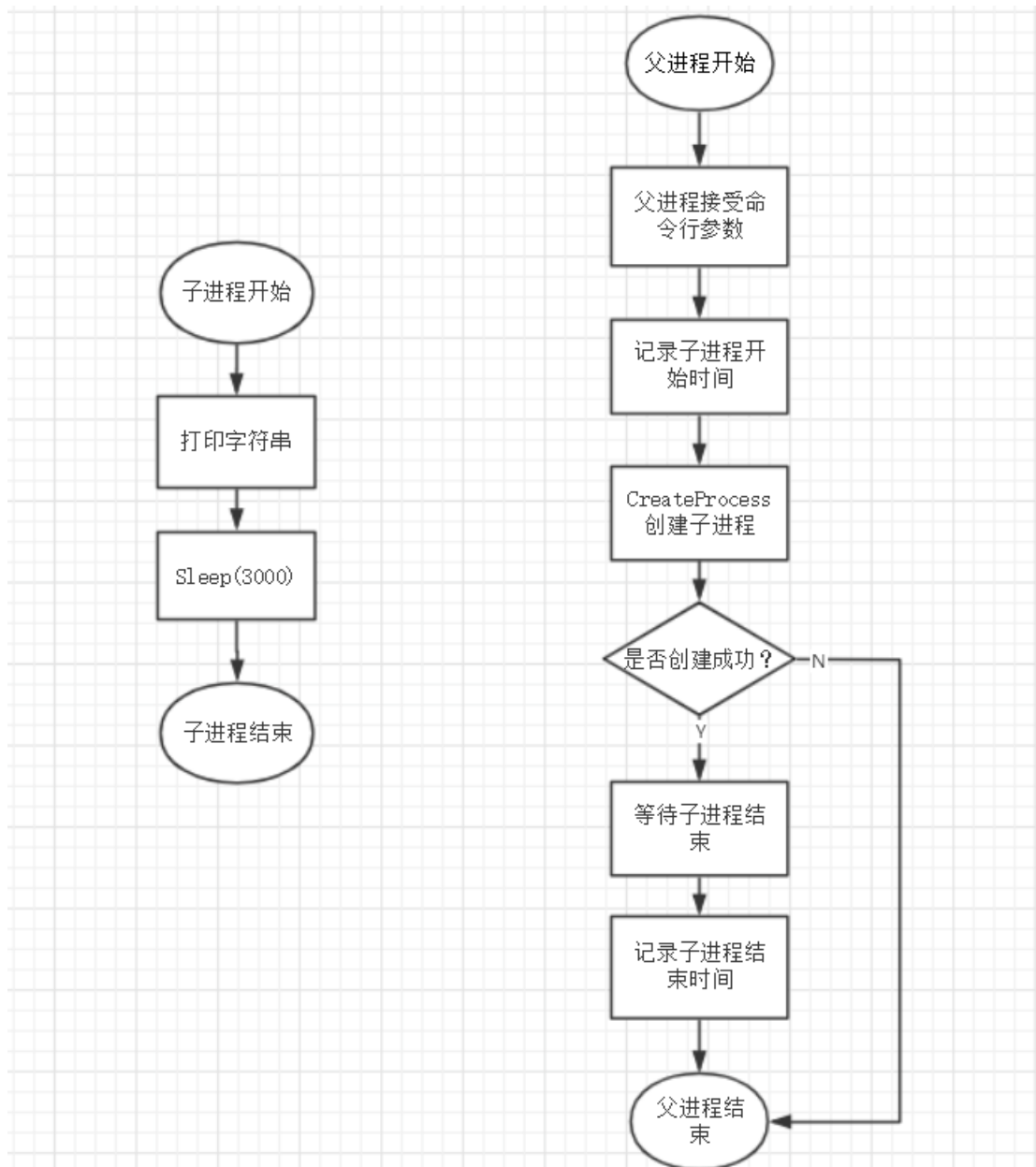


实验名称	进程控制		
学号	1120180207	姓名	唐小娟
<p>1. 实验目的</p> <p>1. 了解进程的概念</p> <p>(1). 了解 linux 和 windows 系统进程的区别</p> <p>2. 掌握进程的有关操作。</p> <p>(1). 掌握 linux 和 windows 进程调用的功能</p> <p>1). 掌握如何创建进程</p> <p>2). 掌握如何实现进程的同步问题</p> <p>二、实验内容</p> <p>1. Windows 下的进程控制</p> <p>(1). 使用 CreateProcess() 创建进程</p> <p>(2). 使用 WaitForSingleObject() 和子进程同步</p> <p>(3). 调用 GetSystemTime() 来获取时间</p> <p>2. Linux 下的进程控制</p> <p>(1). 使用 fork()/execv() 来创建进程运行程序</p> <p>(2). 使用 wait() 等待新创建的进程结束</p> <p>(3). 调用 gettimeofday() 来获取时间</p> <p>三、实验环境及配置方法</p> <p>操作系统: Windows 10, Ubuntu 20.04, Linux 5.4.0-42</p> <p>集成开发环境: Microsoft VS Code</p> <p>编译器: gcc 9.3.0</p> <p>四、实验方法和实验步骤（程序设计与实现）</p> <p>1. Windows 下的进程控制</p> <p>根据实验需要完成的功能，程序控制流程如下：</p>			



(1). `CreateProcess()` 创建进程

```

BOOL bCreateOK = CreateProcess(
    NULL, //可执行的应用程序名称（这里可以为 NULL）
    szCmdLine, //要执行的命令行
    NULL, //缺省的进程安全性
    NULL, //缺省的线程安全性
    FALSE, //不继承句柄
    CREATE_NEW_CONSOLE, //创建新的控制台
    NULL, //使用父进程的环境块
    NULL, //使用父进程的工作目录
    &si, //启动信息
    &pi); //返回新进程和线程的结构信息
    
```

如果返回值为 True，则表示该子进程创建成功；子进程所执行的程序由前两个变量指定，由于第一个为 NULL，则根据 szCmdLine 的字符串得到执行程序名以及参数；si 是一个 STARTUPINFO 结构体，可以设定启动信息，也可以为 NULL；pi 是输出变量，返回被创建进程的信息。

(2). WaitForSingleObject() 实现进程同步

使用 WaitForSingleObject() 等待子进程执行完毕，等待执行时间是 INFINITE，表示一直等待直到子进程对象变为有信号为止。

(3). GetSystemTime() 来获取时间

在调用 CreateProcess() 函数之前和 WaitForSingleObject() 函数之后分别调用 GetSystemTime() 来获取子进程开始时间与结束时间，二者相减可以得到子进程运行时间。

(4). CloseHandle() 关闭句柄

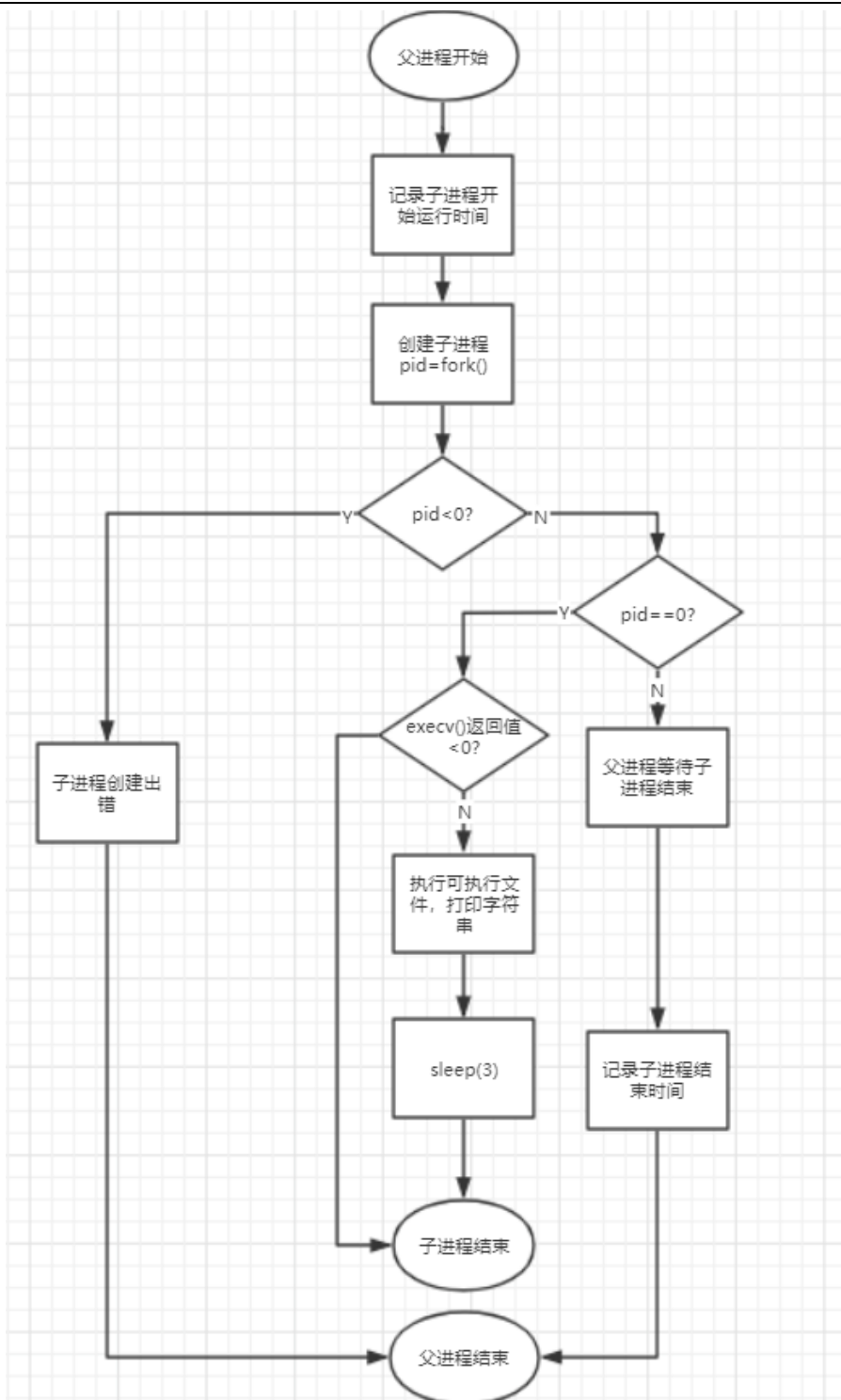
关闭对象句柄，防止造成句柄资源的浪费。

(5). GetTime() 获取以毫秒为单位的时间

获取以毫秒为单位的时间便于计算子进程运行时间。

2. Linux 下的进程控制

依据实验要求，程序控制流程设计如下：



(1) . fork()/execv() 创建并运行进程

在父进程中，调用 fork() 函数创建一个新的子进程，并继承父进程的代码段，返回值在父进程返回的是子进程的进程标识，在子进程中返回的是 0；判断 pid 的值：

如果 pid<0，那么创建进程出错；

如果 pid==0，那么当前进程为子进程运行，由参数 argv[1] 值得到执行程序名，

调用 `execv()` 将新程序覆盖子进程中原来继承父进程的代码和数据，执行子进程；

如果 `pid>0`，那么当前进程为父进程，等待子进程结束后再记录子进程结束时间后结束。

(2). `wait()` 等待子进程结束

父进程调用 `wait()`，等待子进程结束。

(3). `gettimeofday()` 获取时间

在子进程创建之前和 `wait()` 之后调用 `gettimeofday()` 获得子进程开始和结束时间。

五、实验结果和分析

1. Windows 下的进程控制

打开 cmd 切换到代码文件目录下，执行编译命令：

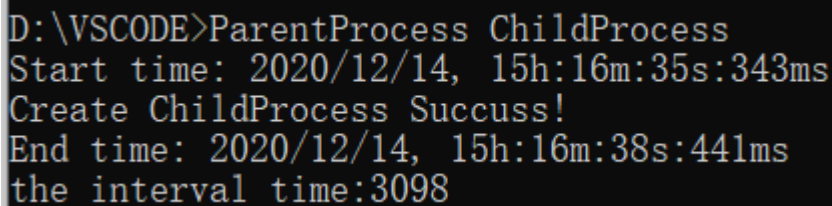
```
g++ -o ParentProcess.exe ParentProcess.cpp
```

```
g++ -o ChildProcess ChildProcess.cpp
```

再执行可执行文件：

```
ParentProcess.exe ChildProcess
```

代码运行后，首先弹出父进程窗口，显示 Start time 和 “Create ChildProcess Succuss!” 后弹出子进程窗口，子进程窗口显示 “Hi, my name is 唐小娟”，停留 3 秒后，子进程窗口自动关闭（子进程运行结束），父进程继续运行，父进程窗口显示 End time 和 interval time，之后父进程结束。结果如下：



```
D:\VSCODE>ParentProcess ChildProcess
Start time: 2020/12/14, 15h:16m:35s:343ms
Create ChildProcess Succuss!
End time: 2020/12/14, 15h:16m:38s:441ms
the interval time:3098
```

3. Linux 下的进程控制

打开终端，输入：

```
gcc -o ParentProcess.exe ParentProcess.c
```

```
gcc -o ChildProcess ChildProcess.c
```

```
/ParentProcess.exe ChildProcess
```

得到运行结果，首先打印子进程开始时间距离 1970 年 1 月 1 日的毫秒数时间，之

操作系统课程设计实验报告

后子进程运行，打印出“Hi, my name is 唐小娟.”，等待 3s 后，子进程结束，父进程继续运行，打印子进程结束时间和运行时间，父进程结束。结果如图：

```
ostxj@ostxj-virtual-machine:~/studycodes$ ./ParentProcess ChildProcess
The ChildProcess starts 1607670374062.
The childProcess is processing.
Hi, my name is 唐小娟.
The ChildProcess ends 1607670377065.
the childprocess interval 3003.
```

六、讨论、心得

在这次实验中，我学会了如何在 linux 下和 windows 下创建进程以及实现进程的同步，但是在这期间，我遇到了一些小问题：在 `pid==0` 的条件下获取 `start_time`，但是在父进程中是没有正确获得 `start_time` 的值的，这是因为子进程虽然继承父进程的代码和数据，但是如果子进程的运行过程中有发生数据的修改，系统会复制一份数据给子进程，父子进程并不共享这些空间，因此父进程无法“看到”子进程中的 `start_time`。