****

实验二 进程控制

班 级： 07111505

姓 名： 徐宇恒

学 号： 1120151839

目录

[一. 实验目的 3](#_Toc512106488)

[二. 实验内容 3](#_Toc512106489)

[2.1. 在Windows下的实现： 3](#_Toc512106492)

[2.2. 在Linux下的实现： 3](#_Toc512106493)

[三. 实验环境 3](#_Toc512106494)

[四. 实验过程 3](#_Toc512106495)

[4.1. 实验思路： 3](#_Toc512106498)

[4.1.1. Mytime(主进程) 3](#_Toc512106499)

[4.1.2. HelloWorld(子进程) 4](#_Toc512106500)

[4.2. Windows实现： 4](#_Toc512106501)

[4.2.1. 步骤 4](#_Toc512106502)

[4.2.2. 核心代码 4](#_Toc512106503)

[4.3. Linux实现： 5](#_Toc512106504)

[4.3.1. 步骤 5](#_Toc512106505)

[4.3.2. 核心代码 5](#_Toc512106506)

[五. 实验结果 6](#_Toc512106507)

[5.1. Windows运行结果 6](#_Toc512106509)

[5.2. Linux运行结果 6](#_Toc512106510)

[六. 心得体会 6](#_Toc512106511)

[七. 附录 7](#_Toc512106512)

# 实验目的

分别在Windows和Linux上实现Unix的time命令，取名为mytime

用mytime命令记录某可执行程序的运行时间，要求用命令行参数执行该可执行程序，并为该可执行程序创建一个独立的进程。

# 实验内容



## 在Windows下的实现：

* 使用CreateProcess()来创建进程。
* 使用WaitForSingleObiect()在“mytime”命令和新创建的进程之间同步。
* 调用GetSystemTime()获取时间。

## 在Linux下的实现：

* 使用fork()/execv()来创建进程运行程序。
* 使用wait()等待新创建的进程结束
* 调用gettimeofday()来获取时间

# 实验环境

|  |  |  |
| --- | --- | --- |
|  | Windows | Linux |
| 操作系统 | Windows10 Pro 64bit | Ubuntu 16.04LTS |
| 编译器 | Visual Studio 2017 IDE | Gcc |

# 实验过程



## 实验思路：

### Mytime(主进程)

* 运行mytime
* 记录开始时间
* 调用子进程
* 阻塞等待子进程结束
* 记录结束时间
* 输出子进程用时

### HelloWorld(子进程)

* 输出“HelloWorld”
* 阻塞等待用户操作

## Windows实现：

### 步骤

* 在cmd中运行主进程可执行文件mytime.exe
* Start = GetSystemTime获取开始时间
* CreateProcess创建子进程通过命令行传递参数调用子进程
* WaitForSingleObject阻塞等待子进程结束
* End = GetSystemTime获得结束时间
* End – start计算子程序用时

### 核心代码

|  |
| --- |
| GetSystemTime(&start);  if (!CreateProcess(  NULL,  argv[1],  NULL,  NULL,  FALSE,  CREATE\_NEW\_CONSOLE,  NULL,  NULL,  &si,  &pi)  )  {  printf("CreateProcess failed (%d).\n", GetLastError());  return 0;  }  WaitForSingleObject(pi.hProcess, INFINITE);  GetSystemTime(&end); |

## Linux实现：

### 步骤

* 编译生成可执行文件mytime
* 运行主进程
* 在主进程中调用fork()创建子进程
* 调用excev()函数运行可执行程序
* 主进程wait()阻塞等待子进程结束
* Gettimeofday()获取当前时间

### 核心代码

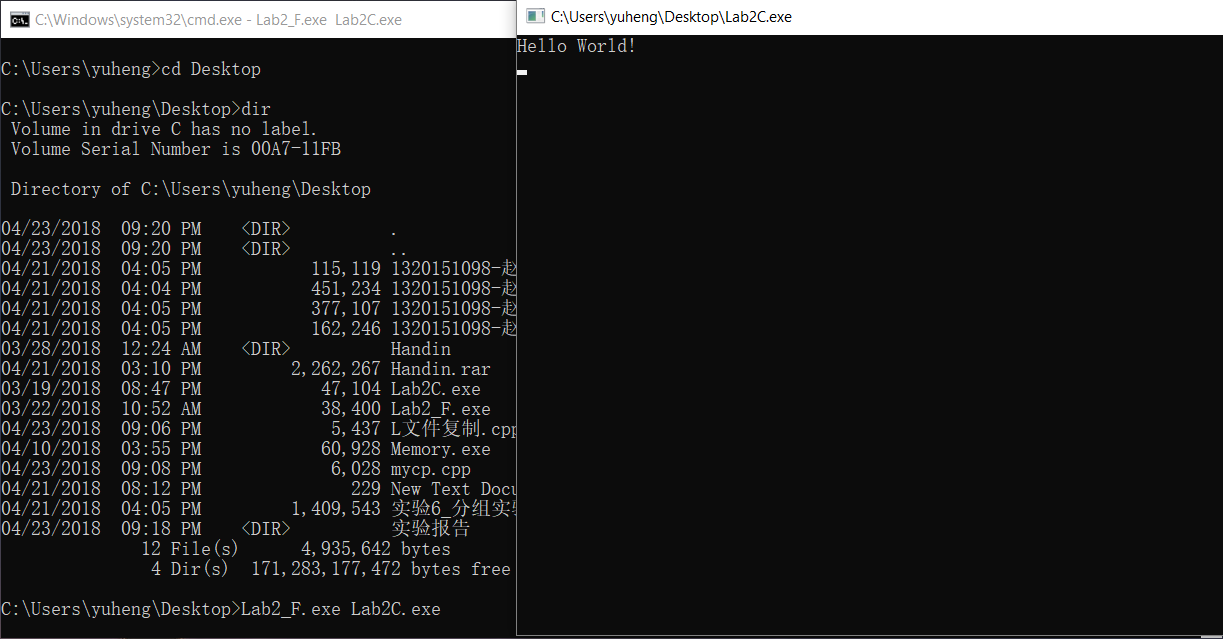
|  |
| --- |
| gettimeofday(&start, NULL);  pid\_t pid = fork();  execv(argv[1], arg);  wait(NULL);  gettimeofday(&end, NULL); |

# 实验结果

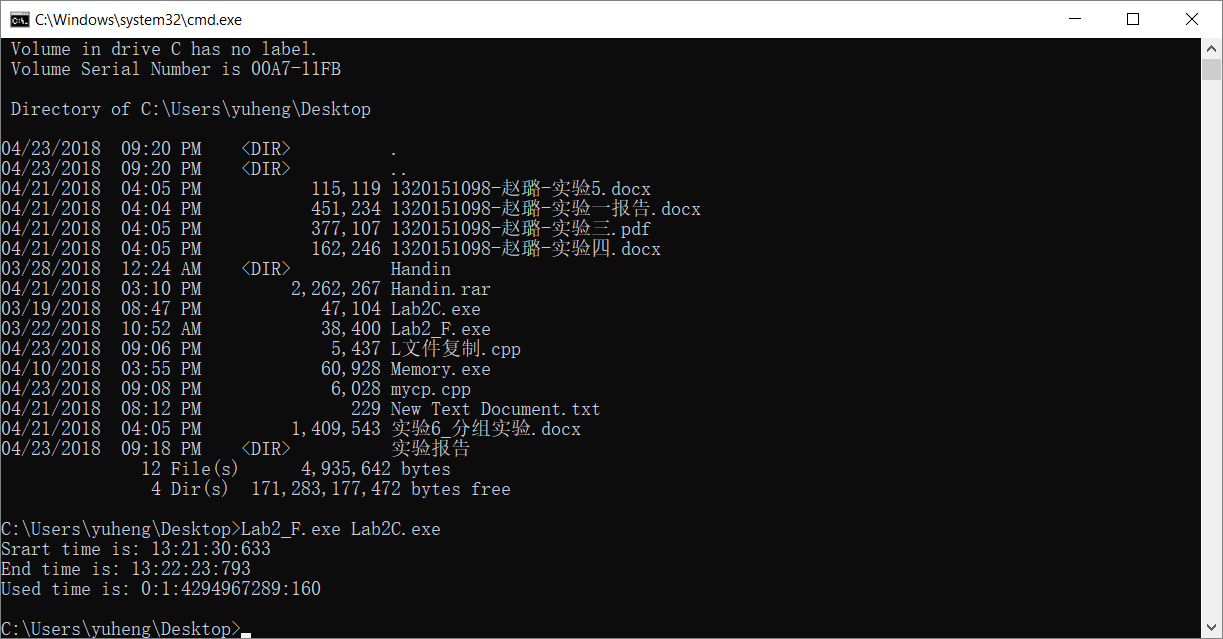


## Windows运行结果

在cmd中输入Lab\_F.exe Lab2C.exe调用子进程成功，等待用户操作

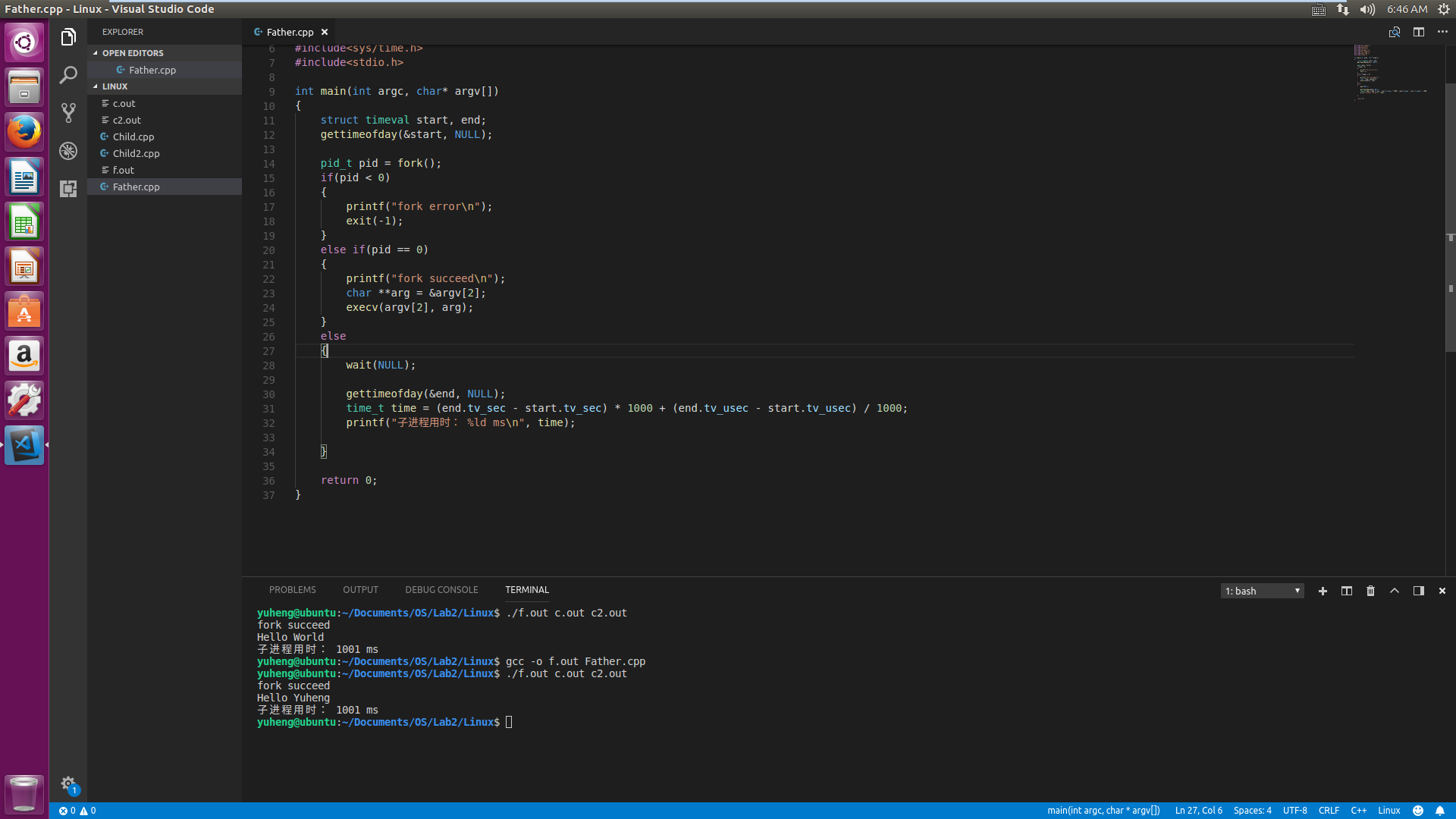


用户任意输入即可返回主进程，并显示子进程调用时间



## Linux运行结果

通过修改argv[1]和argv[2]，可以调用不同的子程序。如图所示



# 心得体会

通过本次实验，基本了解了Windows和Linux进程控制的基本方法，明白了如何在主进程中创建创建新的进程，并调用已经写好的可执行文件。

在本次实验中也遇到了很多问题。之前没用接触过WindowsAPI，对很多参数的含义并不是很明确不过通过阅读windows文档以及翻阅教科书逐渐弄明白整个进程的创建过程。

# 附录

* 1. Windows源码
     1. mytime.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #include<tchar.h>  #include<time.h>  int main(int argc, char \*argv[])  {  STARTUPINFO si;  PROCESS\_INFORMATION pi;    SYSTEMTIME start, end;  ZeroMemory0÷memset  ZeroMemory(&si, sizeof(si));  ZeroMemory(&pi, sizeof(pi));  si.cb = sizeof(si);  if (argc != 2)  {  printf("δú");  return 0;  }  GetSystemTime(&start);    if (!CreateProcess(  NULL,  argv[1],  NULL,  NULL,  FALSE,  CREATE\_NEW\_CONSOLE,  NULL,  NULL,  &si,  &pi)  )  {  printf("CreateProcess failed (%d).\n", GetLastError());  return 0;  }    WaitForSingleObject(pi.hProcess, INFINITE);  GetSystemTime(&end);  printf("Srart time is: %u:%u:%u:%u\n", start.wHour, start.wMinute, start.wSecond, start.wMilliseconds);  printf("End time is: %u:%u:%u:%u\n", end.wHour, end.wMinute, end.wSecond, end.wMilliseconds);  printf("Used time is: %u:%u:%u:%u\n", (end.wHour - start.wHour), (end.wMinute - start.wMinute), (end.wSecond - start.wSecond), (end.wMilliseconds - start.wMilliseconds));  CloseHandle(pi.hProcess);  } |

* + 1. hellow.cpp

|  |
| --- |
| #include<cstdio>  #include<iostream>  #include<cstdlib>  #include<Windows.h>  #include<conio.h>  using namespace std;  int main()  {  printf("Hello World!\n");  while (1)  {  if (kbhit() != 0)  break;  }  return 0;  } |

* 1. Linux源码
     1. mytime.cpp

|  |
| --- |
| #include<sys/types.h>  #include<unistd.h>  #include<stdlib.h>  #include<sys/wait.h>  #include<sys/types.h>  #include<sys/time.h>  #include<stdio.h>  int main(int argc, char\* argv[])  {  struct timeval start, end;  gettimeofday(&start, NULL);  pid\_t pid = fork();  if(pid < 0)  {  printf("fork error\n");  exit(-1);  }  else if(pid == 0)  {  printf("fork succeed\n");  char \*\*arg = &argv[1];  execv(argv[1], arg);  }  else  {  wait(NULL);  gettimeofday(&end, NULL);  time\_t time = (end.tv\_sec - start.tv\_sec) \* 1000 + (end.tv\_usec - start.tv\_usec) / 1000;  printf("子进程用时： %ld ms\n", time);  }  return 0;  } |

* + 1. helloWord.cpp

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<unistd.h>  int main()  {  printf("Hello World\n");  usleep(1000000);  return 0;  } |

* + 1. HelloYuheng.cpp

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<unistd.h>  int main()  {  printf("Hello Yuheng\n");  usleep(1000000);  return 0;  } |