

北京理工大学

操作系统课程设计

实验二、程序控制	实验二，过程控制
----------	----------

学院：计算机学院
专业：计算机科学与技术
学生姓名：夏奇拉
学号：1820171025
：07111705

目录

目的.....	3
问题讨论.....	3
执行 [Windows]	5
结果与分析 [Windows]	9
执行 [Linux]	10
结果与分析[Linux]	15
参考：	16

目的

为 Windows 和 Linux 设计和实现“ParentProcess”命令。

“父进程”命令

- 接受通过命令行参数运行的程序，
- 创建一个单独的进程来运行该程序，并且
- 记录程序运行的时间。

Windows 下实现：

- 使用 CreateProcess() 创建进程
- 使用 WaitForSingleObject() 在“ParentProcess”命令和新创建的进程之间进行同步
- 调用 GetSystemTime() 获取时间

Linux 下实现：

- 使用 fork()/execv() 创建进程运行器
- 使用 wait()等待新创建的进程结束
- 调用 gettimeofday() 获取时间

要求：

- 父进程：
 - 名称：父进程 (.exe)
 - 功能：启动 ChildProcess(.exe)
 - 打印子进程开始时间，结束时间，耗时，精确到 ms
- 子过程：
 - 名称：子进程 (.exe)
 - 功能：打印：嗨，我的名字是+你的名字
 - 延时 3s

父进程的用法：

\$ ParentProcess.exe 子进程

问题讨论

我们需要为 Windows 和 Linux 设计和实现 “ParentProcess”命令。

什么是 “parentprocess”命令？

可以单次、多次或同时执行的指令和数据序列称为程序。一个过程就是这些程序的执行。父进程是创建了一个或多个子进程的进程。子进程是由另一个进程（父进程）创建的进程。

对于这个实验，父进程是 parentprocess.exe，子进程是 childprocess.exe。这些可执行文件可以通过分别编译 parentprocess.c 和 childprocess.c 源文件来创建。

根据实验要求，父进程命令必须通过命令行参数接受一个程序才能运行。这可以通过在操作系统命令行 shell 中的程序（可执行 .exe 文件）名称后添加命令来完成。例如：

在 Linux 下	
\$ nano parantprocess.c	//创建源文件
\$ gcc 父进程.c -o 父进程	//编译源代码并创建一个名为 parentprocess.exe 的可执行文件
\$./parentprocess.exe 子进程	//运行父进程程序并将子进程命令作为参数传递给它

在 C 编程中接受命令行参数

在 C 编程中，可以通过 main() 的声明接受命令行参数。

Main() 可以接受两个参数。

1. 命令行参数的数量。
2. 所有命令行参数的完整列表。

main 的完整声明如下所示：

```
int main (int argc, char *argv[])
```

Windows 特定版本如下所示：（_T（或 _t）约定是 Microsoft 用来允许构建 Unicode 和构建非 Unicode（通常是多字节字符集）而无需更改源代码的约定。）

```
int _tmain (int argc, _TCHAR *argv[])
```

整数 argc 是**参数计数**，从命令行传递给程序的参数数量，包括程序名称。

字符指针数组是所有参数的列表。Argv[0] 是程序的名称，如果名称不可用，则为空字符串。之后，每一个小于 argc 的元素号都是一个命令行参数。

在上面给出的示例中：

```
$ ./parentprocess.exe 子进程 //运行父进程程序并将子进程命令作为参数传递给它
```

argc 将等于 2，argv[0] 将是 ./parentprocess，而 argv[1] 将是 childprocess

创建一个单独的进程来运行程序

在 Windows 和 Linux 中创建子进程是不同的。让我们看看如何在 Windows 和 Linux 中执行此实验的要求。

执行 [视窗]

按照此处列出的步骤设置用于编译 C 程序的 Windows 环境：[演练：在命令行上编译 C 程序 | 微软学习](#)。

第 1 步：创建 parentprocess.c 程序

在 Visual Studio 2022 的开发人员命令提示中，使用记事本创建一个名为 parentprocess.c 的 C 文件，方法是运行以下命令：

```
> 记事本 parentprocess.c
```

在文件中，写入以下代码：

```
// 父进程.c

#include <windows.h>
#include <stdio.h>
#include <tchar.h>

void _tmain(int argc, TCHAR *argv[])
{
    系统时间 st, et; // 指向 SYSTEMTIME 结构的指针以接收当前系统日期和时间。

    启动信息 si;
    PROCESS_INFORMATION pi;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    if (argc != 2) /* argc 应该是 2 才能正确执行 */
    {
        printf("用法: %s [cmdline]\n", argv[0]);
        返回;
    }

    // 启动子进程。
    if (!CreateProcess(NULL, // 没有模块名（使用命令行）
        argv[1], // 命令行
        NULL, // 进程句柄不可继承
        NULL, // 线程句柄不可继承
        FALSE, // 将句柄继承设置为 FALSE
        0, // 没有创建标志
        NULL, // 使用父级的环境块
```

```

NULL, // 使用父级的起始目录
&si, // 指向 STARTUPINFO 结构的指针
&pi) // 指向 PROCESS_INFORMATION 结构的指针
)
{
printf("CreateProcess 失败 (%d).\n", GetLastError());
返回;
}

获取系统时间(&st);
printf("子进程启动时间为: %02dh:%02dm:%02ds.%02dms\n", st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);

// 等到子进程退出。
WaitForSingleObject(pi.hProcess, INFINITE);

获取系统时间 (&et);
printf("子进程结束时间为: %02dh:%02dm:%02ds.%02dms\n", et.wHour, et.wMinute, et.wSecond, et.wMilliseconds);

printf("子进程运行时间为: %02ds.%02dms\n", et.wSecond-st.wSecond, et.wMilliseconds-st.wMilliseconds);

// 关闭进程和线程句柄。
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
}

```

头文件 windows.h 是 Win32 编程的基础头文件。它包含 Windows API 中所有函数的声明，Windows 程序员使用的所有常用宏，例如 HANDLE、CreateProcess 等。通过包含 <windows.h> 头文件，可以将 Win32 API 添加到 C 编程项目中。

在此处阅读有关 tchar.h 的更多信息：[tchar.h 中的通用文本映射 | 微软学习](#)

CreateProcess() 函数创建一个新进程，该进程独立于创建进程运行。然而，为了简单起见，该关系被称为父子关系。

如果 CreateProcess 成功，它会返回一个 PROCESS_INFORMATION 结构，其中包含新进程及其主线程的句柄和标识符。

CreateProcess 函数将指向 STARTUPINFO 结构的指针作为其参数之一。STARTUPINFO 结构在创建时指定进程的窗口站、桌面、标准句柄和主窗口的外观。

该结构的成员用于指定子进程主窗口的特性。对于控制台进程，STARTUPINFO 结构仅在创建新控制台时用于指定窗口属性（使用带有 CREATE_NEW_CONSOLE 的 CreateProcess 或使用 AllocConsole 函数）。STARTUPINFO 结构可用于指定以下控制台窗口属性：

- 新控制台窗口的大小，以字符单元格为单位。
- 新控制台窗口的位置，以屏幕坐标表示。
- 新控制台屏幕缓冲区的大小（以字符单元为单位）。
- 新控制台屏幕缓冲区的文本和背景颜色属性。
- 新控制台窗口的标题。

STARTUPINFO 的成员：[STARTUPINFOA \(processthreadsapi.h\) - Win32 应用程序 | 微软学习](#)

PROCESS_INFORMATION 结构包含有关新创建的进程及其主线程的信息。其成员包括 hProcess、hThread、dwProcessId、dwThreadId

注意：从本质上讲，句柄是一个抽象指针，指针通常被定义为“指向资源内存地址的特殊变量”

ZeroMemory() 是一个用零填充内存块的宏。它的参数包括 *destination[in]*，指向要用零填充的内存块起始地址的指针和 *length[in]* 要用零填充的内存块的大小（以字节为单位）。这个宏没有返回值。

sizeof 运算符返回给定类型的变量占用的字节数。

cb 是 STARTUPINFO 的成员，它是结构的大小（以字节为单位）。

使用 WaitForSingleObject() 在“父进程”命令和新创建的进程之间进行同步

WaitForSingleObject 函数等待，直到指定的对象处于信号状态或超时间隔结束。它接受对象的句柄和以毫秒为单位的超时间隔。如果指定了非零值，则函数会一直等待，直到对象收到信号或间隔结束。如果 dwMilliseconds 为零，则如果对象未发出信号，则函数不会进入等待状态；它总是立即返回。如果 dwMilliseconds 是 INFINITE，则该函数将仅在对象发出信号时返回。

调用 GetSystemTime() 获取时间

GetSystemTime 函数以 UTC 格式检索当前系统日期和时间。该函数接受指向 SYSTEMTIME 结构的指针以接收当前系统日期和时间。

第 2 步：创建 childprocess.c 程序

在 Visual Studio 2022 的开发人员命令提示中，使用记事本创建一个名为 childprocess.c 的 C 文件，方法是运行以下命令：

```
> 记事本 childprocess.c
```

在文件中，写入以下代码：

```
// 子进程.c

#include <stdio.h>
#include <windows.h>

主函数()
{
printf("你好我叫夏奇拉\n");
printf("延时 3s\n");
睡眠 (3000) ;
返回 0;
}
```

Sleep() 函数暂停当前线程的执行，直到超时间隔结束。它接受暂停执行的时间间隔（以毫秒为单位）。

第三步：编译 C 程序

使用以下命令制作两个程序的可执行版本。

```
> cl 父进程.c
> cl 子进程.c
```

这将输出 parentprocess.exe 和 childprocess.exe

第四步：运行程序

使用以下命令运行新创建的可执行 .exe 程序。

```
> 父进程子进程
```

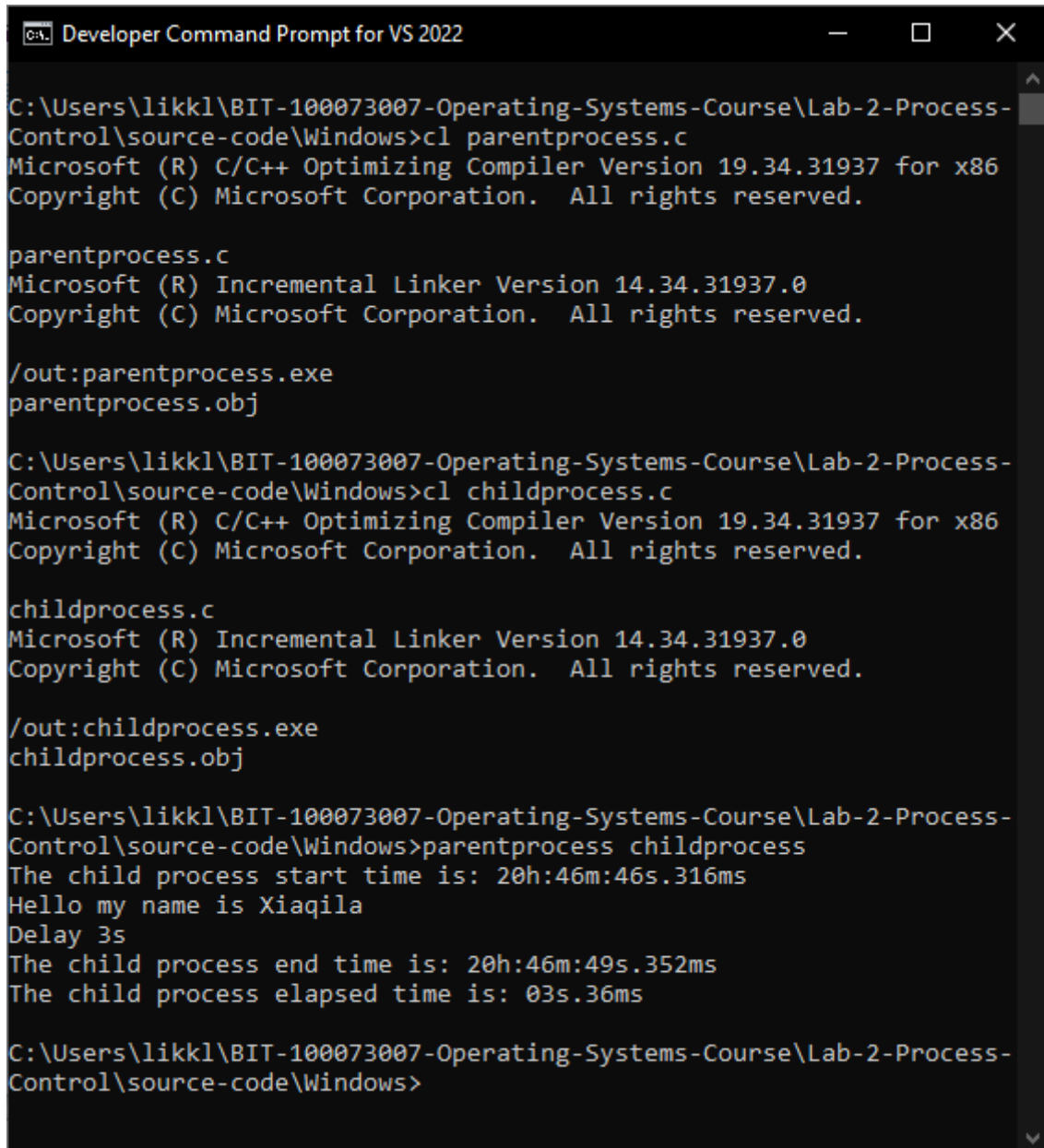
这些参数传递给 `void _tmain(int argc, TCHAR *argv[])`

产生以下输出：

```
子进程启动时间为：20h:46m:46s.316ms
你好我叫夏奇拉
延时 3s
子进程结束时间为：20h:46m:49s.352ms
子进程运行时间为：03s.36ms
```


结果与分析 [Windows]

在命令行中运行 parentprocess childprocess 会启动父进程程序。该程序使用 CreateProcess() 调用打印 “Hello my name is Xiaqila”的子进程程序并休眠 3 秒后返回。父进程使用 WaitForSingleObject() 等待子进程结束。子进程完成后，父进程打印子进程结束时间和子进程运行时间。



```
Developer Command Prompt for VS 2022

C:\Users\likkl\BIT-100073007-Operating-Systems-Course\Lab-2-Process-
Control\source-code\Windows>cl parentprocess.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.34.31937 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

parentprocess.c
Microsoft (R) Incremental Linker Version 14.34.31937.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:parentprocess.exe
parentprocess.obj

C:\Users\likkl\BIT-100073007-Operating-Systems-Course\Lab-2-Process-
Control\source-code\Windows>cl childprocess.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.34.31937 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

childprocess.c
Microsoft (R) Incremental Linker Version 14.34.31937.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:childprocess.exe
childprocess.obj

C:\Users\likkl\BIT-100073007-Operating-Systems-Course\Lab-2-Process-
Control\source-code\Windows>parentprocess childprocess
The child process start time is: 20h:46m:46s.316ms
Hello my name is Xiaqila
Delay 3s
The child process end time is: 20h:46m:49s.352ms
The child process elapsed time is: 03s.36ms

C:\Users\likkl\BIT-100073007-Operating-Systems-Course\Lab-2-Process-
Control\source-code\Windows>
```

执行 [Linux]

叉子 ()

系统调用 `fork()` 用于创建进程。它不接受任何参数并返回一个进程 ID。`fork()` 的目的是创建一个新进程，成为调用者的子进程。创建新的子进程后，两个进程将执行 `fork()` 系统调用之后的下一条指令。因此，我们必须区分父母和孩子。这可以通过测试 `fork()` 的返回值来完成：

- 如果 `fork()` 返回负值，则子进程的创建不成功。
- `fork()` 向新创建的子进程返回一个零。
- `fork()` 向父进程返回一个正值，即子进程的进程 ID。返回的进程 ID 是在 `sys/types.h` 中定义的 `pid_t` 类型。通常，进程 ID 是一个整数。此外，进程可以使用函数 `getpid()` 来检索分配给该进程的进程 ID。

因此，在系统调用 `fork()` 之后，一个简单的测试就可以知道哪个进程是子进程。Unix 将精确复制父地址空间并将其提供给子地址。因此，父进程和子进程具有独立的地址空间。

等待 ()

系统调用 `wait()` 很简单。此函数会阻塞调用进程，直到其子进程之一退出或收到信号。为了我们的目的，我们将忽略信号。`wait()` 获取整数变量的地址并返回已完成进程的进程 ID。一些指示子进程完成状态的标志与整数指针一起传回。`wait()` 的主要目的之一是等待子进程完成。

`wait()` 的执行可能有两种情况。

如果在调用 `wait()` 时至少有一个子进程正在运行，则调用者将被阻塞，直到其子进程之一退出。在那一刻，调用者恢复执行。

如果调用 `wait()` 时没有子进程在运行，那么这个 `wait()` 根本没有作用。也就是说，就好像没有 `wait()` 一样。

执行()

创建的子进程不必运行与父进程相同的程序。`exec` 类型的系统调用允许进程运行任何程序文件，包括二进制可执行文件或 shell 脚本。对于这个实验，使用了 `execv()`。`execv()` 系统调用需要两个参数：

- 按照惯例，第一个参数应该指向与正在执行的文件关联的文件名。
- `char *const argv[]` 参数是指向空终止字符串的指针数组，表示新程序可用的参数列表，它与主程序中使用的 `argv` 数组完全相同：

`int main(int argc, char **argv)`

第一个参数 `argc` (argument count) 是一个整数，表示程序启动时在命令行上输入了多少个参数。第二个参数 `argv` (参数向量) 是指向字符对象数组的指针数组。数组对象是以 `null` 结尾的字符串，表示程序启动时在命令行中输入的参数。

数组的第一个元素 `argv[0]` 是指向包含程序名称或从命令行运行的程序的调用名称的字符数组的指针。`argv[1]` 表示传递给程序的第一个参数，`argv[2]` 表示第二个参数，依此类推。

请注意，此参数必须以零结尾。

如果执行失败（例如，请求文件不存在），`execv()` 将返回一个负值。

获取时间 ()

`gettimeofday()` 系统函数在 `sys/time.h` 头文件中定义。它用有关当前时间的详细信息填充两个结构：

```
int gettimeofday( 结构 timeval *, 结构 tzp *);
```

`timeval` 结构包含两个成员，

- `tv_sec`：一个 `time_t` 值，从 1970 年 1 月 1 日开始经过的秒数。
- `tv_usec`：微秒值，计算机知道但不包含在 `time_t` 值中

`tzp` 结构包含时区信息。

这将涉及编写一个 C 程序并构建一个可执行文件以作为父进程在终端中运行

第 1 步：创建 C 程序

在终端中，通过运行以下命令使用名称为 `parenprocess.c` 的 `nano` 编辑器创建一个文件：

```
$ nano 父进程.c
```

在文件中我们编写如下代码：

```
#include <stdio.h>
#include <unistd.h>
#include <标准库.h>
#include <字符串.h>
#include <sys/wait.h>
#include <sys/time.h>

int main (int argc, char *argv[]) {

    结构 timeval 开始，结束，start2;

    pid_t pid;
    pid = fork();

    如果 (pid==0) {
        int sts_cd = execv("/home/shak/Documents/BIT-100073007-Operating-Systems-Course/Lab-2-Process-Control/Source-Code/Linux/childprocess", argv);
        printf("运行命令有问题\n");
        如果 (sts_cd == -1) {
            printf("执行错误! \n");
            返回 1;
        }
    }
    否则如果 (pid > 0) {
```

```

        gettimeofday(&开始, NULL);
        printf("子进程启动时间为: %ld 秒%ld 微秒\n", start.tv_sec, start.tv_usec);
        int wc = wait(NULL);
        gettimeofday(&end, NULL);
        printf("子进程结束时间为: %ld 秒%ld 微秒\n", end.tv_sec, end.tv_usec);
        printf("子进程运行时间为: %lds.%ldms\n", (end.tv_sec - start.tv_sec), (end.tv_usec - start.tv_usec) / 1000);
    }
    别的 {
        printf("分叉时出错\n");
        退出 (退出失败);
    }

    返回 0;
}

```

关于代码：

在这个节目中，我们

- 声明结构 timeval start end 以捕获从 gettimeofday() 返回的值。
- 将字符 cmd 声明为“ls”作为运行 int execvp(const char *file, char *const) 时要执行的文件。
- 声明在 sys/types.h 中定义的 pid_t 类型的进程 ID，以捕获我们运行 fork() 时返回的进程 ID。
- 运行 fork() 创建一个子进程，它是父进程的精确副本。我们进行一个简单的测试来判断哪个 process@id:ms-vscode.cpptools-extension-pack 是孩子。
- 如果 fork() 返回 0，则执行子进程。它包括用于在源目录中运行 childprocess.exe 程序的 execv() 函数。childprocess.exe 程序打印“Hello my name is Xiaqila”并休眠 3 秒后返回。
- 如果 fork() 返回一个大于 0 的值，则父进程继续。我们通过调用 gettimeofday() 来捕获进程的开始时间。然后父进程在调用 wait(NULL) 时等待子进程完成。
- wc 捕获父进程正在等待的子进程的进程 ID。
- 当子进程完成时，我们使用 gettimeofday() 来捕获子进程的结束时间。然后父进程打印子进程的开始时间、结束时间和运行时间。

通过点击退出 nano 编辑器

CTRL + X

是 (是)

进入

CTRL + X

这使我们返回到终端

第 3 步：创建 childprocess.c 程序

在终端中，通过运行以下命令使用名称为 childprocess.c 的 nano 编辑器创建一个文件：

```
$ nano 子进程.c
```

在文件中我们编写如下代码：

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

主函数()
{
printf("你好我叫夏奇拉\n");
printf("延时 3s\n");
睡觉 (3) ;
返回 0;
}
```

Sleep() 函数暂停当前线程的执行，直到超时间隔结束。它接受以秒为单位暂停执行的时间间隔。

第四步：编译 C 程序

使用以下命令制作两个程序的可执行版本。

```
$ gcc 父进程.c -o 父进程
$ gcc 子进程.c -o 子进程
```

这将输出 parentprocess.exe 和 childprocess.exe

第五步：运行程序

使用以下命令运行新创建的可执行 .exe 程序。

```
$ ./parentprocess 子进程
```

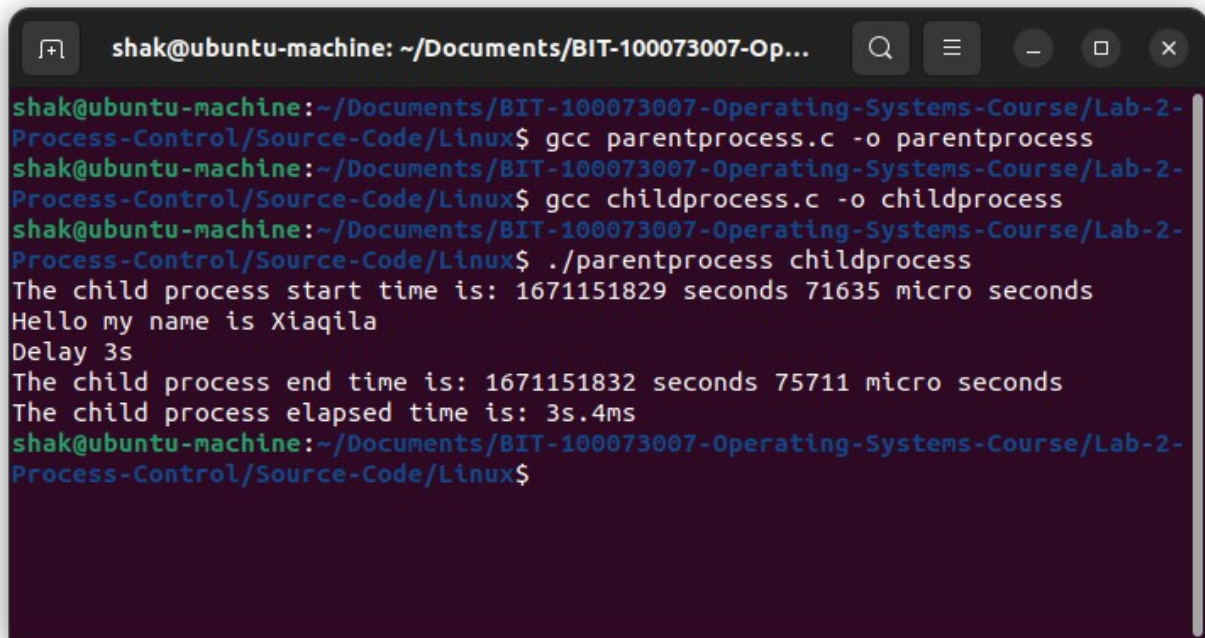
这些参数传递给 `int main (int argc, char *argv[])`

产生以下输出：

```
子进程启动时间为：1671151558 秒 641829 微秒
你好我叫夏奇拉
延时 3s
子进程结束时间为：1671151561 秒 644295 微秒
子进程运行时间为：3s.2ms
```

结果与分析[Linux]

在命令行运行 `./parentprocess childprocess` 会启动父进程程序。该程序使用 `fork()` 和 `execv()` 调用打印 “Hello my name is Xiaqila” 的子进程程序并休眠 3 秒后返回。父进程使用 `wait()` 等待子进程结束。子进程完成后，父进程打印子进程结束时间和子进程运行时间。



```
shak@ubuntu-machine: ~/Documents/BIT-100073007-Op...
shak@ubuntu-machine:~/Documents/BIT-100073007-Operating-Systems-Course/Lab-2-Process-Control/Source-Code/Linux$ gcc parentprocess.c -o parentprocess
shak@ubuntu-machine:~/Documents/BIT-100073007-Operating-Systems-Course/Lab-2-Process-Control/Source-Code/Linux$ gcc childprocess.c -o childprocess
shak@ubuntu-machine:~/Documents/BIT-100073007-Operating-Systems-Course/Lab-2-Process-Control/Source-Code/Linux$ ./parentprocess childprocess
The child process start time is: 1671151829 seconds 71635 micro seconds
Hello my name is Xiaqila
Delay 3s
The child process end time is: 1671151832 seconds 75711 micro seconds
The child process elapsed time is: 3s.4ms
shak@ubuntu-machine:~/Documents/BIT-100073007-Operating-Systems-Course/Lab-2-Process-Control/Source-Code/Linux$
```

参考：

- <https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html>
- <https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/wait.html>
- <http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/exec.html>
- <https://vitux.com/fork-exec-wait-and-exit-system-call-explained-in-linux/>
- <https://nipunbatra.github.io/os2020/labs/>
- <https://www.ibm.com/docs/en/i/7.1?topic=functions-main-function>
- https://linuxhint.com/gettimeofday_c_language/
- <https://c-for-dummies.com/blog/?p=4236>
- <https://linuxhint.com/execvp-function-c/>
- <https://www.cyberciti.biz/faq/compiling-c-program-and-creating-executable-file/>
- [C 中的命令行参数 - Cprogramming.com](#)
- [创建进程 - Win32 应用程序 | 微软学习](#)
- [WaitForSingleObject 函数 \(synchapi.h\) - Win32 应用程序 | 微软学习](#)
- [休眠功能 \(synchapi.h\) - Win32 应用程序 | 微软学习](#)
- [GetSystemTime 函数 \(sysinfoapi.h\) - Win32 应用程序 | 微软学习](#)
- [ZeroMemory 宏 \(Windows\) | 微软学习](#)
- [STARTUPINFOA \(processthreadsapi.h\) - Win32 应用程序 | 微软学习](#)
- [PROCESS_INFORMATION \(processthreadsapi.h\) - Win32 应用程序 | 微软学习](#)
- <https://man7.org/linux/man-pages/man3/exec.3.html>