นิสิตใช้ Linux System call ที่เกี่ยวข้องกับการทำงานของ shell เช่น fork(), exec()

## ความรู้เบื้องต้น

เชลล์ (Shell) คือโปรแกรมประเภท Command Interpreter หรือ Command Line Interface ที่ติดต่อผู้ใช้ในรูปแบบตัวอักษร (text mode) โดยเชลล์จะรับคำสั่งทางคีย์บอร์ดแล้วถ่ายทอดคำสั่งนั้นไปยังระบบปฏิบัติการหรือเรียกโปรแกรมประยุกต์ให้ทำงาน

เนื่องจากเชลล์เป็นช่องทางพื้นฐานสำหรับสั่งให้คอมพิวเตอร์ทำงาน เชลล์จึงเป็น System program ที่ถูกติดตั้งมาพร้อมระบบปฏิบัติการส่วนใหญ่ เช่น Microsoft Windows มีโปรแกรม Command Prompt (cmd.exe) ส่วน Unix/Linux มีเชลล์หลายตัวให้เลือกใช้ เช่น sh, csh, ksh, bash ฯลฯ แต่ตัวที่เป็นที่นิยมคือ bash

เชลล์ส่วนใหญ่นอกจากจะมีความสามารถในการสั่งให้โปรแกรมทำงานแล้ว ยังมี ความสามารถอื่นๆ ที่ช่วยให้ผู้ใช้ทำงานได้อย่างมีประสิทธิภาพขึ้น เช่น Command history, Command autocomplete, input/output redirection, pipe, background/foreground process control, shell environment, shell scripting ฯลฯ เชลล์แต่ละตัวมีความสามารถหรือวิธีใช้งานบางอย่างแตกต่างกันไป

การทำงานโดยทั่วไปของเชลล์คือการวนลูปรับคำสั่งที่ละบรรทัด หากคำสั่งที่ได้รับเป็นการเรียกโปรแกรมขึ้นมาทำงาน เชลล์ก็จะสร้างโปรเซสใหม่ขึ้นมาโดยใช้ fork() และให้ child process ที่ถูกสร้างขึ้นใช้ exec() เพื่อ เรียกโปรแกรมที่ระบุในคำสั่งมาทำงานแทนเชลล์ ในขณะที่ shell ซึ่งเป็น parent process จะใช้ wait() เพื่อ รอให้โปรแกรมนั้นทำงานเสร็จ แล้วจึงวนกลับไปรับคำสั่งใหม่

ฟังก์ชันที่ใช้เรียกโปรแกรมมาทำงาน (execute) มีอยู่หลายตัวและมีชื่อขึ้นต้นด้วย exec ซึ่งได้แก่ execl(), execv(), execlp(), execvp(), execl() และ execve() ฟังก์ชันเหล่านี้แตกต่างกัน ที่พารามิเตอร์ที่รับตัวอย่างเช่น

```
int execvp(const char *file, char *const argv[]);  
int execl (const char *path, const char *arg, ..., char * const envp  
[]);
```

พารามิเตอร์ที่แต่ละฟังก์ชันต้องการสามารถสังเกตได้จากชื่อ ดังนี้

- ถ้ามีอักษร l (แอล) ได้แก่ execl(), execlp(), execl() จะรับพารามิเตอร์ของโปรแกรมที่จะเรียกทำงาน เป็น list
- ถ้ามีอักษร v ได้แก่ execv(), execvp(), execve () จะรับพารามิเตอร์ของโปรแกรมที่จะเรียกทำงานเป็นเวกเตอร์หรืออาร์เรย์นั่นเอง
- ถ้ามีอักษร p ได้แก่ execlp() และ execvp() จะค้นหาชื่อของโปรแกรมในรายการของไคเรททอรีที่ระบุโดย environment variable ชื่อ PATH (ถ้าไม่มี p ก็ถือว่าชื่อที่ให้มามี path ที่ถูกต้องอยู่แล้ว)
- ถ้ามีอักษร e ได้แก่ execl() และ execve() จะรับพารามิเตอร์เพิ่มอีกตัวเป็น environment ของโปรแกรมที่จะ execute (ถ้าไม่มี e ก็จะใช้ environment เหมือนโปรเซสเดิม)

นอกจากจะรับชื่อโปรแกรมเพื่อนำไป execute แล้ว เशलล์ยังมีคำสั่งที่เशलล์รับไปทำเองโดยไม่ต้องเรียกใช้ โปรแกรมอื่นและไม่ต้องสร้างโปรเซสใหม่ด้วย เรียกว่า คำสั่งภายใน (internal command)

## กิจกรรม

### 1. โปรแกรมข้างล่างแสดงการใช้ fork() และ execvp()

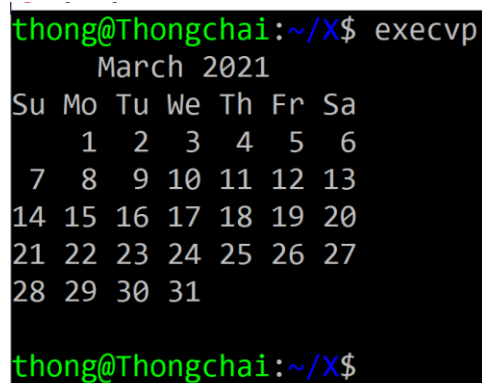
โปรแกรมสร้างปฏิทินปฏิทิน จากนั้นแม่คอยให้ลูกเรียก execvp() ทำงาน cal 3 2021

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

int main()
{
    pid_t pid;
    char *av[] = {"cal", "3", "2021", (char *)0};

    pid = fork() ;
    if (pid < 0) {
        printf("Error : cannot fork\n");
        exit(1);
    }
    else if (pid == 0) {
        execvp("cal", av);
    }
    else {
        wait(NULL);
        return(0);
    }
}
```

เมื่อ run โปรแกรมนี้ จะได้ผลลัพธ์ตามรูปข้างล่าง



```
thong@Thongchai:~/X$ execvp
March 2021
Su Mo Tu We Th Fr Sa
   1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

thong@Thongchai:~/X$
```

ขอให้นิสิตดัดแปลงโปรแกรมข้างบนให้รับ argument เป็นคำสั่ง LINUX แล้วทำงานคำสั่ง LINUX ตามที่ป้อน จากนั้นหยุดทำงาน

ถ้าผู้ใช้ไม่ป้อน argument โปรแกรมจะบอกให้ผู้ใช้ป้อนคำสั่งและหยุดทำงาน

ข้างล่างแสดงตัวอย่างการทำงานของโปรแกรมตามข้อกำหนด

```
thong@Thongchai: ~/Activity3_2021
thong@Thongchai:~/Activity3_2021$ sol1
Please enter UNIX command
thong@Thongchai:~/Activity3_2021$ sol1 date
Sun 07 Feb 2021 10:46:05 AM +07
thong@Thongchai:~/Activity3_2021$ sol1 head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
thong@Thongchai:~/Activity3_2021$ sol1 cal 12 2021
    December 2021
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

## 2. พิจารณาโปรแกรมข้างล่าง

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

int main()
{
    int    run = 1;

    while(run) {
        printf("mysh >");

        /*
        After reading user input, do these steps
        1. use tokenize() function to get command
        2. fork a child process
        3. child use execvp() to run command
        4. parent call wait() until user enter "exit"
        */

    }
}

/*
Split input string into substrings (called tokens)
which are sequences of contiguous characters separated by any of the
characters

in the string of accepted delimiters and fill the tokens into an
array.

The last element of the array contains a NULL pointer.
Return number of tokens or -1 if not success.

Example:
char delim[] = " \t\n";
char **tokens;
char string[256];
int numtokens;
int i;

fgets(string, 256, stdin);
numtokens = tokenize(string, delim, &tokens);
for (i = 0; i < numtokens; i++) {
    printf("%d:%s\n", i, tokens[i]);
}
```

```

*/

int tokenize(char *string, char *delimiters, char ***arrayOfTokens)
{
    char *token;
    int numtokens;
    int i;

    /* skip the beginning delimiters */
    string += strspn(string, delimiters);
    if ((token = malloc(strlen(string) + 1)) == NULL)
        return -1;

    /* count tokens */
    strcpy(token, string);
    numtokens = 0;
    if (strtok(token, delimiters) != NULL)
        for (numtokens = 1; strtok(NULL, delimiters) != NULL;
            numtokens++);

    /* create array of pointers to tokens */
    if ((*arrayOfTokens = malloc((numtokens+1)*sizeof(char *))) ==
        NULL) {
        free(token);
        return -1;
    }

    /* fill pointers to tokens into the array */
    if (numtokens == 0)
        free(token);
    else {
        strcpy(token, string);
        (*arrayOfTokens)[0] = strtok(token, delimiters);
        for (i = 1; i < numtokens; i++)
            (*arrayOfTokens)[i] = strtok(NULL, delimiters);
        (*arrayOfTokens)[numtokens] = NULL;
    }

    return numtokens;
}

```

ใส่โค้ดใน main() เพื่อให้ทำงานเป็น shell แบบง่าย โดยที่ shell fork โปรแกรมลูกเพื่อให้ลูกทำงานคำสั่ง  
 LINUX ตามที่ผู้ใช้ป้อน และแม่ fork ลูกคอยรับคำสั่งใหม่เรื่อย ๆ จนกระทั่งผู้ใช้ป้อน exit จึงจะหยุด  
 ทำงาน

Hint : ใช้ fgets() และฟังก์ชัน tokenize() เพื่อรับคำสั่งจากแป้นพิมพ์

ข้างล่างแสดงตัวอย่างการทำงานของโปรแกรมตามข้อกำหนด

```
thong@Thongchai: ~/Activity3_2021
thong@Thongchai:~/Activity3_2021$ sol2
mysh >date
Sun 07 Feb 2021 11:06:55 AM +07
mysh >cal 12 2021
    December 2021
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

mysh >exit
thong@Thongchai:~/Activity3_2021$ mysh >
```