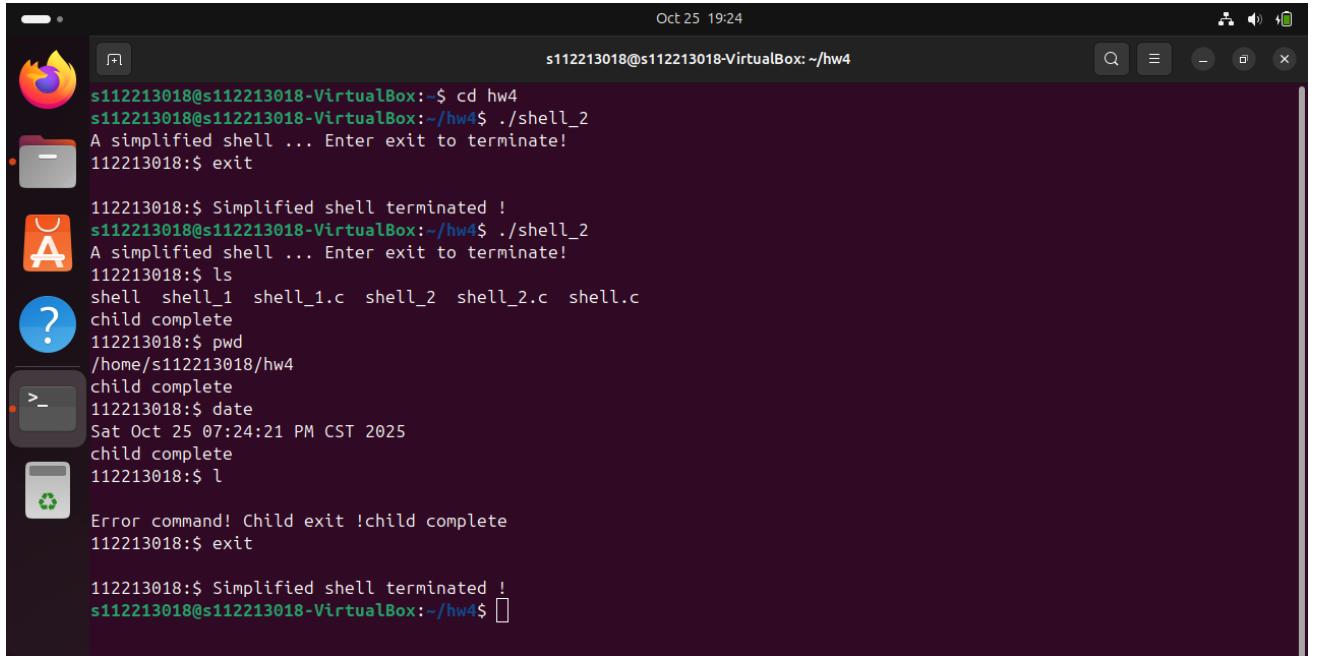


# Implement a simplified shell

## 一、執行結果

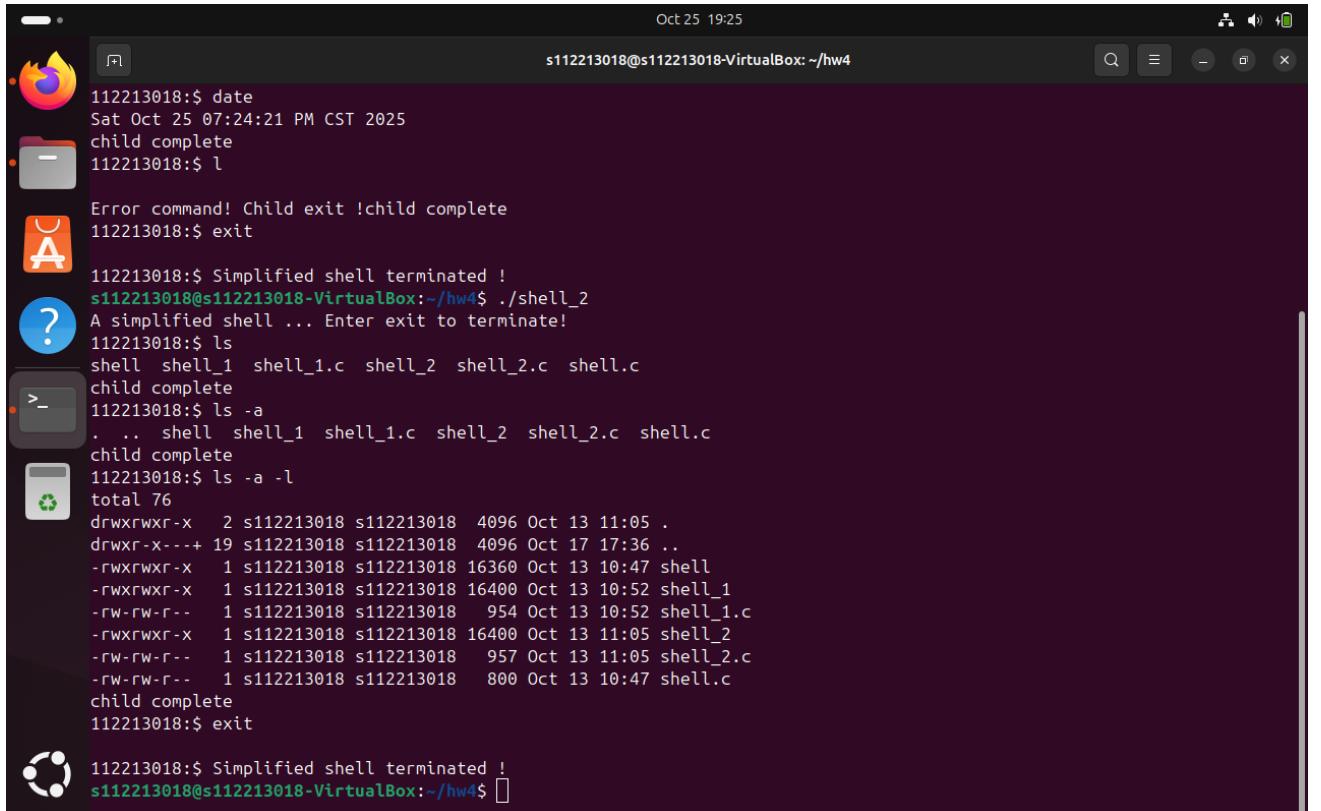


```
Oct 25 19:24
s112213018@s112213018-VirtualBox:~/hw4$ cd hw4
s112213018@s112213018-VirtualBox:~/hw4$ ./shell_2
A simplified shell ... Enter exit to terminate!
112213018:$ exit

112213018:$ Simplified shell terminated !
s112213018@s112213018-VirtualBox:~/hw4$ ./shell_2
A simplified shell ... Enter exit to terminate!
112213018:$ ls
shell shell_1 shell_1.c shell_2 shell_2.c shell.c
child complete
112213018:$ pwd
/home/s112213018/hw4
child complete
112213018:$ date
Sat Oct 25 07:24:21 PM CST 2025
child complete
112213018:$ l
112213018:$
Error command! Child exit !child complete
112213018:$ exit

112213018:$ Simplified shell terminated !
s112213018@s112213018-VirtualBox:~/hw4$ 
```

(圖一) 使用範例指令的執行結果



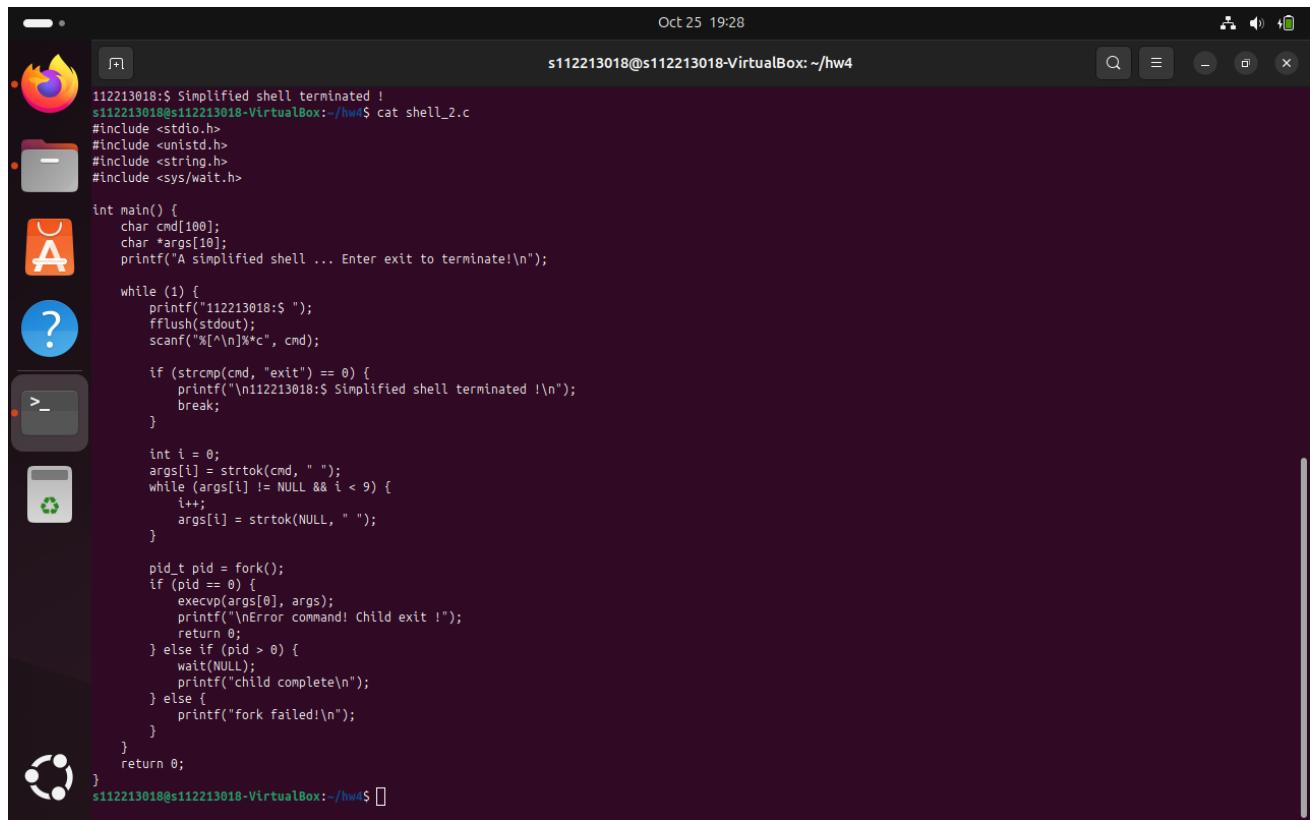
```
Oct 25 19:25
s112213018@s112213018-VirtualBox:~/hw4$ date
Sat Oct 25 07:24:21 PM CST 2025
child complete
112213018:$ l
112213018:$
Error command! Child exit !child complete
112213018:$ exit

112213018:$ Simplified shell terminated !
s112213018@s112213018-VirtualBox:~/hw4$ ./shell_2
A simplified shell ... Enter exit to terminate!
112213018:$ ls
shell shell_1 shell_1.c shell_2 shell_2.c shell.c
child complete
112213018:$ ls -a
. .. shell shell_1 shell_1.c shell_2 shell_2.c shell.c
child complete
112213018:$ ls -a -l
total 76
drwxrwxr-x  2 s112213018 s112213018  4096 Oct 13 11:05 .
drwxr-x---+ 19 s112213018 s112213018  4096 Oct 17 17:36 ..
-rw-rw-r--x  1 s112213018 s112213018 16360 Oct 13 10:47 shell
-rw-rw-r--x  1 s112213018 s112213018 16400 Oct 13 10:52 shell_1
-rw-rw-r--  1 s112213018 s112213018   954 Oct 13 10:52 shell_1.c
-rw-rw-r--x  1 s112213018 s112213018 16400 Oct 13 11:05 shell_2
-rw-rw-r--  1 s112213018 s112213018   957 Oct 13 11:05 shell_2.c
-rw-rw-r--  1 s112213018 s112213018   800 Oct 13 10:47 shell.c
child complete
112213018:$ exit

112213018:$ Simplified shell terminated !
s112213018@s112213018-VirtualBox:~/hw4$ 
```

(圖二) 使用不同個數的參數的執行結果

## 二、 程式碼介紹



The screenshot shows a terminal window titled "s112213018@s112213018-VirtualBox: ~/hw4". The terminal displays the following C code:

```
Oct 25 19:28
s112213018$ Simplified shell terminated !
s112213018@s112213018-VirtualBox:~/hw4$ cat shell_2.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main() {
    char cmd[100];
    char *args[10];
    printf("A simplified shell ... Enter exit to terminate!\n");

    while (1) {
        printf("112213018:$ ");
        fflush(stdout);
        scanf("%[^\\n]*c", cmd);

        if (strcmp(cmd, "exit") == 0) {
            printf("\n112213018:$ Simplified shell terminated !\n");
            break;
        }

        int i = 0;
        args[i] = strtok(cmd, " ");
        while (args[i] != NULL && i < 9) {
            i++;
            args[i] = strtok(NULL, " ");
        }

        pid_t pid = fork();
        if (pid == 0) {
            execvp(args[0], args);
            printf("\nError command! Child exit !");
            return 0;
        } else if (pid > 0) {
            wait(NULL);
            printf("child complete\n");
        } else {
            printf("fork failed!\n");
        }
    }
    return 0;
}

s112213018@s112213018-VirtualBox:~/hw4$
```

(圖三) 完整程式碼

1. `char cmd[100];` 是存放使用者輸入的內容，包含指令和參數。
2. `char *args[10];` 是建立一個陣列，存放字串的位置，這裡設定成 10 個位置，但最後一個要留給 `NULL`，所以總共能輸入一個指令加 8 個參數。
3. `while (1){`

```
    printf("112213018:$ ");
    fflush(stdout);
```

利用無限迴圈做互動式的 shell，然後強制把提示字立即顯示出來，避免因為沒有換行被系統緩衝住。

4. `scanf("%[^\\n]*c", cmd);` 讀取輸入的整行文字，包含空白，直到按下 `Enter` 為止。然後清掉那個換行符號，不留在緩衝區裡。
5. `if (strcmp(cmd, "exit") == 0) {`

```
    printf("\n112213018:$ Simplified shell terminated !\n");
    break;
```

如果讀到的字串是 `exit`，就用 `break` 結束迴圈。

```
6. int i = 0;  
    args[i] = strtok(cmd, " ");  
    while (args[i] != NULL && i < 9) {  
        i++;  
        args[i] = strtok(NULL, " ");  
    }
```

用 `strtok()` 把輸入的整行字串按照空白切開。每次切出一個詞就放進 `args[i]` 裡，最後會自動用 `NULL` 作為結尾。

```
7. pid_t pid = fork();  
if (pid == 0) {  
    execvp(args[0], args);  
    printf("\nError command! Child exit!");  
    return 0;  
}  
else if (pid > 0) {  
    wait(NULL);  
    printf("child complete\n");  
}  
else {  
    printf("fork failed!\n");  
}  
return 0;
```

先分成子行程和父行程，子行程負責執行 `execvp` 指令（`execvp()` 是用來執行使用者輸入的指令，並用這些資訊去執行外部程式），成功的話就會變成那個程式，失敗就會印出錯誤訊息並結束子行程。而父行程剛開始是 `wait(NULL)`，就是等子行程結束再開始，等 `wait()` 回傳後，父行程印出 `child complete`，代表這一輪指令執行結束。如果 `fork()` 失敗，就會印出 `fork failed!`。