操作系统课程设计 实验报告

姓名：刘天祺

班级：07121502

学号：1320151097

学院：计算机学院

专业：物联网工程

日期：2018 年 3 月 21 日

实验二　进程控制

一、实验要求

设计并实现类似 Unix 的“time”命令：“mytime”。“mytime”命令通过命令行参数接收要运行的程序 ,创建一个独立的进程来运行该程序，并记录该程序运行的时间。

实现 Windows 版本和 Linux 版本。

二、实验环境

2.1 Linux 环境

操作系统：Ubuntu 14.04.5 LTS 64bit

Shell：zsh 5.0.5 (x86_64-pc-linux-gnu)

编译器：gcc 4.8.5 (Ubuntu 4.8.5-2ubuntu1~14.04.1)

2.2 Windows 环境

操作系统：Windows 10 64bit

Shell：cmder 160710

编译器：gcc 3.4.5 (mingw-vista special r3)

三、实验步骤

3.1 在 Linux 环境实现

- 使用 fork()/execvpe()来创建进程运行程序

```
if ((pid = fork()) == -1){
    perror("fork error!\n");
}
else if (pid == 0){
    if (debug) printf("[+] pid:%5d executing...\n", getpid());
    gettimeofday(&start, NULL);
    buf[0] = ms(start); // start time of exec
    write(fd[1], buf, sizeof(long));
    execvpe(argv[1], argv+1, envp); // parmlist is argv[2:]
}
```

使用 fork()创建子进程后，在子进程中使用 execvpe()运行程序 P。

P 为 mytime 的第一个参数，mytime 的第二个及其后面的参数为 P 的参数，P 的运行环境与 mytime 相同。

- 使用 wait()等待新创建的进程结束

```
    read(fd[0], buf, sizeof(long));
    wait(0);
    gettimeofday(&end, NULL); // time of child proc done
```

- 调用 gettimeofday()来获取时间

gettimeofday 函数定义在头文件<sys/time.h>中，用于获取系统时间。

其用于存储时间信息的结构体 timeval，定义如下：

```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

方便起见，本实验中定义 ms 宏，将 timeval 类型的变量转化为以微秒为单位的时间戳，ms 宏定义如下：

```
#define ms(X) X.tv_sec*1000000+X.tv_usec
```

- 计算程序运行的时间

```
else if (pid == 0){
    if (debug) printf("[+] pid:%5d executing...\n", getpid());
    gettimeofday(&start, NULL);
    buf[0] = ms(start); // start time of exec
    write(fd[1], buf, sizeof(long));
    execvpe(argv[1], argv+1, envp); // parmlist is argv[2:]
}
else{
    if (debug) printf("[+] pid:%5d waiting for child proc...\n", getpid());
    read(fd[0], buf, sizeof(long));
    wait(0);
    gettimeofday(&end, NULL); // time of child proc done
    if (debug) printf("[-] pid:%5d done.\n", getpid());
}
printf("[*] %s cost: %fs.\n", argv[1], (ms(end)-*buf)/1000000.0);
```

由于使用 fork()创建子进程，子进程与父进程的变量不共享，故在程序中使用 pipe()进行进程间的通讯：子进程开始执行时，获取系统时间，即子进程开始执行的时间，随后将其开始时间发送至写管道；父进程通过读管道获取子进程的开始时间，在等待子进程结束执行后，再次获取系统时间，即子程序执行完毕的时间。

二者的时间差即为 mytime 通过命令行参数接收到的要运行的程序的执行时间。

3.2 在 Windows 环境实现

- 使用 CreateProcess()来创建进程

通过指定 CreateProcess 的第二个参数(lpCommandLine)为要运行的程序，即 mytime 的首个参数。实现了创建进程并运行程序。

```
HANDLE hchild = startproc(argv[1]);
```

```
HANDLE startproc(char* cmd) {
        STARTUPINFO si;
        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        PROCESS_INFORMATION pi;
        BOOL bCreateOk = CreateProcess(
                NULL,
                cmd,
                NULL,
                NULL,
                FALSE,
                0,
                NULL,
                NULL,
                &si, //StartupInfo
                &pi  //ProcessInfo
                );
        if (bCreateOk) {
                return pi.hProcess;
        }
        else {
                return INVALID_HANDLE_VALUE;
        }
}
```

- 使用 WaitForSingleObject()在“mytime”命令和新创建的进程之间同步

```
HANDLE hchild = startproc(argv[1]);
WaitForSingleObject(hchild, INFINITE); // wait for child proc done
```

- 调用 GetSystemTime()来获取时间

  该 API 使用 SYSTEMTIME 类型记录时间信息。

  本次实验中使用其结构体中的 wSecond 和 wMilliseconds。

  方便起见，定义 ms 宏，将 SYSTEMTIME 类型变量转换为以毫秒为单位的时间戳，ms 宏定义如下：

```
#define ms(X) X.wSecond*1000+X.wMilliseconds
```

- 计算程序运行的时间

  在进程创建并执行后获取系统时间，如下：

```
HANDLE hchild = startproc(argv[1]);
GetSystemTime(&begin);
```

  在等待进程结束后再次获取系统时间，如下：

```
WaitForSingleObject(hchild, INFINITE); // wait for child proc done
GetSystemTime(&end);
```

  二者的时间差即为 mytime 通过命令行参数接收到的要运行的程序的执行时间。

四、实验结果

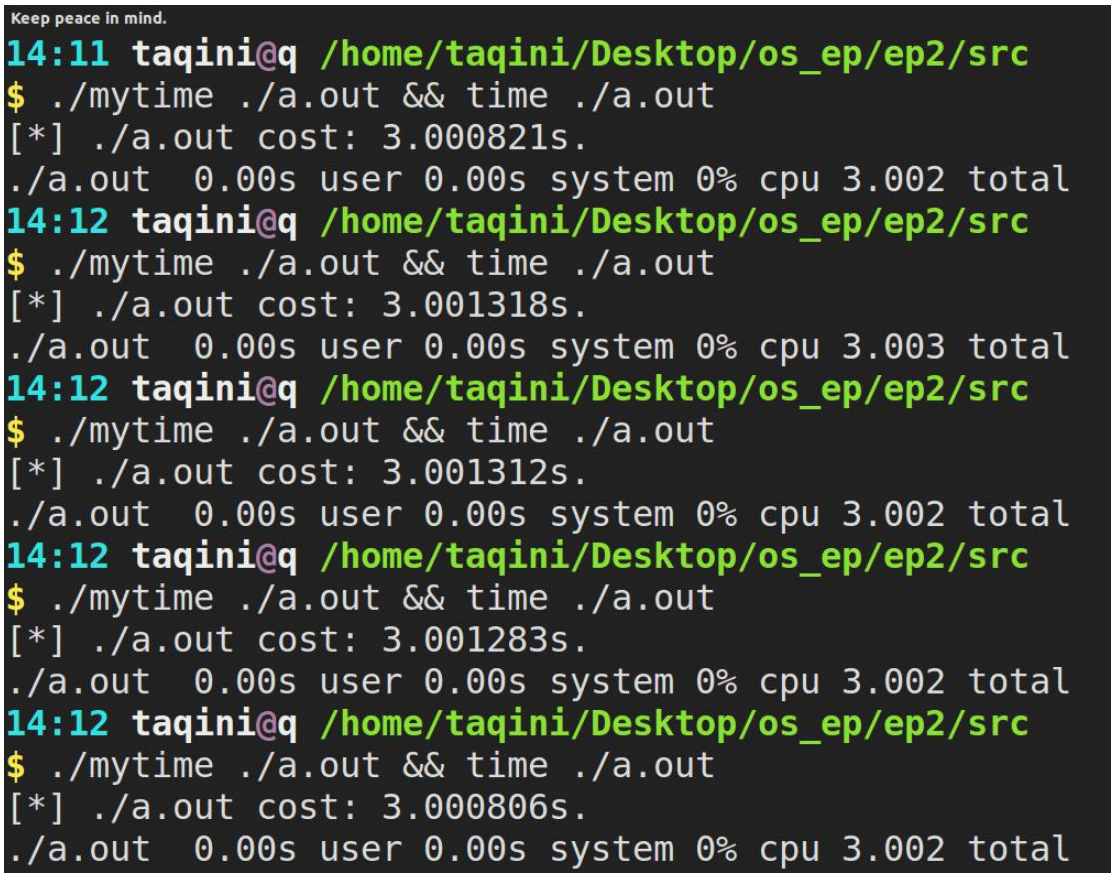
4.1 在 Linux 环境下的实验结果

要运行的测试程序源代码 a.c 如下：

```
#include <stdio.h>
#include <unistd.h>
#define debug 0
int main(){
    sleep(3);
    if (debug) printf("[-] pid:%5d done.\n", getpid());
    return 0;
}
```

使用 gcc 将测试程序源代码 a.c 和 mytime 命令源代码 mytime.c 分别编译为可执行文件 mytime 和 a.out。

指定./a.out 文件作为参数，先后执行 mytime 命令和系统 time 命令，执行 5 次后的结果如下图：



由 mytime 命令统计的 a.out 的执行时间平均为 3.001108 秒；

由 time 命令统计的 a.out 的执行时间平均为 3.0022 秒。

mytime 命令的误差约为 0.04%。

此外，本实验中编写的 mytime 程序实现了与 time 命令类似的功能，可以统计**带参数**的程序执行的时间，如下：

```
16:22 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ time ls -alh
time ls -alh
total 100K
drwxrwxr-x 3 taqini taqini 4.0K Mar 21 16:22 .
drwxrwxr-x 5 taqini taqini 4.0K Mar 21 14:10 ..
-rw-rw-r-- 1 taqini taqini  145 Mar 21 16:21 a.c
-rwxrwxr-x 1 taqini taqini 8.4K Mar 21 16:22 a.out
-rwxrwxr-x 1 taqini taqini  137 Mar 20 17:35 cmp.sh
-rw-rw-r-- 1 taqini taqini  206 Mar 20 17:29 makefile
-rwxrwxr-x 1 taqini taqini 8.9K Mar 21 16:22 mytime
-rw-rw-r-- 1 taqini taqini  903 Mar 21 15:48 mytime.c
drwxrwxr-x 2 taqini taqini 4.0K Mar 21 16:04 windows
ls --color=tty -alh  0.00s user 0.00s system 0% cpu 0.002 total
16:22 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ls -alh
./mytime ls -alh
total 100K
drwxrwxr-x 3 taqini taqini 4.0K Mar 21 16:22 .
drwxrwxr-x 5 taqini taqini 4.0K Mar 21 14:10 ..
-rw-rw-r-- 1 taqini taqini  145 Mar 21 16:21 a.c
-rwxrwxr-x 1 taqini taqini 8.4K Mar 21 16:22 a.out
-rwxrwxr-x 1 taqini taqini  137 Mar 20 17:35 cmp.sh
-rw-rw-r-- 1 taqini taqini  206 Mar 20 17:29 makefile
-rwxrwxr-x 1 taqini taqini 8.9K Mar 21 16:22 mytime
-rw-rw-r-- 1 taqini taqini  903 Mar 21 15:48 mytime.c
drwxrwxr-x 2 taqini taqini 4.0K Mar 21 16:04 windows
[*] ls cost: 0.002533s.
```

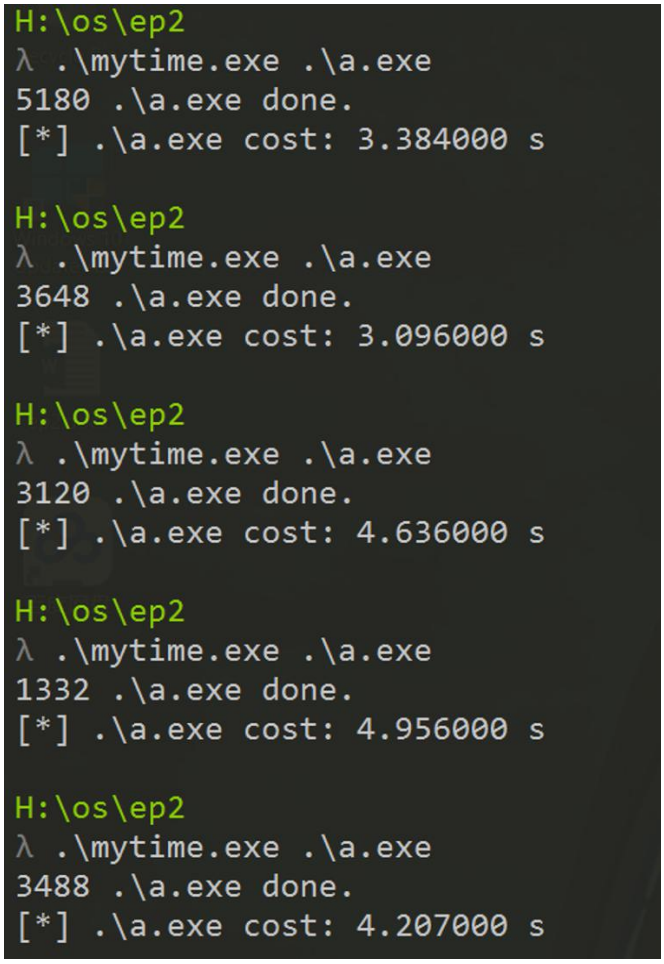
4.2 在 Windows 环境下的实验结果

要运行的测试程序源代码 a.c 如下：

```c
#include <windows.h>
int main(int argc, char **argv){
    Sleep(3000);
    printf("%d %s done.\n",getpid(),argv[0]);
    return 0;
}
```

使用 gcc 将测试程序源代码 a.c 和 mytime 命令源代码 mytime.c 分别编译为可执行文件 mytime.exe 和 a.exe。

指定.\a.exe 文件作为参数，执行 mytime.exe 命令 5 次后的结果如下图：



由 mytime 命令统计的 a.exe 的执行时间平均为 4.0558 秒。

五、实验总结

在 linux 环境下，mytime 命令在统计新运行的程序的时间时，对于同样一个主函数中只有 sleep(3)的程序，如果不使用管道通讯，在主进程开始时记录开始时间，在主进程等待子进程结束后记录结束时间，则他们的差值在 4s 左右，和使用管道通讯得到的差值 3s 相差不少。我认为原因在于子进程和主进程开始的时间不同，fork()调用之后，创建了子进程，此时可能因为操作系统任务调度、内存换页等因素导致主进程并未立即执行下去。因此在调用 execv()之前记录开始时间较为合理，对比于系统的 time 命令，mytime 和 time 统计的时间相近，

因此可以猜测，time 命令的源码中，可能就是通过 pipe()实现的子进程和主进程之间通讯，从而获取的子进程执行时间。

未使用管道的 mytime 执行结果，如下图：

```
22:47 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ./a.out
[*] ./a.out cost: 4.952247s.
22:47 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ./a.out
[*] ./a.out cost: 3.522478s.
22:47 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ./a.out
[*] ./a.out cost: 4.882575s.
22:47 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ./a.out
[*] ./a.out cost: 4.066476s.
22:47 taqini@q /home/taqini/Desktop/os_ep/ep2/src
$ ./mytime ./a.out
[*] ./a.out cost: 3.238853s.
```

相较于 Win32 API，linux 的系统调用更加简练明了，给人一种短小精悍的感觉，Win32 API 虽然功能强大，但是使用起来过于复杂，不光函数名字长，函数的参数也多。API 固然方便开发者使用 ,但是同时也限制住了开发者的思维，如果把编程比作搭建一间房子，linux 的系统调用像是一块砖一片瓦，程序员做的事情是用水泥把他砖头一块一块的垒起来，把瓦片一片一片的铺上去，而 Win32 API 则像是请建筑工队帮忙搭建房间，程序员做的事情则是指挥建筑工队如何搭房子。对于我来说，使用 linux 的系统调用编程更加轻松愉快。