# 实验二 进程控制

徐恒达 1120151811

## 目录

# 1 实验目的

设计并分别在 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。要求用命令行参数接受某可执行程序，并为该可执行程序创建一个独立的进程。

# 2 实验内容

在 Windows 下实现：

- 使用 CreateProcess() 来创建进程
- 使用 WaitForSingleObject() 在“mytime”命令中和新创建的进程之间同步
- 调用 GetSystemTime() 来获取时间

在 Linux 下实现：

- 使用 fork() 和 execv() 来创建进程并运行新的可执行程序
- 使用 wait() 等待新创建的进程结束
- 调用 gettimeofday() 来获取时间

# 3 实验环境

## 3.1 硬件配置

处理器：Intel(R) Core(TM) i3-4020Y CPU @ 1.5GHz
内存：4.0GB DDR3

## 3.2 Windwos 配置

Windows 10 Pro Version 1709 64bit

## 3.3 Linux 配置

Ubuntu 16.04 LTS

# 4 实验过程

## 4.1 实现思路

在主进程中创建子进程并记录开始时间，并将所需的命令行参数传递给子进程，阻塞等待子进程执行结束后得到结束时间。两个时间相减即得到子程序的执行时间。

主进程程序 mytime 的伪代码如下所示。

**Algorithm 1** mytime

**if** 用户没有指定子进程 **then**
    输出错误信息
**else**
    记录开始时间 $t_{start}$
    启动子进程
    阻塞等待子进程结束
    记录结束时间 $t_{end}$
    输出子进程用时 $t_{end} - t_{start}$
**end if**

子进程程序 mysleep 的伪代码如下所示。

**Algorithm 2** mysleep

**if** 用户指定了睡眠时间 $x$ **then**
    睡眠时间 $t \leftarrow x$
**else**
    $t \leftarrow 0$ 到 1000 间的一个随机值
**end if**
睡眠 $t$ 毫秒

## 4.2 Windows 实现

Windows 系统中 mytime 程序的核心代码如下所示。

```
DWORD time = GetTickCount();
CreateProcess(
    NULL, (LPTSTR)arg, NULL, NULL,
    FALSE, 0, NULL, NULL, &si, &pi);
WaitForSingleObject(pi.hProcess, INFINITE);
```

```
time = GetTickCount() - time;
```

其中CreateProcess()函数负责创建子进程并执行，函数声明如下。

```
BOOL WINAPI CreateProcess(
   _In_opt_  LPCTSTR            lpApplicationName,
   _Inout_opt_ LPTSTR           lpCommandLine,
   _In_opt_  LPSECURITY_ATTRIBUTES lpProcessAttributes,
   _In_opt_  LPSECURITY_ATTRIBUTES lpThreadAttributes,
   _In_      BOOL               bInheritHandles,
   _In_      DWORD              dwCreationFlags,
   _In_opt_  LPVOID             lpEnvironment,
   _In_opt_  LPCTSTR            lpCurrentDirectory,
   _In_      LPSTARTUPINFO      lpStartupInfo,
   _Out_     LPPROCESS_INFORMATION lpProcessInformation
   );
```

**lpApplicationName:** 要执行的模块名称。

**lpCommandLine:** 要执行的命令行参数。

**lpProcessAttributes:** 一个指向SECURITY_ATTRIBUTES结构体的指针，这个结构体中的信息指明了函数返回的进程句柄是否可以被当前进程的子进程继承，这里赋值为空指针NULL，表示不被继承。

**lpThreadAttributes:** 同lpProcessAttributes，指定线程句柄的可继承性，值为NULL表示不继承。

**bInheritHandles:** 如果其值为TRUE，则子进程继承父进程的句柄表，FALSE表示不继承。

**dwCreationFlags:** 指定附加的、用来控制优先类和进程的创建的标志。

**lpEnvironment:** 指向一个新进程的环境块，如果此参数为空，则使用新的环境。

**lpCurrentDirectory:** 指定子进程的工作路径。

**lpStartupInfo:** 指向STARTUPINFO结构体，为启动信息。

**lpProcessInformation:** 指向PROCESS_INFORMATION结构体，返回进程和线程信息。

WaitForSingleObject()函数阻塞等待对象直至对象为有信号状态或超过指定的时间，函数声明如下。

```
DWORD WINAPI WaitForSingleObject(
    _In_ HANDLE hHandle,
    _In_ DWORD dwMilliseconds
    );
```

GetTickCount()函数返回自程序启动开始经过的毫秒数，函数原型如下。

```
DWORD WINAPI GetTickCount(void);
```

完整代码见附录。

## 4.3 Linux 实现

Linux 系统中 mytime 程序的核心代码如下所示。

```
pid_t pid = fork();
struct timeval start, end;
gettimeofday(&start, NULL);
if (pid == 0) execv(argv[1], &argv[1]);
waitpid(pid, NULL, 0);
gettimeofday(&end, NULL);
int time = (end.tv_sec - start.tv_sec) * 1000
    + (end.tv_usec - start.tv_usec) / 1000;
```

fork()函数通过复制自身来创建一个新的进程，这个新的进程通常被称为子进程，基本上只是当前进程的一个复制。fork 函数是一个双返回函数，子进程则返回 0，父进程返回子进程的进程标识符 pid。函数声明如下。

```
pid_t fork(void);
```

exec()系列的函数将指定程序的执行体覆盖到当前进程。execv(),execvp() 和 execvpe() 函数中都含有一个指向数组指针的参数，可以将命令行参数通过此方式传递进去。按照惯例，第一个函数指针应指向要被执行的文件，且指针数组必须以 NULL 指针作为结尾。函数声明如下。

```
int execv(const char *path, char *const argv[]);
```

wait()函数阻塞等待，直到调用进程的任意一个子进程终止运行。调用成功后返回终止子进程的 PID 码并将状态的说明放入 STAT_LOC 参数指明的地址，如果发生错误则返回-1。函数声明如下。

```
pid_t wait(int *wstatus);
```

gettimeofday()函数取得自 19700101 00:00:00 +0000 (UTC) 以来所经过的时间，放在 tv 所指向的结构体中。函数原型如下。

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

结构体 tv 的定义如下。

```
struct timeval {
    time_t      tv_sec;
    suseconds_t tv_usec;
};
```

tv_sec字段记录秒数，tv_usec记录微秒数。

完整代码见附录。

# 5 实验结果

分别编译主进程文件 mytime.c 和子进程文件 mysleep.c，makefile 文件内容如下所示。为了在一个 make 中生成两个目标文件，将 mytime 和 mysleep 共同指向一个根节点 all，并将其设为虚（phony）根节点。

```
all: mytime mysleep
.phony: all
mytime: mytime.c
    gcc mytime.c -o mytime
mysleep: mysleep.c
    gcc mysleep.c -o mysleep
```

## 5.1 Windows 实验结果

在命令行中运行 mytime，把子进程名和子进程参数传入，记录运行时间。

```
> mytime mysleep 2000
```

运行结果如下。可以看出，子进程核心部分暂停 2000 毫秒，但因为进程切换以及函数调用等开销，mytime 测出的运行时间略大于 2000 毫秒。

```
Mysleep started.
Sleep 2000 ms...
Mysleep finished.
Time: 2078 ms
```

mytime 的容错性测试。当没有指定子进程名是，mytime 输出错误信息，并提示正确的使用格式。

```
> mytime
The syntax of the command is incorrect.
Usage: mytime <FILE> [OPTION]
```

### 5.2 Linux 实验结果

Linux 环境下的测试与 Windows 相同，mytime 测得的运行时间比 mysleep 中的暂停时间稍大。

```
$ mytime mysleep 2000
Mysleep started.
Sleep 2000 ms...
Mysleep finished.
Time: 2005 ms
```

## 6 心得体会

通过本次实验，我基本掌握了 Windows 和 Linux 上的进程创建技巧，可利用多进程完成日常中的一些任务，以至于充分利用现代计算机的多核特性，受益良多。

不过过程中也遇到一些问题，问题主要出在 Windows 编程上，由于对 Windows API 的了解不多，导致过程中出了很多问题。我对 Windows 编程中定义的一些类型很不了解，例如 LPWSTR，LPCWSTR 等等，导致参数设置错误。

像 CreateProcess 这个函数，它的前两个参数 lpApplicationName，lpCommandLine 的类型就分别为 LPCWSTR 和 LPWSTR。对于 LPCWSTR 是指指向常量宽字符串的长指针，而 LPWSTR 则是指向宽字符串的长指针。虽然两者区别仅有一个“常量”，但是事实上差异巨大。通过不断查阅资料，我逐渐掌握了获取函数说明信息的能力，切实体会到了标准化的好处，可以为开发人员提供极大的便利。

# A 附录

## A.1 Windows 版本源代码

### A.1.1 mytime.c

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

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf("The syntax of the command is incorrect.\n");
        printf("Usage: mytime <FILE> [OPTION]\n");
        return 1;
    }

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

    char arg[1024];
    strcpy(arg, argv[1]);
    for (int i = 2; i < argc; i++)
    {
        strcat(arg, " ");
        strcat(arg, argv[i]);
    }

    DWORD time = GetTickCount();
    BOOL bSucceed = CreateProcess(
            NULL, (LPTSTR)arg, NULL, NULL,
            FALSE, 0, NULL, NULL, &si, &pi);
    if (!bSucceed)
```

```
    {
        printf("Create process '%s' failed!\n", argv[1]);
        return 1;
    }
    WaitForSingleObject(pi.hProcess, INFINITE);
    time = GetTickCount() - time;

    printf("\nTime: %d ms\n", time);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}
```

### A.1.2 mysleep.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

int main(int argc, char *argv[])
{
    printf("Mysleep started.\n");
    int t = 0;
    if (argc > 1) t = strtol(argv[1], NULL, 10);
    if (t == 0)
    {
        srand(time(NULL));
        t = rand() % 1001;
    }
    printf("Sleep %d ms...\n", t);
    Sleep(t);
    printf("Mysleep finished.\n");
    return 0;
}
```

### A.1.3 makefile

```
all: mytime mysleep
.phony: all
mytime: mytime.c
    gcc mytime.c -o mytime
mysleep: mysleep.c
    gcc mysleep.c -o mysleep
```

## A.2 Linux 版本源代码

### A.2.1 mytime.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/time.h>

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf("mytime: missing operand\n");
        printf("Usage: mytime <FILE> [OPTION]\n");
        return 0;
    }

    pid_t pid = fork();
    struct timeval start, end;
    gettimeofday(&start, NULL);

    if (pid == 0)
    {
        int error = execv(argv[1], &argv[1]);
        if (error == -1)
        {
            printf("Execute '%s; failed!\n", argv[1]);
```

```
            return 1;
        }
    }

    waitpid(pid, NULL, 0);
    gettimeofday(&end, NULL);
    int time = (end.tv_sec - start.tv_sec) * 1000
             + (end.tv_usec - start.tv_usec) / 1000;
    printf("\nTime: %d ms\n", time);
    return 0;
}
```

### A.2.2 mysleep.c

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

int main(int argc, char *argv[])
{
    printf("Mysleep started.\n");
    int t = 0;
    if (argc > 1) t = strtol(argv[1], NULL, 10);
    if (t == 0)
    {
        srand(time(NULL));
        t = rand() % 1001;
    }
    printf("Sleep %d ms...\n", t);
    usleep(t * 1000);
    printf("Mysleep finished.\n");
    return 0;
}
```

### A.2.3 makefile

```
all: mytime mysleep
.phony: all
mytime: mytime.c
    gcc mytime.c -o mytime
mysleep: mysleep.c
    gcc mysleep.c -o mysleep
```