

## 实验四：UNIX V6++中新进程创建与父子进程同步

### 1. 实验目的

(1) 结合课程所学知识，通过在 UNIX V6++实验环境中编写与父进程创建子进程的系统调用 `fork` 有关的应用程序，并观察其调试运行，熟悉 UNIX V6++中关于进程创建的过程及多进程编程技巧。

(2) 结合课程所学知识，通过在 UNIX V6++实验环境中编写与进程终止及父子进程同步的系统调用 `exit` 和 `wait` 有关的应用程序，并观察其调试运行，熟悉 UNIX V6++中关于子进程终止的详细过程及父子进程之间的数据传递。

### 2. 实验设备及工具

已配置好 UNIX V6++运行和调试环境的 PC 机一台。

### 3. 实验准备工作

(1) 在 UNIX V6++中添加获得某一进程的父进程 ID 号的系统调用及库函数（具体过程参照实验二）。其中，系统调用处理子程序 `Sys_Getppid()` 的实现可参考代码 1。

```
/* 50 = getpid count = 0 */
int SystemCall::Sys_Getppid()
{
    User& u = Kernel::Instance().GetUser();
    u.u_ar0[User::EAX] = u.u_procp->p_ppid;
    return 0; /* GCC likes it ! */
}
```

代码 1

(2) 参照实验二，进一步熟悉如何在 UNIX V6++中编译、调试和运行一个用户编写的应用程序。这里不再赘述。

### 4. 实验内容

#### 4.1. 关于 FORK 系统调用

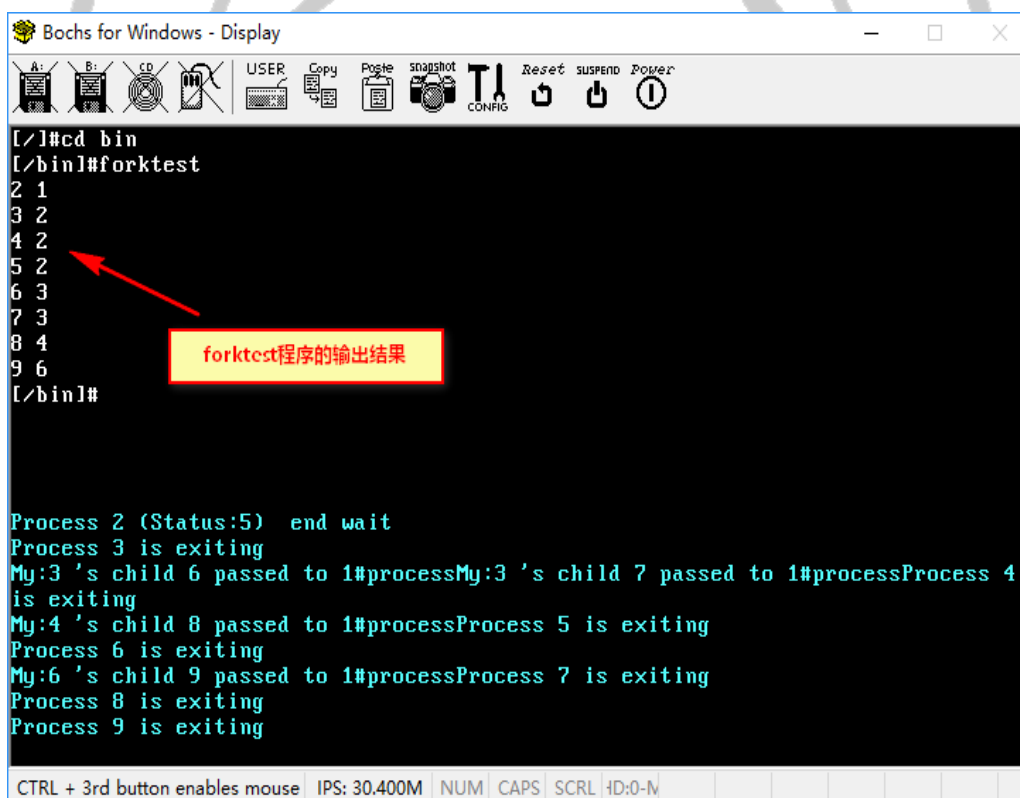
在 UNIX V6++的 `src/program` 文件夹下添加一个 `forktest.c` 文件，代码如代码 2 所示。按照实验二的方法编译后形成 UNIX V6++内核/bin 目录下的可执行文件 `forktest`。

```
#include <stdio.h>
#include <sys.h>

int main1()
{
    int i;
    printf("%d %d \n", getpid(),getppid());
    for(i = 0; i < 3; ++i)
        if(fork()==0)
            printf("%d %d \n", getpid(),getppid());
    sleep(2);
    return 1;
}
```

代码 2

启动 UNIX V6++, 并运行 forktest 程序, 得到如图 1 所示的输出 (具体的进程 ID 号可能会有不同)。



```
[/]  
[/]#cd bin  
[/bin]#forktest  
2 1  
3 2  
4 2  
5 2  
6 3  
7 3  
8 4  
9 6  
[/bin]#  
  
Process 2 (Status:5) end wait  
Process 3 is exiting  
My:3 's child 6 passed to 1#process  
My:3 's child 7 passed to 1#process  
Process 4 is exiting  
My:4 's child 8 passed to 1#process  
Process 5 is exiting  
Process 6 is exiting  
My:6 's child 9 passed to 1#process  
Process 7 is exiting  
Process 8 is exiting  
Process 9 is exiting  
  
CTRL + 3rd button enables mouse  IPS: 30.400M  NUM  CAPS  SCRL  ID:0-N
```

图 1

将代码 2 中的 sleep(2)语句删除, 重复上述的步骤后, 重新运行 forktest 程序, 获得如图 3 所示的输出 (具体的进程 ID 号可能会有不同)。这时按回车键, 观察屏幕输出会有什么变化。

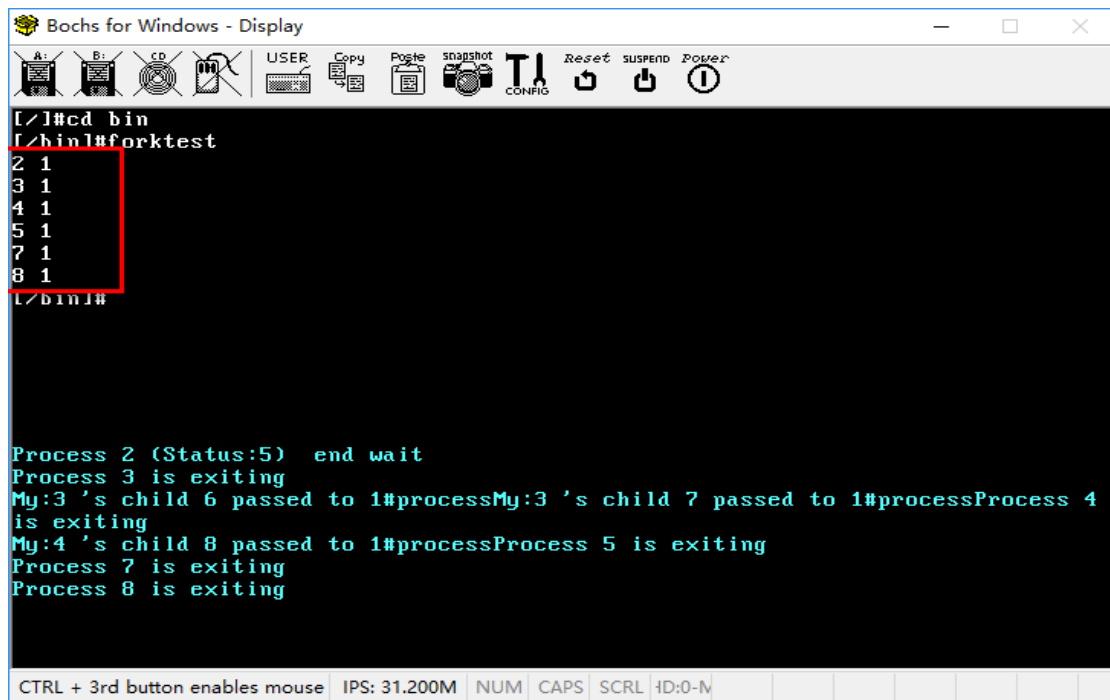


图 2

#### 4.2. 关于 WAIT 和 EXIT 系统调用

在 UNIX V6++ 的 src/program 文件夹下添加一个 exitwaittest.c 文件, 代码如代码 3 所示。  
按照实验三的方法编译后形成 UNIX V6++ 内核/bin 目录下的可执行文件 exitwaittest。

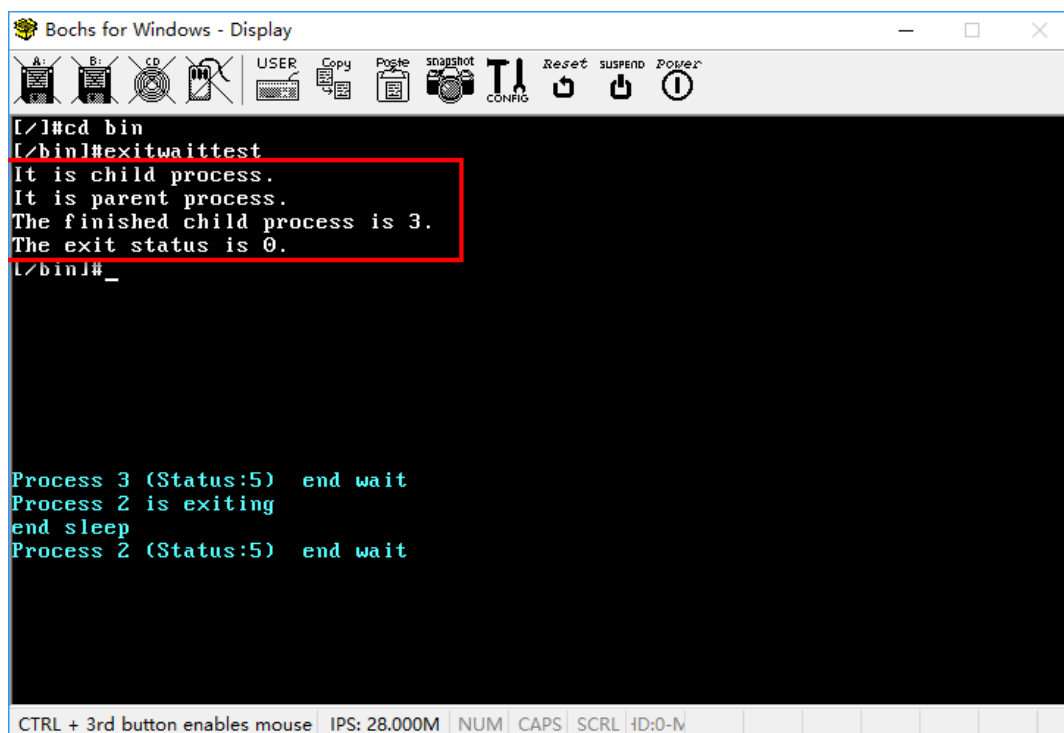
```

#include <stdio.h>
#include <sys.h>
main1( )
{
    int i,j;
    if(fork())
    {
        i=wait(&j);    /* i 为终止的子进程的进程号 */
        printf("It is parent process. \n");
        printf("The finished child process is %d. \n", i);
        printf("The exit status is %d. \n", j);
    }
    else
    {
        printf("It is child process. \n");
        exit(0);
    }
}

```

代码 3

启动 UNIX V6++, 并运行 `exitwaittest` 程序, 得到如图 3 所示的输出。



```
[/]#cd bin
[/bin]#exitwaittest
It is child process.
It is parent process.
The finished child process is 3.
The exit status is 0.
[/bin]#_

Process 3 (Status:5) end wait
Process 2 is exiting
end sleep
Process 2 (Status:5) end wait
```

图 3

将代码 3 中的语句 `exit(0)` 修改为 `exit(1)`, 观察编译运行后的输出情况。

## 5. 实验报告要求

(1) (1 分) 按上述过程, 分别编辑、编译并运行 `forktest` 和 `exitwaittest` 程序, 截图展示程序的输出结果。

(2) 回答下列思考题:

思考题 1 (2 分): 分析图 1 中的输出结果, 指出 `i` 分别为 0, 1, 2 时, 创建的是哪个进程? 画出各个进程之间的父子关系。

思考题 2 (1 分): 分析图 3 中的输出结果, 解释父进程是如何接收到子进程的终止码的 (可结合调试过程说明)?

思考题 3\*: 分析图 2 中的输出结果和图 1 输出不同的原因是什么?

## 6. 特别提醒 (重要)

在完成实验的过程中, 如果需要调试 `forktest` 或 `exitwaittest` 程序, 需要对调试环境的设置做一些修改。将需要调试的可执行程序由 `Kernel.exe` 分别修改为 `forktest.exe` 或

exitwaittest.exe（如图 4），将起始调试点修改为 main1（如图 5 所示，forktest 和 exitwaittest 中的定义主程序）。然后就可以在程序中设置断点，观察变量和寄存器的值（如图 6）。

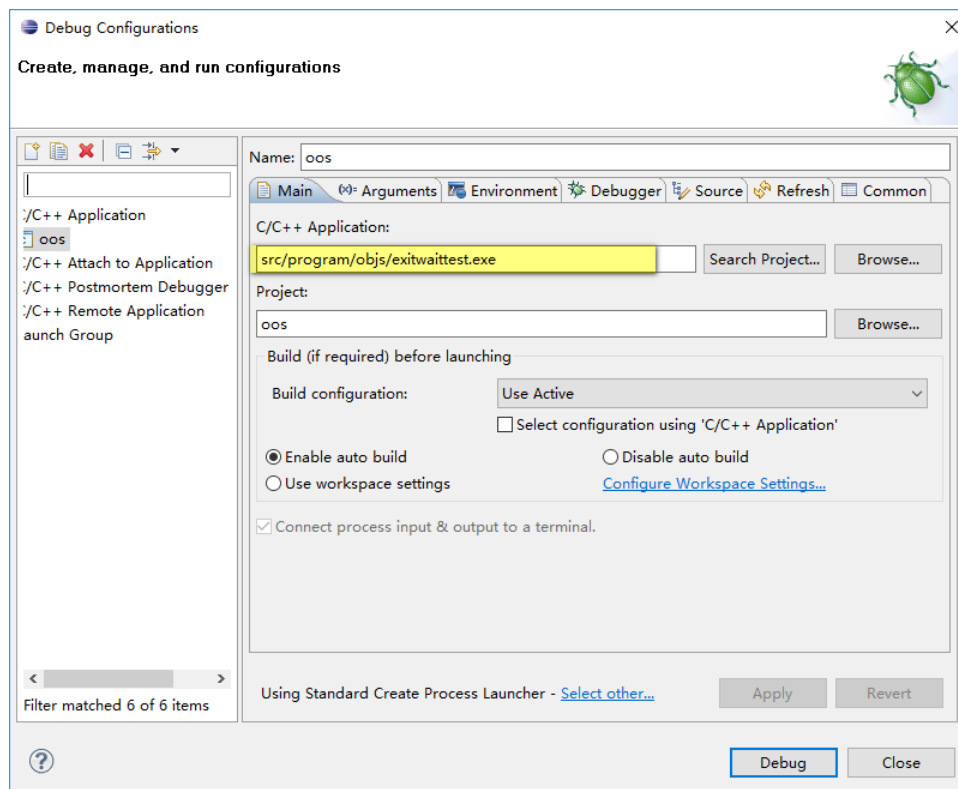


图 4

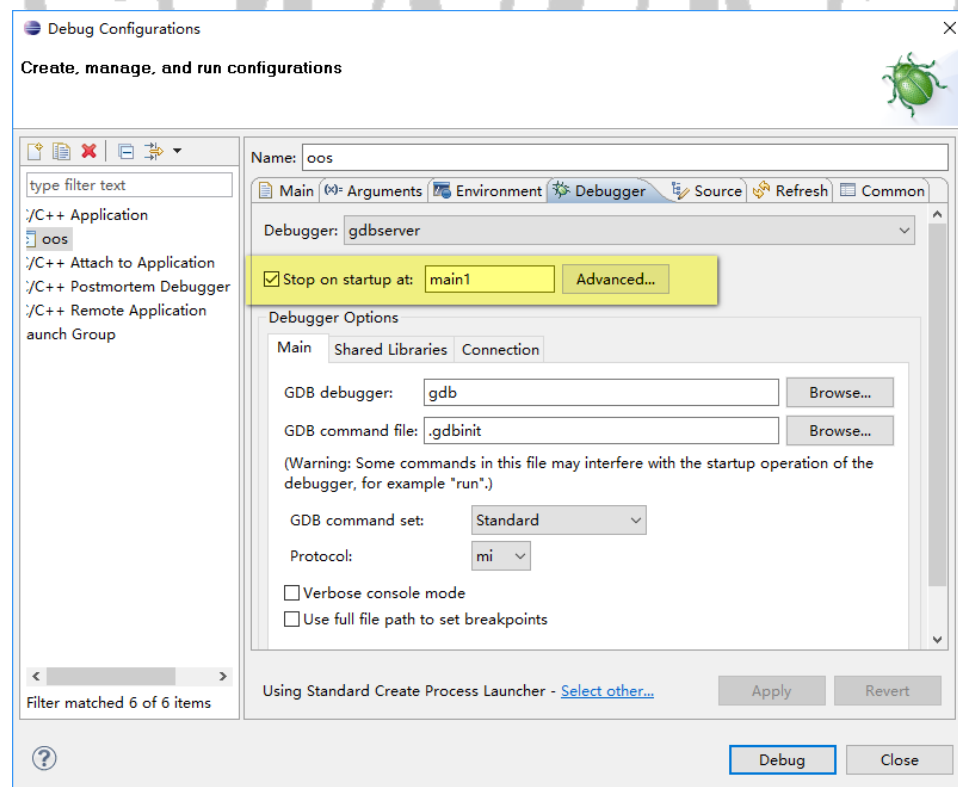


图 5

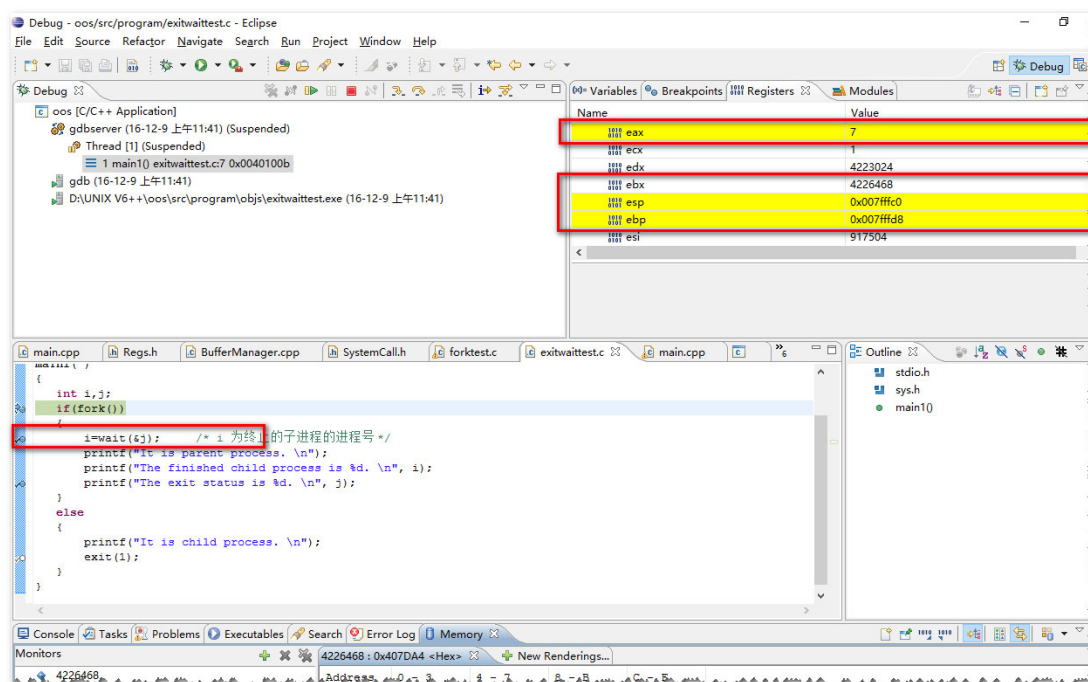


图 6