Operating System Homework 3

Jinyan Xu, 3160101126, Information Security

1. Operating System Concept Chapter 3 Exercises: 3.1.

Answer:

3.1) Using the program shown in Figure 3.30, explain what the output will be at LINE A.

```
#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;
   }
}
```

Figure 3.30 What output will be at Line A?

I modify the code to explain the question more clearly:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int value = 5;
int main()
{
        pid_t pid;
        pid = fork();
        if (pid == 0) { /* child process */
                printf("CHILD: value = %d\n",value);
                printf("CHILD: The address of value = 0x%06x\n", &value);
                value += 15;
                printf("CHILD: i = i + 15\nCHILD: value = %d\n",value);
                return 0;
        }
        else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d\n",value); /* LINE A */
                printf("PARENT: The address of value = 0x%06x\n", &value);
        return 0;
        }
}
```

The output is:

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ gcc 3_1.c -o 3_1 2> /dev/null
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ ./3_1
CHILD: value = 5
CHILD: The address of value = 0x601048
CHILD: i = i + 15
CHILD: value = 20
PARENT: value = 5
PARENT: The address of value = 0x601048
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$
```

The initial value of ***value*** in child process is 5, and we can see that the parent process is not affected by the operation of the child process, so the child process inherits the parent process's data, rather than sharing it.

I notice that the addresses are same, I guess this is because the virtual memory.

And I did another experiment:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int data = 0;
int main() {
        pid_t pid;
        int stack = 1;
        int* heap = malloc(sizeof(int));
        *heap = 2;
        pid = fork();
        if (pid == 0) { /* child process */
                printf("CHILD: data = %d stack = %d heap = %d\n", data, stack, *heap);
                return 0;
        }
        else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PERENT: data = %d stack = %d heap = %d\n", data, stack, *heap);
        return 0;
        }
}
```

The output is:

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ gcc 3_1_1.c -o 3_1_1 2> /dev/null
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ ./3_1_1
CHILD: data = 0 stack = 1 heap = 2
PERENT: data = 0 stack = 1 heap = 2
```

Combine this, we can explain that:

The output at **LINE A** is 5, that's because when forking a child process, the child process will copy everything from its parent process, including the code, data, stack, and heap, and the operations in child process won't influence parent process, so the value of ***value*** is still 5.

Easter egg:

Recall the ASLR in homework 1, I find that the ASLR won't change the address of data segment.

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ cat /proc/sys/kernel/randomize_va_space
2
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ ./3_1
CHILD: value = 5
CHILD: The address of value = 0x601048
CHILD: i = i + 15
CHILD: value = 20
PARENT: value = 5
PARENT: The address of value = 0x601048
phantom0308@phantom0308-VirtualBox:~/桌面/OS 3$ ./3_1
CHILD: value = 5
CHILD: The address of value = 0x601048
CHILD: i = i + 15
CHILD: value = 20
PARENT: value = 5
PARENT: The address of value = 0x601048
```

2. Operating System Concept Chapter 3 Exercises: 3.2.

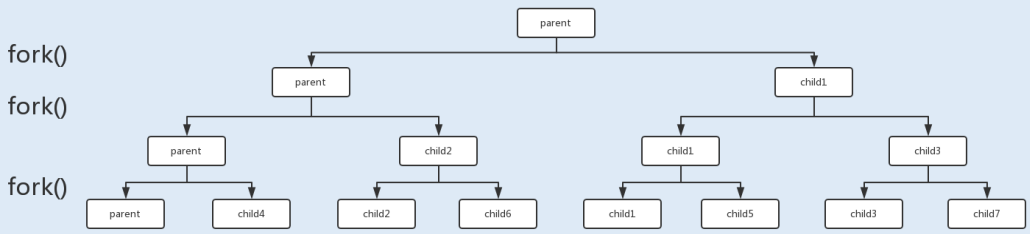
Answer:

3.2) Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

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

int main()
{
   /* fork a child process */
   fork();

   /* fork another child process */
   fork();

   /* and fork another */
   fork();

   return 0;
}
```

Figure 3.31 How many processes are created?

Before answering the questions, let's guess the flow of the program:



The result should be 8, let's prove it:

Modify the code following, I make the parent process terminates last, so the output won't be interrupted by the output from the children processes.

And I took a little bit of thought to make ***count*** look like it was shared by all the processes, so that it eventually outputs 0 to 7, 8 different results.

```c
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int main() {
        pid_t pid1, pid2, pid3;
        int count = 0;
        /* fork a child process */
        pid1 = fork();
        if (pid1 == 0)
                count++;

        /* fork another child process */
        pid2 = fork();
        if (pid2 == 0)
                count+=2;

        /* and fork another */
        pid3 = fork();
        if (pid3 == 0)
                count+=4;

        if(pid3)
                wait(NULL);
        if(pid2)
                wait(NULL);
        if(pid1)
                wait(NULL);
        printf("Process%d(pid = %d) terminated.\n", count, getpid());
        return 0;
}
```

And the result show that our guess is right!

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ gcc 3_2.c -o 3_2
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ ./3_2
Process4(pid = 4287) terminated.
Process6(pid = 4288) terminated.
Process2(pid = 4286) terminated.
Process5(pid = 4290) terminated.
Process7(pid = 4291) terminated.
Process3(pid = 4289) terminated.
Process1(pid = 4285) terminated.
Process0(pid = 4284) terminated.
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$
```

Here are 8 processes with different pid, so the answer is 8.

3. Write a kernel module that outputs the pid, process name, process state of the system.

Answer:

Here is my code, module, and Makefile:

Web links：https://pan.baidu.com/s/1UXNc_EBkg9KbY0upDGQ6PA Password：gt4e

First, we need to install the essential development tools and the kernel headers, using command follows:

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ uname -r
4.15.0-29-generic
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ sudo apt-get install build-essential linux-headers-$(uname -r)
[sudo] phantom0308 的密码：
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
build-essential 已经是最新版 (12.1ubuntu2)。
linux-headers-4.15.0-29-generic 已经是最新版 (4.15.0-29.31~16.04.1)。
linux-headers-4.15.0-29-generic 已设置为手动安装。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 104 个软件包未被升级。
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$
```

Well, it looks like that the files have already in my computer, let's get started!

I modify the code teacher showed in the class, following is my code:

```c
#include <linux/module.h>       // included for all kernel modules
#include <linux/kernel.h>       // included for KERN_INFO
#include <linux/init.h>         // included for __init and __exit macros
#include <linux/sched.h>        // include for struct task_struct
#include <linux/sched/signal.h> // include for for_each_process

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Jinyan Xu");
MODULE_DESCRIPTION("Operating System HW3.3 Print PCB");
MODULE_VERSION("0.01");

char* State_table[21] = { "TASK_RUNNING", "TASK_INTERRUPTIBLE", "TASK_UNINTERRUPTIBLE",
"TASK_NORMAL", "__TASK_STOPPED", "__TASK_TRACED", "TASK_ALL", "TASK_PARKED", "TASK_REPORT",
"TASK_DEAD", "TASK_WAKEKILL", "TASK_KILLABLE", "TASK_STOPPED", "TASK_TRACED", "TASK_WAKING",
"TASK_NOLOAD", "TASK_IDLE", "TASK_NEW", "TASK_STATE_MAX", "TASK_NOT_RUNNING", "WRONG_STATE"};

int Map_state(int state) {
        if(state < 0)
                return 19;
        switch (state) {
                case 0x0000: return 0;          case 0x0001: return 1;
                case 0x0002: return 2;          case 0x0003: return 3;
                case 0x0004: return 4;          case 0x0008: return 5;
                case 0x000F: return 6;          case 0x0040: return 7;
                case 0x007F: return 8;          case 0x0080: return 9;
                case 0x0100: return 10;         case 0x0102: return 11;
                case 0x0104: return 12;         case 0x0108: return 13;
                case 0x0200: return 14;         case 0x0400: return 15;
                case 0x0402: return 16;         case 0x0800: return 17;
                case 0x1000: return 18;         default:     return 20;
        }

}
static int __init print_PCB_init(void) {
        struct task_struct *task;
        printk(KERN_INFO "\033[1;34mProcess name     Pid  \tState\033[0m\n");
        for_each_process(task) {
                printk(KERN_INFO "\033[1;32m%-16s \033[0m[\033[36m%d\033[0m]\t%s\n", task->comm, task->pid, State_table[Map_state(task->state)]);
        }
        return 0;
}
static void __exit print_PCB_exit(void) {
        printk(KERN_INFO "\033[1;34mExit Print PCB Module.\033[0m\n");
}
module_init(print_PCB_init);
module_exit(print_PCB_exit);
```

Based on this code, let's do some discussion.

1) To running your kernel module, we will use the functions **module_init()** and **module_exit()**, these two functions declared in init.h, when we install our module the program will call module_init(), when remove calls module_exit().
2) Then put our attention on functions print_PCB_init() and print_PCB_exit(), the true kernel functions. When declared, I used __init and __exit, which defined in init.h, The kernel can take these macro as hint that the function is used only during the initialization phase and free up used memory resources after.

```
#define __init          __section(.init.text) __cold  __latent_entropy __noinitretpoline
#define __exit          __section(.exit.text) __exitused __cold notrace
```

3) Next is the struct: task_struct, which is known as process control block, knowing its specific content is crucial for us next, I cut out the definition related to the state.

```
/* Used in tsk->state: */
#define TASK_RUNNING                    0x0000
#define TASK_INTERRUPTIBLE              0x0001
#define TASK_UNINTERRUPTIBLE            0x0002
#define __TASK_STOPPED                  0x0004
#define __TASK_TRACED                   0x0008
/* Used in tsk->exit_state: */
#define EXIT_DEAD                       0x0010
#define EXIT_ZOMBIE                     0x0020
#define EXIT_TRACE                      (EXIT_ZOMBIE | EXIT_DEAD)
/* Used in tsk->state again: */
#define TASK_PARKED                     0x0040
#define TASK_DEAD                       0x0080
#define TASK_WAKEKILL                   0x0100
#define TASK_WAKING                     0x0200
#define TASK_NOLOAD                     0x0400
#define TASK_NEW                        0x0800
#define TASK_STATE_MAX                  0x1000

/* Convenience macros for the sake of set_current_state: */
#define TASK_KILLABLE                   (TASK_WAKEKILL | TASK_UNINTERRUPTIBLE)
#define TASK_STOPPED                    (TASK_WAKEKILL | __TASK_STOPPED)
#define TASK_TRACED                     (TASK_WAKEKILL | __TASK_TRACED)

#define TASK_IDLE                       (TASK_UNINTERRUPTIBLE | TASK_NOLOAD)

/* Convenience macros for the sake of wake_up(): */
#define TASK_NORMAL                     (TASK_INTERRUPTIBLE | TASK_UNINTERRUPTIBLE)
#define TASK_ALL                        (TASK_NORMAL | __TASK_STOPPED | __TASK_TRACED)

/* get_task_state(): */
#define TASK_REPORT                     (TASK_RUNNING | TASK_INTERRUPTIBLE | \
                                         TASK_UNINTERRUPTIBLE | __TASK_STOPPED | \
                                         __TASK_TRACED | EXIT_DEAD | EXIT_ZOMBIE | \
                                         TASK_PARKED)
```

State	Hex	Dec
TASK_RUNNING	0x0000	0
TASK_INTERRUPTIBLE	0x0001	1
TASK_UNINTERRUPTIBLE	0x0002	2
__TASK_STOPPED	0x0004	3
__TASK_TRACED	0x0008	8
TASK_PARKED	0x0040	64
TASK_DEAD	0x0080	128
TASK_WAKEKILL	0x0100	256
TASK_WAKING	0x0200	512
TASK_NOLOAD	0x0400	1024
TASK_NEW	0x0800	2048
TASK_STATE_MAX	0x1000	4096
TASK_KILLABLE	0x0102	258
TASK_STOPPED	0x0104	260
TASK_TRACED	0x0108	264
TASK_IDLE	0x0402	1026
TASK_NORMAL	0x0003	3
TASK_ALL	0x000F	15
TASK_REPORT	0x007F	127

I put them in a list to observe.
There are only 19 numbers but ranging from 0 to 4096. At first, I want to find a hash function to create a hash table to make the code clear.
Unluckily, I do not get it, as you can see, I finally use ***switch*** to make contact between state and state number.
And for_each_process() is also a macro, which in sched/signal.h, it defined as:

```
for (p = &init_task ; (p = next_task(p)) != &init_task ; )
```

We can use it to scan the whole process linked list.

4) We use printk() to display the output, printk() doesn't share the same parameters as printf(). For example, the KERN_INFO, which is a flag to declare what priority of logging should be set for this line, is defined without a comma. The kernel sorts this out inside the printk function to save stack memory.

Now the source code has completed, we need a Makefile to make the code become a module.

```
obj-m += 3_3.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
test:
	sudo dmesg -C
	sudo insmod 3_3.ko
	sudo rmmod 3_3.ko
	dmesg
```

The First line means to build a kernel module.
The command behind ***all*** means:
-C indicates that jump to the kernel source directory to read the Makefile there; and ***M=$(PWD)*** indicates that you then go back to the current directory to continue reading and executing the current Makefile.
The command behind ***clean*** means detele all file except source code and Makefile.
The commands behind ***test*** help us to input the commands to test our module.
Input ***make***:

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ make
make -C /usr/src/linux-headers-4.15.0-29-generic M=/home/phantom0308/桌面/OS_3 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-29-generic'
Makefile:976: "Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev, libelf-devel or elfutils-libelf-devel"
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-29-generic'
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$
```

Then we get 3_3.ko, this is the module we want, input ***make test***:

```
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$ make test
sudo dmesg -C
sudo insmod 3_3.ko
sudo rmmod 3_3.ko
dmesg
[70746.338387] Process name     Pid     State
[70746.338390] systemd          [1]     TASK_INTERRUPTIBLE
[70746.338391] kthreadd         [2]     TASK_INTERRUPTIBLE
[70746.338392] kworker/0:0H     [4]     TASK_IDLE
[70746.338393] mm_percpu_wq     [6]     TASK_IDLE
[70746.338394] ksoftirqd/0      [7]     TASK_INTERRUPTIBLE
[70746.338396] rcu_sched        [8]     TASK_IDLE
[70746.338397] rcu_bh           [9]     TASK_IDLE
[70746.338398] migration/0      [10]    TASK_INTERRUPTIBLE
[70746.338399] watchdog/0       [11]    TASK_INTERRUPTIBLE
```

……

```
[70746.338555] cupsd            [7230]  TASK_INTERRUPTIBLE
[70746.338558] cups-browsed     [7231]  TASK_INTERRUPTIBLE
[70746.338560] dbus             [7235]  TASK_INTERRUPTIBLE
[70746.338563] dbus             [7236]  TASK_INTERRUPTIBLE
[70746.338564] kworker/u2:2     [7721]  TASK_IDLE
[70746.338565] kworker/0:0      [7806]  TASK_IDLE
[70746.338566] kworker/u2:0     [8377]  TASK_IDLE
[70746.338567] kworker/u2:1     [9164]  TASK_IDLE
[70746.338568] make             [10799] TASK_INTERRUPTIBLE
[70746.338570] sudo             [10802] TASK_INTERRUPTIBLE
[70746.338571] insmod           [10803] TASK_RUNNING
[70746.359668] Exit Print PCB Module.
phantom0308@phantom0308-VirtualBox:~/桌面/OS_3$
```

We can't directly see the result of the module output, so we need to use dmesg to see the kernel log.
dmesg -c means clean the kernel log.
insmod means install the module, while rmmod mans remove.

Following is my code:

Web links：https://pan.baidu.com/s/1UXNc_EBkg9KbY0upDGQ6PA Password：gt4e

3_3.c

```
1.  #include <linux/module.h> // included for all kernel modules
2.  #include <linux/kernel.h> // included for KERN_INFO
3.  #include <linux/init.h>       // included for __init and __exit macros
4.  #include <linux/sched.h>  // include for struct task_struct
5.  #include <linux/sched/signal.h>   // include for for_each_process
6.
7.  MODULE_LICENSE("GPL");
8.  MODULE_AUTHOR("Jinyan Xu");
9.  MODULE_DESCRIPTION("Operating System HW3.3 Print PCB");
10. MODULE_VERSION("0.01");
11.
12. char* State_table[21] = { "TASK_RUNNING", "TASK_INTERRUPTIBLE", "TASK_UNINTERRUPTIBLE",
13. "TASK_NORMAL", "__TASK_STOPPED", "__TASK_TRACED", "TASK_ALL", "TASK_PARKED", "TASK_REPORT",
```

```
"TASK_DEAD", "TASK_WAKEKILL", "TASK_KILLABLE", "TASK_STOPPED", "TASK_TRACED", "TASK_WAKING",
"TASK_NOLOAD", "TASK_IDLE", "TASK_NEW", "TASK_STATE_MAX", "TASK_NOT_RUNNING", "WRONG_STATE"};

int Map_state(int state) {
    if(state < 0)
        return 19;
    switch (state) {
        case 0x0000: return 0;      case 0x0001: return 1;
        case 0x0002: return 2;      case 0x0003: return 3;
        case 0x0004: return 4;      case 0x0008: return 5;
        case 0x000F: return 6;      case 0x0040: return 7;
        case 0x007F: return 8;      case 0x0080: return 9;
        case 0x0100: return 10;     case 0x0102: return 11;
        case 0x0104: return 12;     case 0x0108: return 13;
        case 0x0200: return 14;     case 0x0400: return 15;
        case 0x0402: return 16;     case 0x0800: return 17;
        case 0x1000: return 18;     default:     return 20;
    }

}
static int __init print_PCB_init(void) {
    struct task_struct *task;
    printk(KERN_INFO "\033[1;34mProcess name     Pid  \tState\033[0m\n");
    for_each_process(task) {
        printk(KERN_INFO "\033[1;32m%-
16s \033[0m[\033[36m%d\033[0m]\t%s\n", task->comm, task->pid, State_table[Map_state(task->state)]);
    }
    return 0;
}
static void __exit print_PCB_exit(void) {
    printk(KERN_INFO "\033[1;34mExit Print PCB Module.\033[0m\n");
}

module_init(print_PCB_init);
module_exit(print_PCB_exit);
```

Makefile:

```
obj-m += 3_3.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
test:
    sudo dmesg -C
    sudo insmod 3_3.ko
    sudo rmmod 3_3.ko
    dmesg
```