

OpenEuler实验：Linux下典型的进程控制操作

基本操作

- 输入 `vi name.cpp` 指令编写代码
- 输入 `i` 进入Insert模式开始输入代码
- 按 `Esc` 退出Insert模式
- 输入 `:wq` 保存并退出
- 输入 `touch name.txt` 创建一个空文件
- 输入 `g++ name.cpp -o name` 编译代码
- 输入 `./name > name.txt` 运行程序，并将结果保存在name.txt文件中

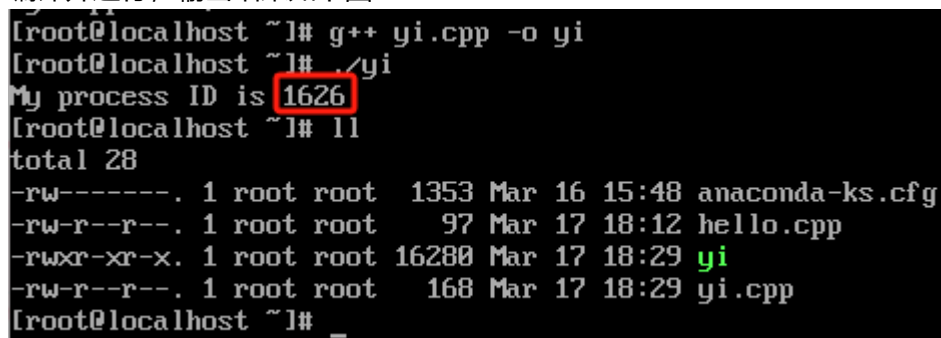
1. 获取进程的PID

- 输入以下代码
 - `getpid()` 函数用于获取当前进程的PID。

```
#include <iostream>
#include <unistd.h>

int main() {
    pid_t mypid;
    mypid = getpid();
    printf("The process ID is: %d\n", mypid);
    return 0;
}
```

- 编译并运行，输出结果如下图



```
[root@localhost ~]# g++ yi.cpp -o yi
[root@localhost ~]# ./yi
My process ID is 1626
[root@localhost ~]# ll
total 28
-rw-----. 1 root root 1353 Mar 16 15:48 anaconda-ks.cfg
-rw-r--r--. 1 root root 97 Mar 17 18:12 hello.cpp
-rwxr-xr-x. 1 root root 16288 Mar 17 18:29 yi
-rw-r--r--. 1 root root 168 Mar 17 18:29 yi.cpp
[root@localhost ~]# _
```

- 结果分析：输出process ID为1626，表示当运行这个程序时，操作系统为这个程序分配了一个PID，值为1626。

2. 进程创建与父子进程关系

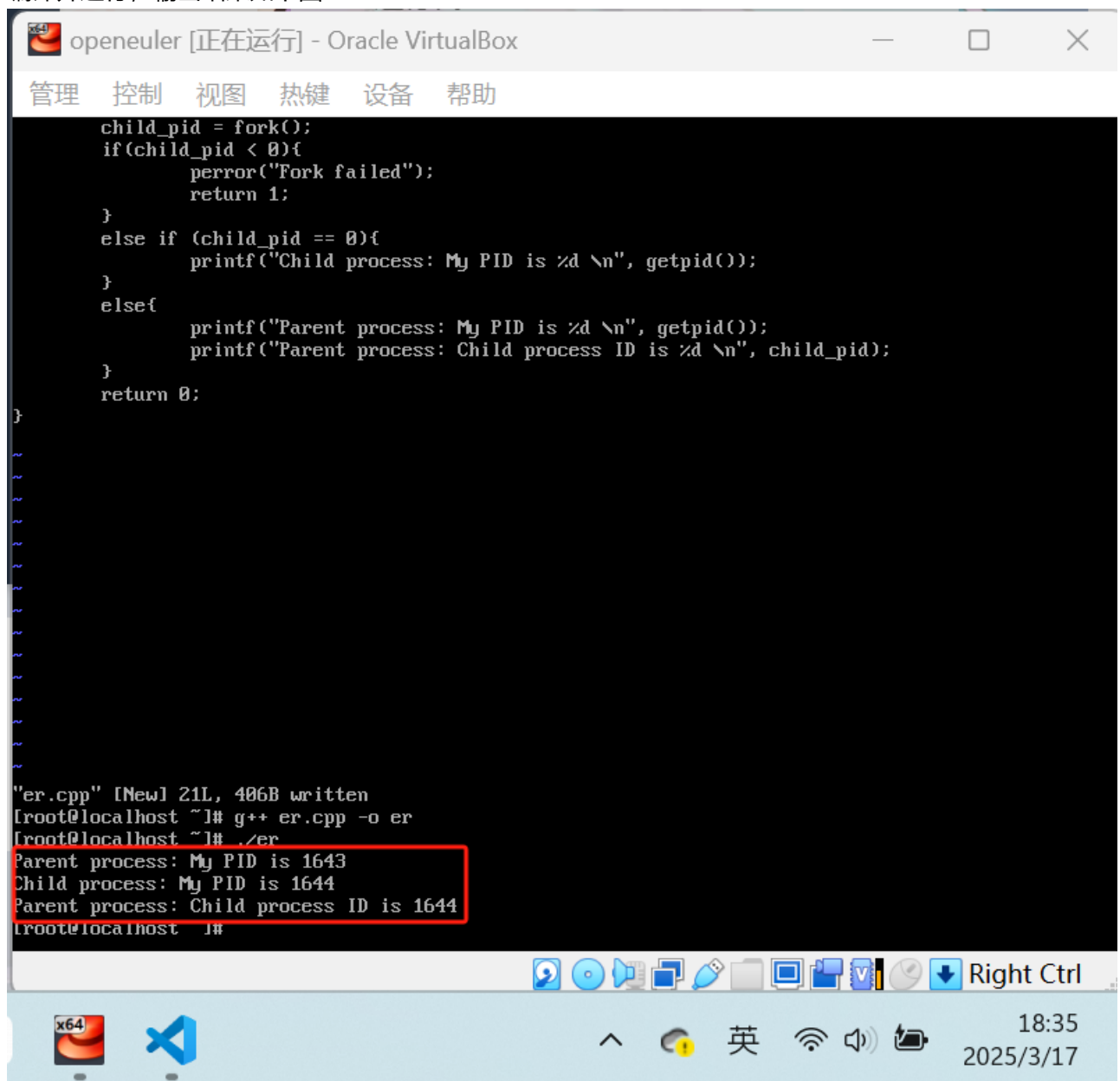
- 输入以下代码：使用`fork()`系统调用创建一个子进程，并分别在父进程和子进程中打印它们的进程ID。

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

```
#include <unistd.h>

int main() {
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0) {
        perror("fork failed");
        return 1;
    } else if (child_pid == 0) {
        printf("Child process: My PID is %d\n", getpid());
    } else {
        printf("Parent process: My PID is %d\n", getpid());
        printf("Parent process: Child process PID is %d\n", child_pid);
    }
    return 0;
}
```

- 编译并运行，输出结果如下图



```
openeuler [正在运行] - Oracle VirtualBox
管理 控制 视图 热键 设备 帮助

child_pid = fork();
if(child_pid < 0){
    perror("Fork failed");
    return 1;
}
else if (child_pid == 0){
    printf("Child process: My PID is %d \n", getpid());
}
else{
    printf("Parent process: My PID is %d \n", getpid());
    printf("Parent process: Child process ID is %d \n", child_pid);
}
return 0;
}

"er.cpp" [New] 21L, 406B written
[root@localhost ~]# g++ er.cpp -o er
[root@localhost ~]# ./er
Parent process: My PID is 1643
Child process: My PID is 1644
Parent process: Child process ID is 1644
[root@localhost ~]#
```

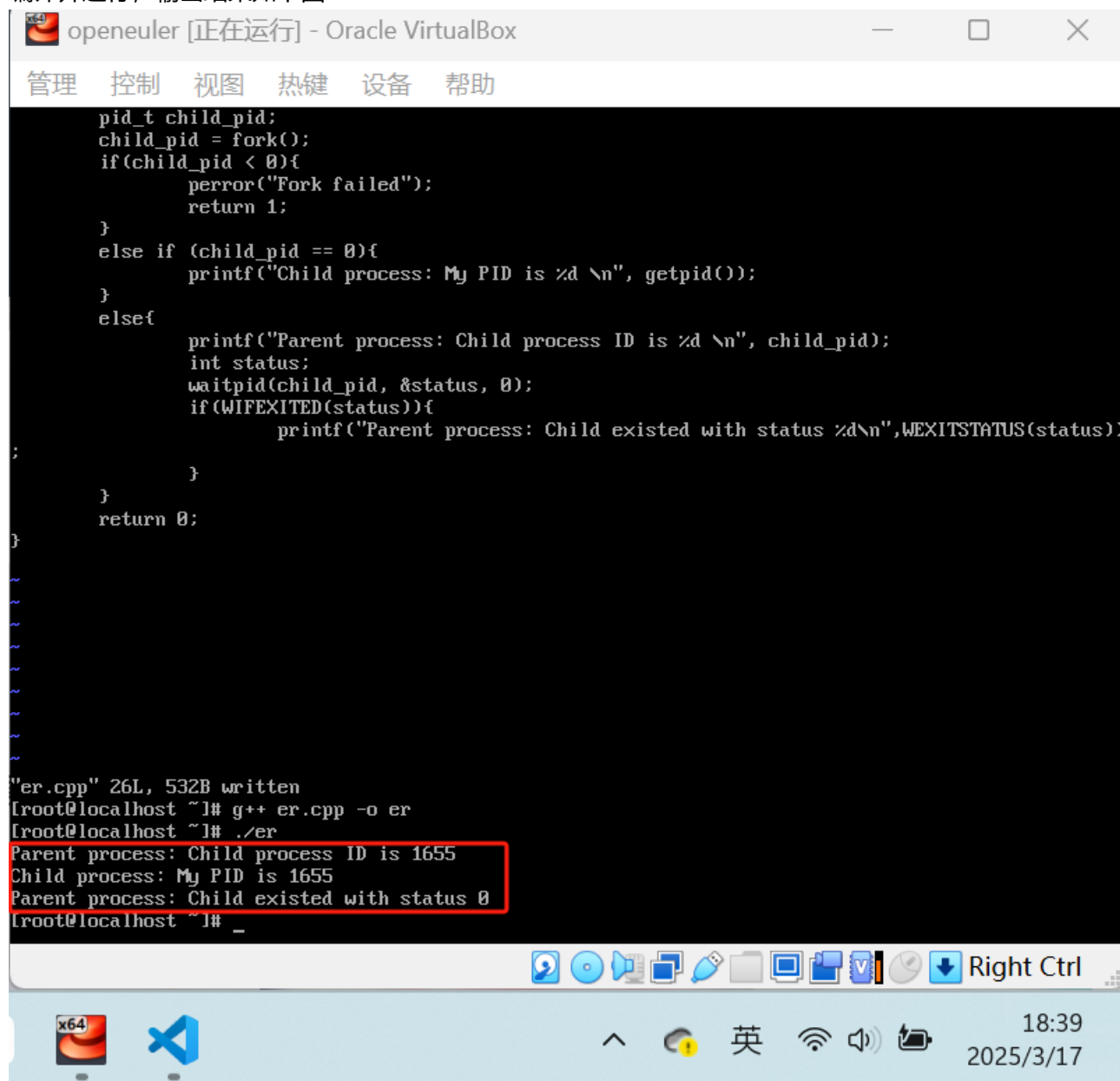
- 结果分析：子进程的PID和父进程的PID不同，表明虽然子进程是父进程的副本，但是它们是独立的进程，有自己的资源和地址空间

3. 父进程等待子进程结束

- 将2中代码修改为一下代码

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (child_pid == 0)
    {
        printf("Child process:My PID is %d \n", getpid());
    }
    else
    {
        printf("Parent process: Child process ID is %d \n", child_pid);
        int status;
        waitpid(child_pid, &status, 0);
        if (WIFEXITED(status))
        {
            printf("Parent process: Child exited with status %d\n",
WEXITSTATUS(status));
        }
    }
    return 0;
}
```

- 编译并运行，输出结果如下图



```
openeuler [正在运行] - Oracle VirtualBox
管理 控制 视图 热键 设备 帮助

pid_t child_pid;
child_pid = fork();
if(child_pid < 0){
    perror("Fork failed");
    return 1;
}
else if (child_pid == 0){
    printf("Child process: My PID is %d \n", getpid());
}
else{
    printf("Parent process: Child process ID is %d \n", child_pid);
    int status;
    waitpid(child_pid, &status, 0);
    if(WIFEXITED(status)){
        printf("Parent process: Child existed with status %d\n",WEXITSTATUS(status));
    }
}
return 0;

"er.cpp" 26L, 532B written
[root@localhost ~]# g++ er.cpp -o er
[root@localhost ~]# ./er
Parent process: Child process ID is 1655
Child process: My PID is 1655
Parent process: Child existed with status 0
[root@localhost ~]# _
```

- 结果分析：父进程在调用`waitpid()`后进入等待状态，直到子进程结束后才继续执行后续的代码。子进程正常退出。父进程通过`WEXITSTATUS()`函数获取子进程的退出状态，并打印出来。

4. 多次fork()实验

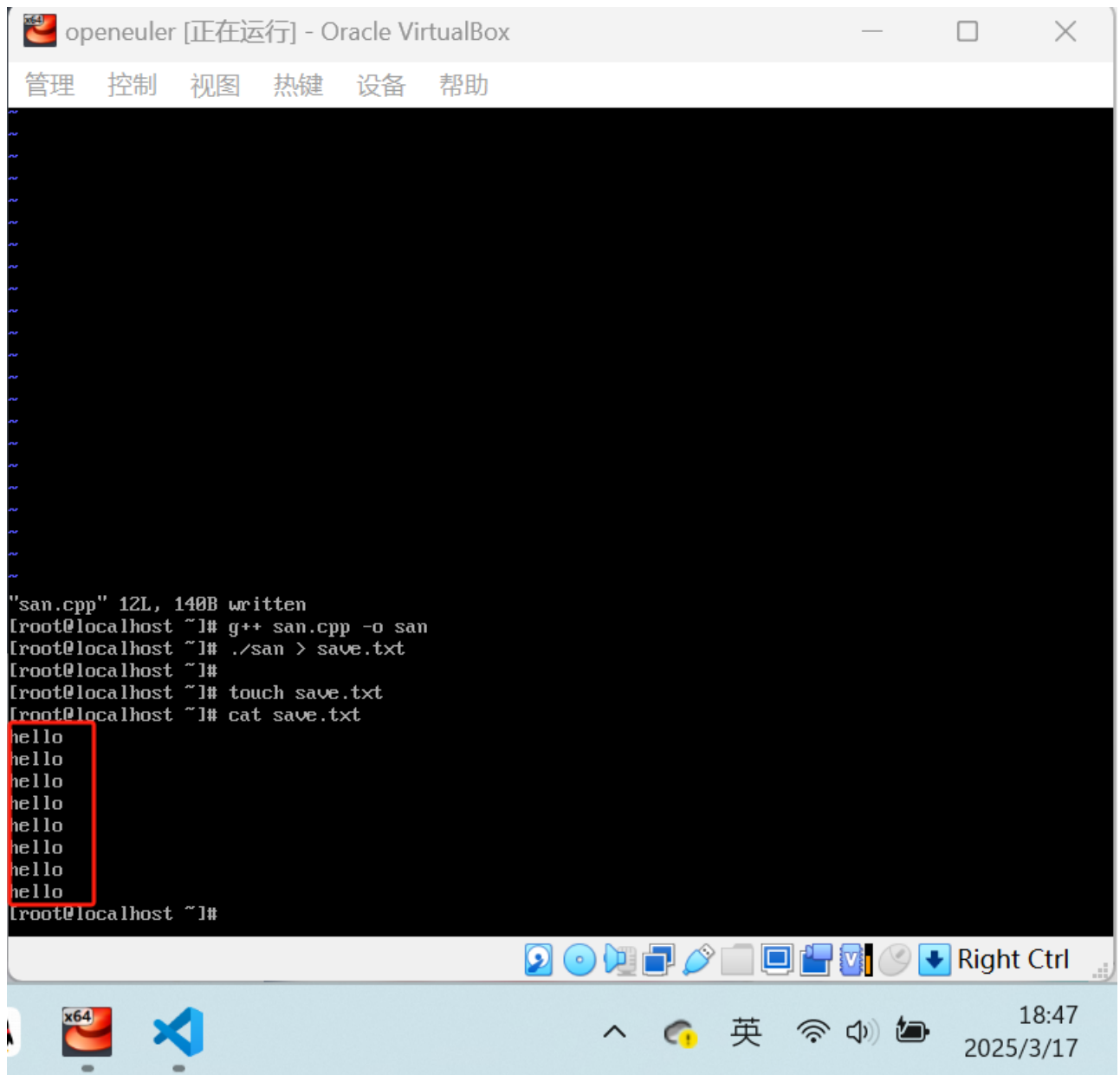
- 输入以下代码，使用`fork()`创建多个子进程

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

int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
}
```

```
    return 0;
}
```

- 编译并运行，输出结果如下图



```
openeuler [正在运行] - Oracle VirtualBox
管理 控制 视图 热键 设备 帮助

"san.cpp" 12L, 140B written
[root@localhost ~]# g++ san.cpp -o san
[root@localhost ~]# ./san > save.txt
[root@localhost ~]#
[root@localhost ~]# touch save.txt
[root@localhost ~]# cat save.txt
hello
hello
hello
hello
hello
hello
hello
hello
hello
[root@localhost ~]#
```

- 结果分析：每次调用 `fork()` 后，当前进程都会复制出一个新的进程，第一次 `fork()` 后有两个进程，第二次 `fork()` 后每个进程复制出一个子进程，总共有四个进程。第三次 `fork()` 后，每个进程又复制出一个子进程，总共有八个进程，输出8次"hello"。

5. 进程独立性实验

- 输入以下代码

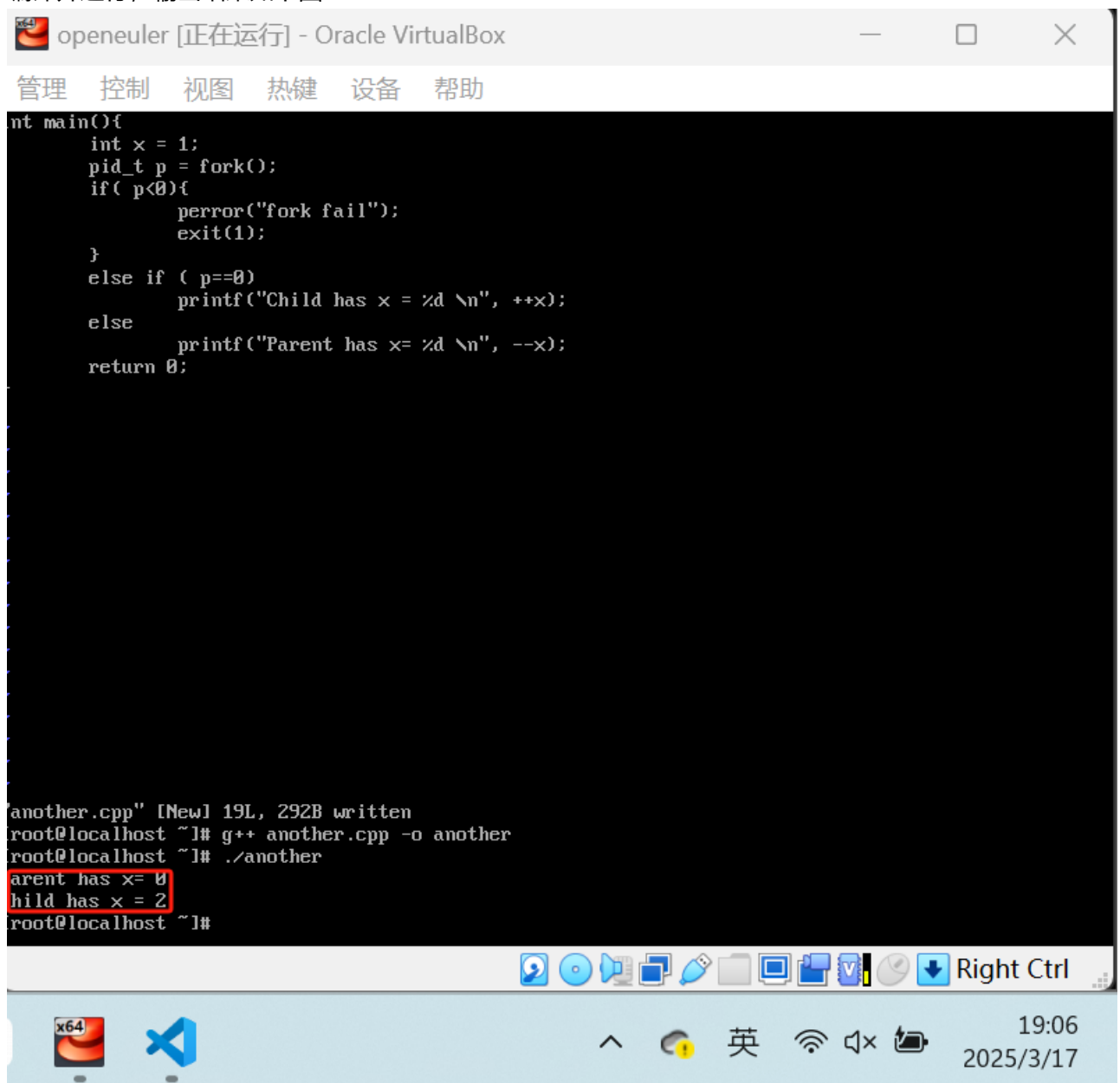
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main()
```

```
{
    int x = 1;
    pid_t p = fork();
    if (p < 0)
    {
        perror("fork fail");
        exit(1);
    }
    else if (p == 0)
        printf("Child has x = %d \n", ++x);
    else
        printf("Parent has x = %d\n", --x);

    return 0;
}
```

- 编译并运行，输出结果如下图



```
nt main(){
    int x = 1;
    pid_t p = fork();
    if ( p<0){
        perror("fork fail");
        exit(1);
    }
    else if ( p==0)
        printf("Child has x = %d \n", ++x);
    else
        printf("Parent has x= %d \n", --x);
    return 0;
}

another.cpp" [New] 19L, 292B written
root@localhost ~]# g++ another.cpp -o another
root@localhost ~]# ./another
Parent has x= 0
Child has x = 2
root@localhost ~]#
```

- 结果分析：在`fork()`调用后创建的子进程和原本的父进程拥有各自独立的内存空间。子进程的自增操作对父进程的`x`无影响，验证了进程间变量互不干扰的特性。

