

手寫

3.1

Ans: PARENT: value = 5

Figure 3.30

```

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
    pid_t pid;

    pid = fork();
    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d", value); /* LINE A */
        return 0;
    }
}

```

→ 分成 2 個 process  
 →  $5 + 15 = 20$   
 → child process 結束  
 → parent 和 child process 各自執行，不共享記憶體  
 所以 value 不會互相影響，印出 5

3.2

Ans: 8

Figure 3.31

```

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

int main()
{
    /* fork a child process */
    fork();
    /* fork another child process */
    fork();
    /* and fork another */
    fork();
    return 0;
}

```

每次 fork 會讓目前存在的  
 每個 process 產生一個新的  
 child process.

$f_0$   $\underline{\text{fork()}}$  →  $1 \times 2 = 2$   
 $f_1$   $\underline{\text{fork()}}$  →  $2 \times 2 = 4$   
 $f_2$   $\underline{\text{fork()}}$  →  $4 \times 2 = 8$

3.13

3.13      (A) 0      (C) 2603  
 Ans:      (B) 2603      (D) 2600

Figure 3.34

```

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred (A)
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d", pid); /* A */
        printf("child: pid1 = %d", pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d", pid); /* C */
        printf("parent: pid1 = %d", pid1); /* D */
        wait(NULL);
    }

    return 0;
}

```

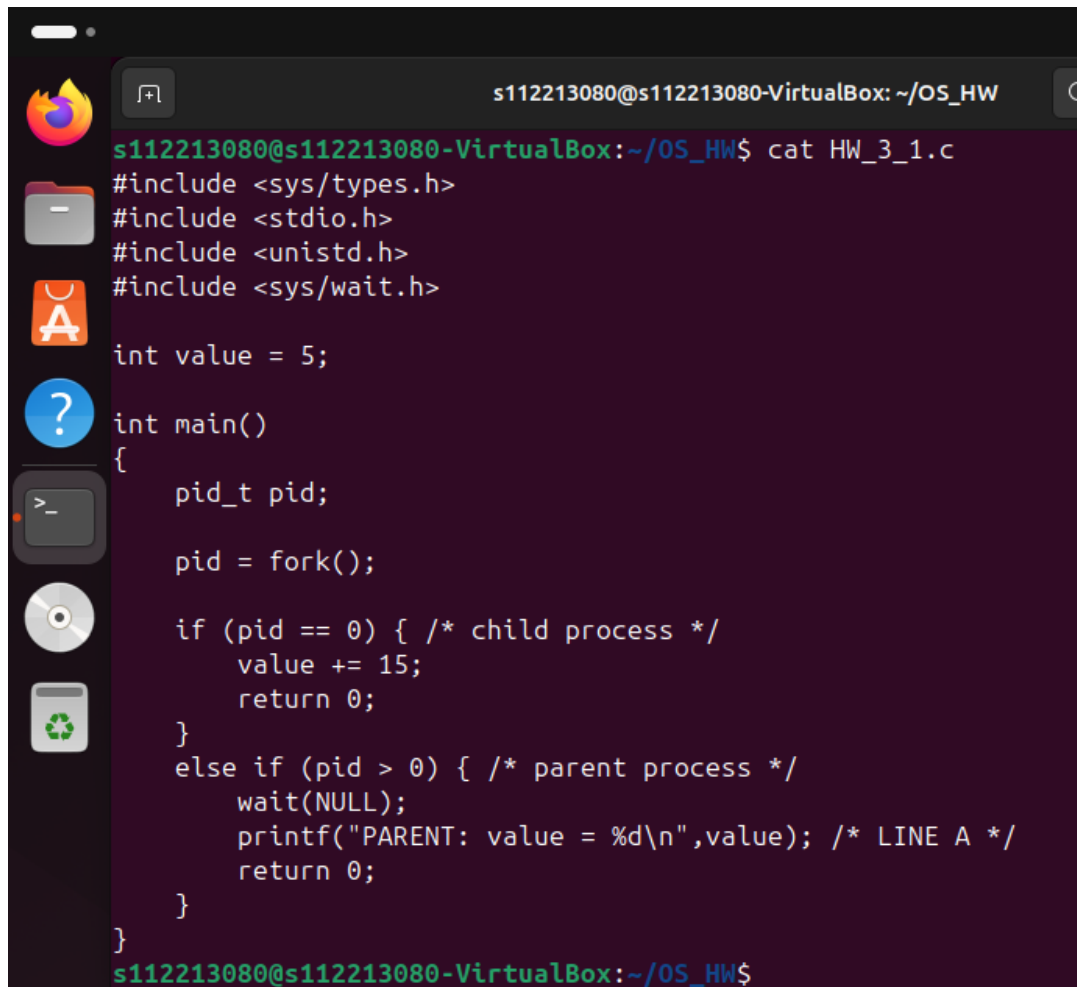
parent 的 pid = 2600  
 child 的 pid = 2603

(A) 在 child process 中, fork 回 0  
 (B) 在 child process 中, getpid() 得到 2603  
 (C) 在 parent process 中, getpid() 得到 2600  
 (D) 在 parent process 中, pid1 得到 2600

程式實作

3.1

原始程式：



A terminal window titled "s112213080@s112213080-VirtualBox: ~/OS\_HW". The user has entered the command `cat HW_3_1.c`. The output shows the source code of a C program that uses `fork()` to create a child process. The parent process prints "PARENT: value = 5" before returning. The child process increments the value by 15 and returns 0. The program is enclosed in `main()` and includes standard headers.

```
s112213080@s112213080-VirtualBox:~/OS_HW$ cat HW_3_1.c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int value = 5;

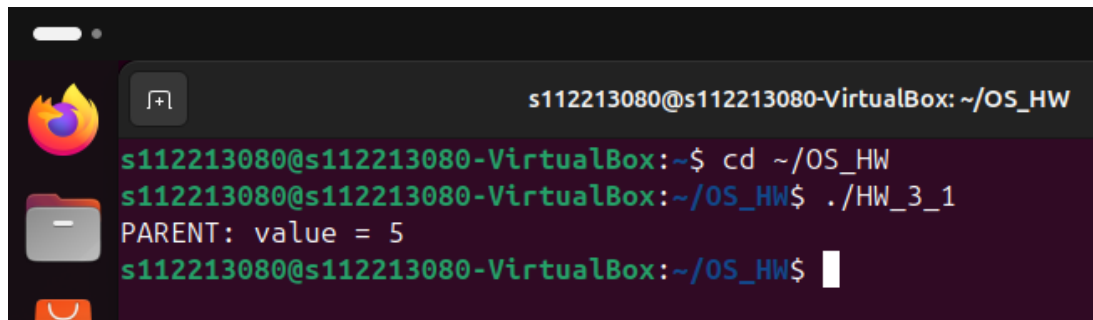
int main()
{
    pid_t pid;

    pid = fork();

    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d\n",value); /* LINE A */
        return 0;
    }
}
```

s112213080@s112213080-VirtualBox:~/OS\_HW\$

執行結果：

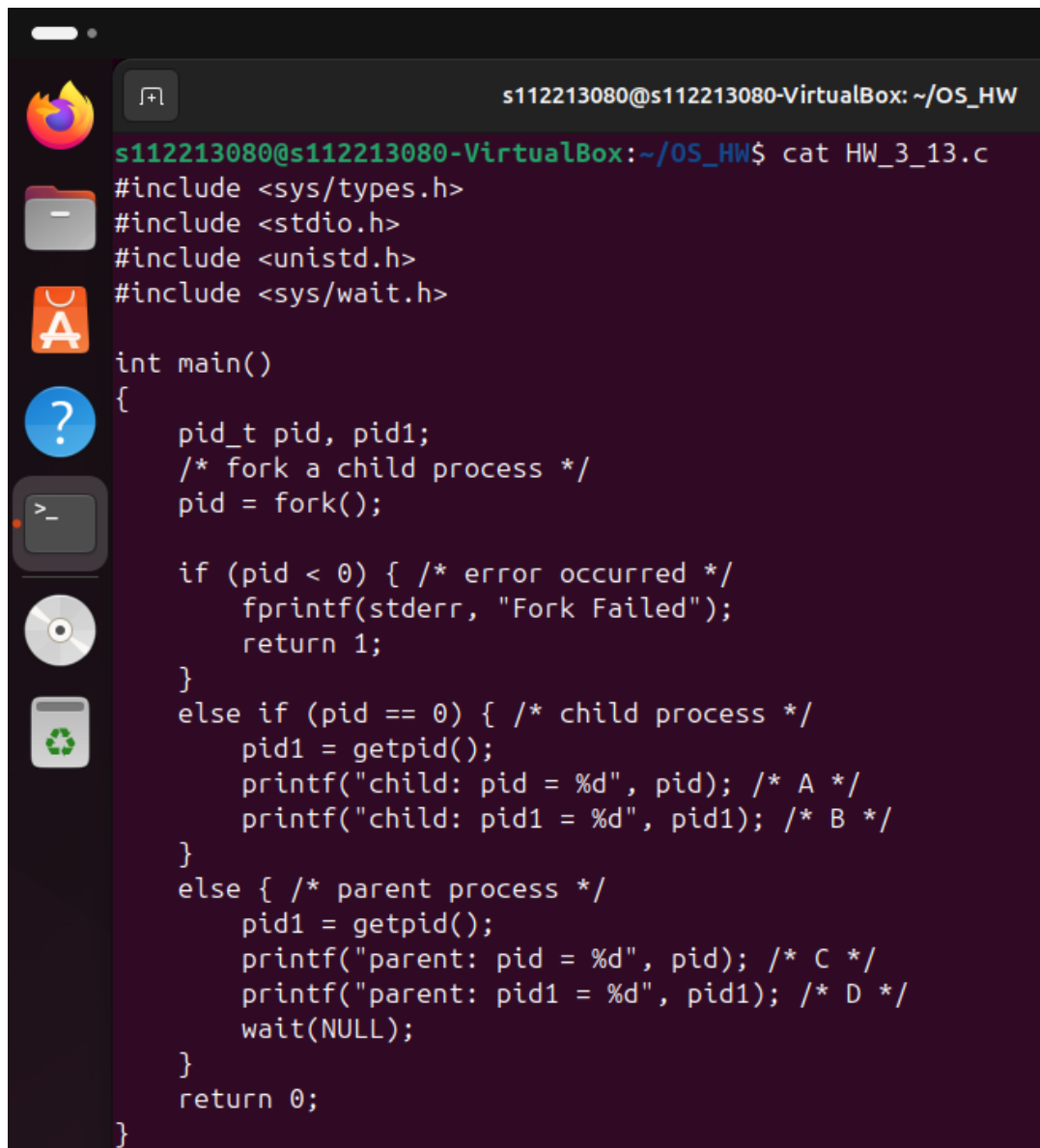


A terminal window titled "s112213080@s112213080-VirtualBox: ~/OS\_HW". The user has navigated to the directory `~/OS_HW` and executed the program `./HW_3_1`. The output shows "PARENT: value = 5", which matches the parent's value in the source code, indicating that the child process did not execute the `printf` statement.

```
s112213080@s112213080-VirtualBox:~$ cd ~/OS_HW
s112213080@s112213080-VirtualBox:~/OS_HW$ ./HW_3_1
PARENT: value = 5
s112213080@s112213080-VirtualBox:~/OS_HW$
```

3.13

原始程式：

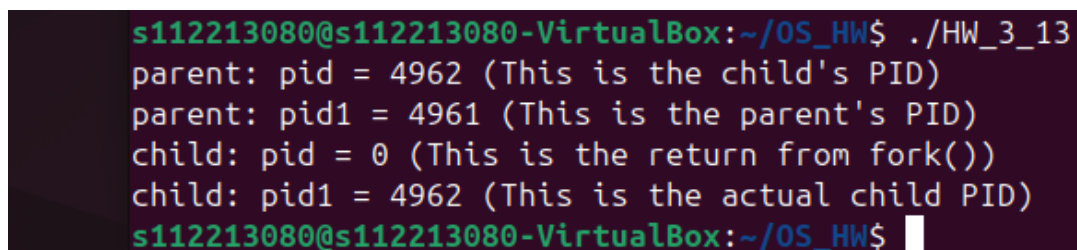
A terminal window titled 's112213080@s112213080-VirtualBox: ~/OS\_HW'. The terminal shows the command 'cat HW\_3\_13.c' and the contents of the file. The code is a C program that demonstrates the use of the fork() system call. It includes headers for <sys/types.h>, <stdio.h>, <unistd.h>, and <sys/wait.h>. The main function starts by declaring pid\_t variables pid and pid1. It then calls fork() to create a child process. If fork() returns a value less than 0, it indicates an error, and the program prints 'Fork Failed' to stderr and returns 1. If fork() returns 0, it indicates that the current process is the child. The child process prints its own pid (which is 0) and its parent's pid (pid1). If fork() returns a positive value, it indicates that the current process is the parent. The parent process prints its own pid (pid) and the child's pid (pid1), and then calls wait(NULL) to wait for the child to finish. Finally, the parent process returns 0.

```
s112213080@s112213080-VirtualBox: ~/OS_HW$ cat HW_3_13.c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t pid, pid1;
    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d", pid); /* A */
        printf("child: pid1 = %d", pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d", pid); /* C */
        printf("parent: pid1 = %d", pid1); /* D */
        wait(NULL);
    }
    return 0;
}
```

執行結果：

A terminal window showing the execution results of the program. The command './HW\_3\_13' has been executed, and the output is displayed. The output shows the parent process printing its pid (4962) and the child's pid (4961), followed by the child process printing its pid (0) and the parent's pid (4962).

```
s112213080@s112213080-VirtualBox:~/OS_HW$ ./HW_3_13
parent: pid = 4962 (This is the child's PID)
parent: pid1 = 4961 (This is the parent's PID)
child: pid = 0 (This is the return from fork())
child: pid1 = 4962 (This is the actual child PID)
s112213080@s112213080-VirtualBox:~/OS_HW$
```