**操作系统课程设计实验报告**

实验名称： 进程控制

姓名/学号： 宋尚儒/1120180717

1. **实验目的**

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

1. **实验内容**

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

在Windows下实现:

使用CreateProcess()来创建进程

使用WaitForSingleObject()在“mytime”命令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()/vfork /exec()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

mytime的用法： $ mytime.exe program1

要求输出程序program1运行的时间。Pragram1可以为自己写的程序，也可以是系统里的应用程序。如Linux: ls vi top等命令 或者Windows：notepad

$ mytime.exe program2 t

t为时间参数，为program2的输入参数，控制program2的运行时间。最后输出program2的运行时间，应和t基本接近。

显示结果： \*\*小时\*\*分\*\*秒\*\*毫秒\*\*微秒

1. **实验环境**

Windows10

VMWare15.5

Ubuntu-20.04

1. **程序设计与实现**

mytime的简要设计思路为

(1) 定义并初始化需要的信息

(2) 创建子进程并传递参数运行程序，主进程开始计时

(3) 主进程等待至子进程结束，结束计时

(4) 输出时间

上述各步骤在不同系统下的实现方式不同，下面来看两个系统下的具体实现

1. 在Windows下实现：
2. 定义系统时间对象，初始化子进程的进程信息和启动信息

SYSTEMTIME start,end;

STARTUPINFO si;

memset(&si,0,sizeof(si));

si.cb=sizeof(si);

PROCESS\_INFORMATION pi;

读入mytime主进程参数，将其保存到字符串Cmdstr中

char Cmdstr[1005]="";

for(int i=1;i<argc;i++)

{

strcat(Cmdstr,argv[i]);

strcat(Cmdstr," ");

}

1. 以传入命令行参数的形式调用CreateProcess函数创建进程，命令行参数即为Cmdstr，如果创建成功，则子进程信息会被保存，主进程调用系统函数GetSystemTime获取当前系统时间start

if(CreateProcess(

    NULL, //不指定模块名称，使用命令行

    Cmdstr, //具体命令行

    NULL, //不继承进程句柄

    NULL, //不继承线程句柄

    FALSE, //不继承当前进程句柄

    0, //无进程创建标志

    NULL, //使用父进程环境

    NULL, //使用父进程目录

    &si, //进程启动信息地址

    &pi ) //进程信息地址

)

GetSystemTime(&start);

1. 主进程调用WaitForSingleObject等待子进程结束，进入阻塞状态。

WaitForSingleObject(pi.hProcess,INFINITE);

子进程结束后，主进程调用系统函数GetSystemTime获取当前系统时间end，并关闭子进程句柄

CloseHandle(pi.hProcess);

GetSystemTime(&end);

1. 计算end和start之间的时间差值，输出具体文本，这里不考虑时间超过24小时的情况

int t[5];

t[0]=end.wHour-start.wHour;

t[1]=end.wMinute-start.wMinute;

t[2]=end.wSecond-start.wSecond;

t[3]=end.wMilliseconds-start.wMilliseconds;

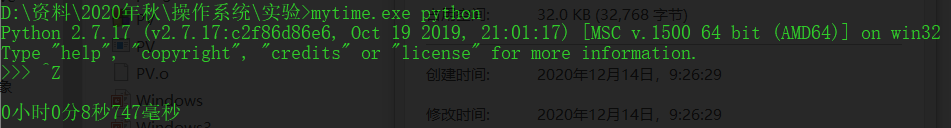
if(t[1]<0) t[0]-=1,t[1]+=60;

if(t[2]<0) t[1]-=1,t[2]+=60;

if(t[3]<0) t[2]-=1,t[3]+=1000;

printf("%d小时%d分%d秒%d毫秒\n",t[0],t[1],t[2],t[3]);

编译生成mytime.exe，在该目录下输入命令行mytime.exe python，系统应用程序调用成功



hello.exe为自己编写的可复制输入并输出的程序,输入命令行mytime.exe hello.exe，调用成功



program2.exe为可根据输入参数睡眠指定时间后结束的程序，输入命令行mytime.exe program2.exe 3，mytime反馈的时间与输入参数接近，调用成功

![C:\Users\94824\AppData\Roaming\Tencent\Users\948249890\TIM\WinTemp\RichOle\TB7](~(L)GU9XQ7NAYY)ZWU.png](data:image/png;base64,)

1. 在Linux下实现：
2. 由于linux系统只有在运行环境变量目录下的可执行文件才可以输入文件名无需’./’就运行，为了完美实现实验要求，首先将mytime文件路径加入临时环境变量，这样就可以“./mytime program”的形式直接调用mytime目录下的可执行文件

char path[105],temp[105];

strcpy(path,getenv("PATH"));

getcwd(temp,sizeof(temp));

strcat(path,":");

strcat(path,temp);

setenv("PATH",path,1);

定义系统时间对象

struct timeval start;

struct timeval end;

1. 创建子进程，以pid值区分父子进程

int pid=fork();

对于子进程，使用execvp函数调用执行文件，execvp第一个参数为调用的文件名称，第二个参数为传递给该文件的参数指针数组，因为要从第主函数二个参数开始传，所以使用argv+1

execvp(argv[1],argv+1);

对于父进程，将当前系统时间保存在start中

gettimeofday(&start,NULL);

1. 主进程调用wait函数等待子进程结束，进入阻塞状态。

wait(NULL);

子进程结束后，主进程结束等待，记录当前系统时间保存在end中

gettimeofday(&end,NULL);

1. 计算end和start之间的时间差值，输出具体文本，这里不考虑时间超过24小时的情况

   int sec=end.tv\_sec-start.tv\_sec;

    int usec=end.tv\_usec-start.tv\_usec;

    if(usec<0)

    {

     sec-=1;

usec+=1000000;

    }

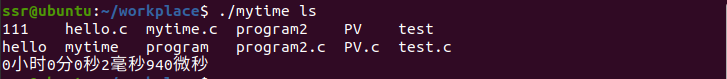
int h=sec/3600; sec=sec%3600;

int m=sec/60; sec=sec%60;

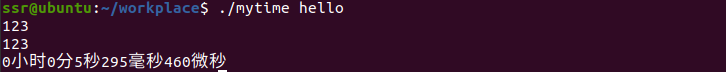
int mm=usec/1000; usec=usec%1000;

printf("%d小时%d分%d秒%d毫秒%d微秒\n",h,m,sec,mm,usec);

使用gcc编译生成mytime，在该目录下输入命令行./mytime ls，系统应用程序调用成功



hello为自己编写的可复制输入并输出的程序,输入命令行./mytime hello，调用成功



program2为可根据输入参数睡眠指定时间后结束的程序，输入命令行./mytime program2 3，mytime反馈的时间与输入参数接近，调用成功

C:\Users\94824\AppData\Roaming\Tencent\Users\948249890\TIM\WinTemp\RichOle\W`%BDDO4Y6HI)}{M[{C$VG2.png

1. **实验收获与体会**

这次实验的思路比较简单，主要精力用于学习API，学习各种函数和具体用法花了很长时间，比如学习CreateProcess函数的参数用法、exec函数族在用法上的差异等，并且由于Linux自身安全性的设置，还学习了用c修改环境变量的方法，最终在两个系统上都实现了要求。