



# 東北大學

## 生物医学工程专业 实验报告

实验课程：\_\_\_\_ 操作系统 \_\_\_\_\_

班级：\_\_\_\_ 1502 \_\_\_\_\_ 姓名：\_\_\_\_ 尚麟静 \_\_\_\_\_

学号：\_\_\_\_ 20155467 \_\_\_\_\_ 同组人：\_\_\_\_\_

指导教师：\_\_\_\_ 张轶 \_\_\_\_\_

实验成绩（教师签字）： \_\_\_\_\_

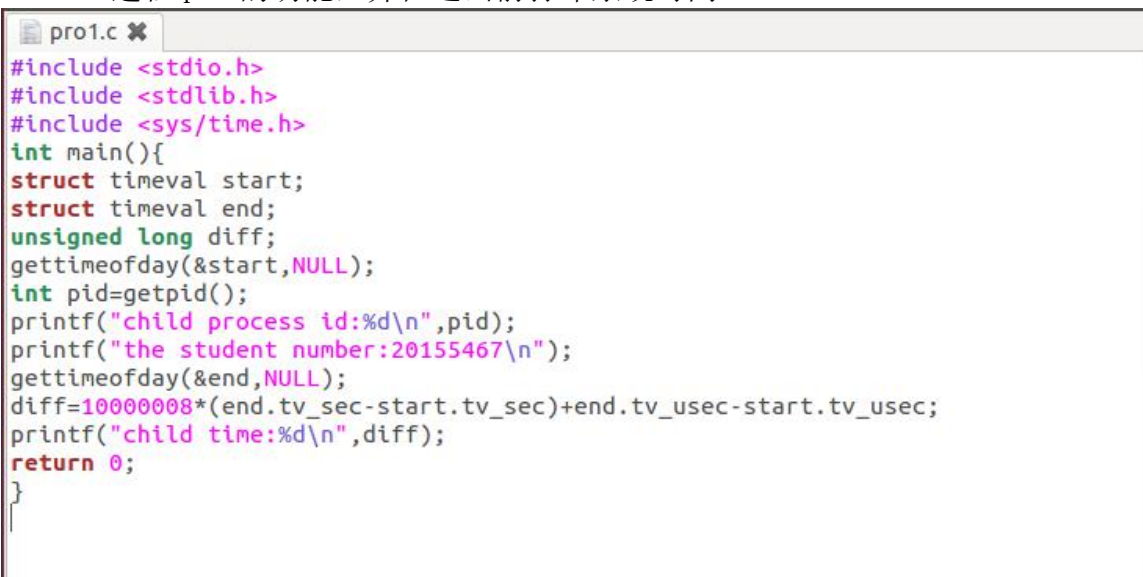
## 实验一 进程的运行

### 一 实验目的：

掌握进程的创建方法；掌握程序执行时间的计量方法（例子 gettimeofday）。通过 Fork 及 Exec 等标准系统函数接口实现进程的创建功能；理解父子进程间的执行关系；通过能够对程序执行时间进行精确计时的函数的使用，掌握程序执行时间的计量方法。

### 二 实验内容：

1. 使用 fork 跟 exec 系统调用编写父进程创建子进程的程序。其中，父进程打印自身 pid，创建子进程，等待子进程结束并打印当前系统时间；子进程完成打印自己学号，当前进程 pid 的功能，并在退出前打印系统时间。



```
pro1.c ✕
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
int main(){
    struct timeval start;
    struct timeval end;
    unsigned long diff;
    gettimeofday(&start,NULL);
    int pid=getpid();
    printf("child process id:%d\n",pid);
    printf("the student number:20155467\n");
    gettimeofday(&end,NULL);
    diff=1000000*(end.tv_sec-start.tv_sec)+end.tv_usec-start.tv_usec;
    printf("child time:%d\n",diff);
    return 0;
}
```

## 实验二 文件系统的使用

### 一 实验目的：

实习文件系统的库函数调用及低级文件函数调用，加深对文件系统构成关系的理解。掌握文件系统上的基本操作。

### 二 实验内容：

2. 分别使用高级文件操作（fopen 类），低级文件操作（open 类）执行以下过程：
  - A) 打开文件；
  - B) 写入自己学号，姓名，当前系统时间；
  - C) 从该文件中读出 B 步骤所写入内容，并将其输出到屏幕；
  - D) 打印当前系统时间到屏幕；
  - E) 统计文件操作所需消耗的时间。
  - F) 程序退出。
3. 对上述实验现象进行解释。

## 高级文件操作 fopen

```
fopen.c ✕
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/time.h>
#include<fcntl.h>
int main(){
FILE* fi;
fi=fopen("slj1.txt","w");
float time_use=0;
struct timeval start;
struct timeval end;
gettimeofday(&start,NULL);
char s[50],buffer[80];
sprintf(s,"20155467\nShanglinjing\nstart.tv_usec:%lld\n",1ll*start.tv_usec);

    fprintf(fi,"%s",s);
    fclose(fi);

    fi=fopen("slj1.txt","r");
    if(fi==NULL){
        printf("open file slj1.txt failed!\n");
        exit(1);
    }
```

C ▾ Tab Width: 8 ▾ Ln 29, Col 34 INS

```
fopen.c ✕
char s[50],buffer[80];
sprintf(s,"20155467\nShanglinjing\nstart.tv_usec:%lld\n",1ll*start.tv_usec);

    fprintf(fi,"%s",s);
    fclose(fi);

    fi=fopen("slj1.txt","r");
    if(fi==NULL){
        printf("open file slj1.txt failed!\n");
        exit(1);
    }
    else{
        printf("open file slj1.txt succeed!\n");
    }

    while(fgets(buffer,80,fi)!=NULL)
        printf("%s", buffer);

    gettimeofday(&end,NULL);
    printf("end.tv_usec:%lld\n",1ll*end.tv_usec);

    time_use=(end.tv_sec-start.tv_sec)*1000000+(end.tv_usec-start.tv_usec);
    printf("time_use is %.10f\n",time_use);

    return 0;
```

Trash

C ▾ Tab Width: 8 ▾ Ln 29, Col 34 INS

```
a@ubuntu:~$ ./fopen
open file slj1.txt succeed!
20155467
Shanglinjing
start.tv_usec:812912
end.tv_usec:813446
time_use is 534.000000000000
a@ubuntu:~$
```

低级文件操作 open

```
open.c ✕
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
int main()
{
    int fd, size;

    float time_use=0;
    struct timeval start;
    struct timeval end;

    gettimeofday(&start, NULL);

    char s[50], buffer[80];
    sprintf(s, "20155467\nShanglinjing\nstart.tv_usec:%lld\n", 1ll*start.tv_usec);

    fd = open("slj1.txt", O_WRONLY|O_CREAT);
    write(fd, s, sizeof(s));
    close(fd);

    fd = open("slj1.txt", O_RDONLY);
    size = read(fd, buffer, sizeof(buffer));
```

```

struct timeval start;
struct timeval end;

gettimeofday(&start,NULL);

char s[50], buffer[80];
sprintf(s,"20155467\nShanglinjing\nstart.tv_usec:%lld\n",1ll*start.tv_usec);

fd = open("slj1.txt", O_WRONLY|O_CREAT);
write(fd, s, sizeof(s));
close(fd);

fd = open("slj1.txt", O_RDONLY);
size = read(fd, buffer, sizeof(buffer));
close(fd);

printf("%s", buffer);
gettimeofday(&end,NULL);
printf("end.tv_usec:%lld\n",1ll*end.tv_usec);

time_use=(end.tv_sec-start.tv_sec)*1000000+(end.tv_usec-start.tv_usec);
printf("time_use is %.10f\n",time_use);

return 0;
}

```

```

a@ubuntu:~$ gcc open.c -o open
a@ubuntu:~$ ./open
20155467
Shanglinjing
start.tv_usec:894992
end.tv_usec:895553
time_use is 561.0000000000
a@ubuntu:~$

```

上述实验分别用 open 和 fopen 函数来进行文件操作，首先打开文件，写入当前系统时间和学号并再次进行读取以及打印系统时间，很明显，通过上述实验结果的对比，fopen 操作的时间比 open 要更快速。

使用 fopen 函数，由于在用户态下就有了缓冲，因此进行文件读写操作的时候就减少了用户态和内核态的切换，而使用 open 函数，在文件读写时则每次都需要进行内核态和用户态的切换。所以顺序访问文件，fopen 系列的函数要比直接调用 open 系列的函数快；如果随机访问文件则相反。

## 实验三 线程基本操作

### 一 实验目的：

掌握线程的创建方法。掌握直接执行系统调用的方法。理解进程及线程间的逻辑联系。

### 二 实验内容：

4. 在同一进程中创建两个线程，在每个线程中打印进程 pid，线程 tid，线程 pid。



5. 结合实验结果解释进程 pid, 线程 tid, 线程 pid。

```
process.c
#include<stdio.h>
#include<pthread.h>
#include<sys/types.h>
#include<sys/syscall.h>
void thread_1(void){
    printf("the first thread,the tid=%lu,pid=%d\n",pthread_self,syscall
(SYS_gettid));
    printf("the first thread,getpid()=%d\n",getpid());
}
void thread_2(void){
    printf("the second thread,the tid=%lu,pid=%d\n",pthread_self,syscall
(SYS_gettid));
    printf("the second thread,getpid()=%d\n",getpid());
}
int main(void){
    pthread_t id_1,id_2;
    int i,ret;
    ret=pthread_create(&id_1,NULL,(void *) thread_1,NULL);
    if(ret!=0){
        printf("Create pthread error!\n");
        return -1;
    }
    ret=pthread_create(&id_2,NULL,(void *) thread_2,NULL);
    if(ret!=0){
        printf("Create pthread error!\n");
        return -1;
    }
    pthread_join(id_1,NULL);
    pthread_join(id_2,NULL);
    return 0;
}

C Tab Width: 8 Ln 24, Col 19 INS
a@ubuntu:~$ gcc process.c -o process -pthread
process.c: In function 'thread_1':
process.c:6:3: warning: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type '__attribute__((const)) pthread_t (*)(void)' [-Wformat]
process.c: In function 'thread_2':
process.c:10:3: warning: format '%lu' expects argument of type 'long unsigned int', but argument 2 has type '__attribute__((const)) pthread_t (*)(void)' [-Wformat]
a@ubuntu:~$ ./process
the second thread,the tid=134513744,pid=3134
the second thread,getpid()=3132
the first thread,the tid=134513744,pid=3133
the first thread,getpid()=3132
a@ubuntu:~$
```

6. 答:同一个进程中的不同线程 pid 不同,但是进程 pid 相同,线程 tid 相同。tid 在某进程中是唯一的,在不同的进程中创建的线程可能出现 ID 值相同的情况。

在进程 P1 要向另外一个进程 P2 中的某个线程发送信号时,既不能使用 P2 的 pid,更不能使用线程的 pthread id,而只能使用该线程的 tid。

## 实验四 多线程同步

一 实验目的:

熟悉多线程间的同步机制；能够使用标准的线程间同步机制实现线程同步功能。

## 二 实验内容：

7. 创建两个线程按数字顺序打印 10 以下自然数，其中一个线程打印 1-3 及 8-10；另一个线程打印 4-6。要求使用线程同步机制实现上述打印顺序。

```
multithread.c ✕
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t lock;
pthread_cond_t cond;

int i = 1;

void* salewinds1(void* args)
{
    while(1)
    {
        pthread_mutex_lock(&lock);
        if(i>10 )
        {
            pthread_mutex_unlock(&lock);
            pthread_exit(0);
        }
        while( ((i>3)&&(i<8)))
        {
            pthread_cond_wait(&cond, &lock);
        }

        if(i > 10)
        {
            pthread_mutex_unlock(&lock);
            pthread_exit(0);
        }

        printf("F1: %d\n", i);
        i += 1;

        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

void* salewinds2(void* args)
{
    while(1)
    {
        pthread_mutex_lock(&lock);
```

```

        pthread_mutex_lock(&lock);
        if(i >10)
        {
            pthread_mutex_unlock(&lock);
            pthread_exit(0);
        }
        while(((i>0)&&(i<4))||((i>6)&&(i<11)) )
        {
            pthread_cond_wait(&cond, &lock);
        }
if(i >10)
    {
        pthread_mutex_unlock(&lock);
        pthread_exit(0);
    }

    printf("F2: %d\n", i);
    i += 1;
if(i==7){
i += 1;}

        pthread_cond_broadcast(&cond);

        pthread_mutex_unlock(&lock);

    }
    return NULL;
}

int main(int argc,const char* argv[])
{
    pthread_t thd1;
    pthread_t thd2;
    int val1=1;
    int val2=2;
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&cond, NULL);
    pthread_create(&thd1, NULL, salewinds1, NULL);
    pthread_create(&thd2, NULL, salewinds2, NULL);

    pthread_join(thd1, (void**)&val1);

    pthread_join(thd2, (void**)&val2);

    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&cond);
    return 0;
}

```

C ▾ Tab Width: 8 ▾ Ln 108, Col 1 INS

试验结果



```
a@ubuntu: ~  
a@ubuntu:~$ gcc multithread.c -lpthread  
a@ubuntu:~$ ./a.out  
F1: 1  
F1: 2  
F1: 3  
F2: 4  
F2: 5  
F2: 6  
F1: 8  
F1: 9  
F1: 10  
a@ubuntu:~$
```

```
pro2.c ✕  
#include<sys/time.h>  
int main(){  
    struct timeval start;  
    struct timeval end;  
    unsigned long diff;  
    gettimeofday(&start,NULL);  
    int pid,mypid,cpid;  
    pid=getpid();  
    printf("parent id:%d\n",pid);  
    cpid=fork();  
    if(cpid>0){  
        mypid=getpid();  
        printf("%d parent of %d",mypid,cpid);  
        wait();  
        gettimeofday(&end,NULL);  
        diff=1000000*(end.tv_sec-start.tv_sec)+end.tv_usec-start.tv_usec;  
        printf("total time %ld\n",diff);  
    }  
    else if(cpid==0){  
        execl("./pro1",NULL);  
    }  
    else{  
        perror("fork failed");  
        exit(1);  
    }  
    exit(0);  
}
```

```

a@ubuntu:~$ gcc pro1.c -o pro1
pro1.c: In function 'main':
pro1.c:14:1: warning: format '%d' expects argument of type 'int', but argument
has type 'long unsigned int' [-Wformat]
a@ubuntu:~$ gcc pro2.c -o pro2
pro2.c: In function 'main':
pro2.c:10:2: warning: incompatible implicit declaration of built-in function
'printf' [enabled by default]
pro2.c:21:3: warning: incompatible implicit declaration of built-in function
'exit' [enabled by default]
pro2.c:21:3: warning: not enough variable arguments to fit a sentinel [-Wformat]
a@ubuntu:~$ ./pro2
parent id:3369
child process id:3370
the student number:20155467
child time:593
3369 parent of 3370total time 1938
a@ubuntu:~$

```

#### 8. 分别解释 fork 跟 exec 的作用。

使用 fork 方式运行 script 时, 就是让 shell(parent process) 产生一个 child process 去执行该 script, 当 child process 结束后, 会返回 parent process, 但 parent process 的环境是不会因 child process 的改变而改变的。

使用 exec 方式运行 script 时, 它和 source 一样, 也是让 script 在当前 process 内执行, 但是 process 内的原代码剩下部分将被终止。同样, process 内的环境随 script 改变而改变。

## 实验五 多线程同步 2

### 一 实验目的:

进一步练习线程间的同步机制, 熟悉针对复杂情况下的线程间的同步方法。

### 二 实验内容:

9. 使用两个线程打印 100 以内的质数及非质数, 其中一个线程只打印质数, 另一个线程只打印非质数, 要求所有数被按照顺序输出。
10. 解释程序运行过程。

```

#include <stdio.h>
#include <pthread.h>
#include "stdlib.h"
#include "unistd.h"

pthread_mutex_t mutex;
pthread_cond_t signal1;
pthread_cond_t signal2;

int i;
int no=1;

void hander(void *arg)
{
    free(arg);
    (void)pthread_mutex_unlock(&mutex);
}

int solveno(int x)
{
    int k=0;
    int j=0;
    for(j=2; j<x; j++){
        if(x%j==0)
            k++;
    }
    if (x==1)
        return 1;
    else
        return k;
}

void *thread1(void *arg)
{
    while(no<=100)
    {
        pthread_mutex_lock(&mutex);
        if(solveno(no)!=0)
            printf("非质数:%d\n",no);
        else
        {
            pthread_cond_signal(&signal2);
            pthread_cond_wait(&signal1, &mutex);
        }
        pthread_mutex_unlock(&mutex);
        no++;
    }
    pthread_cond_signal(&signal2);
}

void *thread2(void *arg)
{

```

```

        while(1)
        {
            pthread_mutex_lock(&mutex);
            pthread_cond_wait(&signal2, &mutex);
            if(no>100)
                exit(0);
            printf("质数:%d\n",no);
            pthread_cond_signal(&signal1);
            pthread_mutex_unlock(&mutex);
        }
    }

int main(void)
{
    pthread_t tid1,tid2;
    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&signal1,NULL);
    pthread_cond_init(&signal2,NULL);
    pthread_create(&tid1,NULL,thread1,NULL);
    pthread_create(&tid2,NULL,thread2,NULL);
    sleep(5);

    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);

    pthread_exit(0);
    return 0;
}

```

C ▾ Tab Width: 8 ▾

Ln 26, Col 2

INS

```

a@ubuntu: ~
a@ubuntu:~$ gcc -g hello.c -o hello -pthread
a@ubuntu:~$ ./hello
非质数:1
质数:2
质数:3
非质数:4
质数:5
非质数:6
质数:7
非质数:8
非质数:9
非质数:10
质数:11
非质数:12
质数:13
非质数:14
非质数:15
非质数:16
质数:17
非质数:18
质数:19
非质数:20
非质数:21
非质数:22

```

```
a@ubuntu: ~  
质数:23  
非质数:24  
非质数:25  
非质数:26  
非质数:27  
非质数:28  
质数:29  
非质数:30  
质数:31  
非质数:32  
非质数:33  
非质数:34  
非质数:35  
非质数:36  
质数:37  
非质数:38  
非质数:39  
非质数:40  
质数:41  
非质数:42  
质数:43  
非质数:44  
非质数:45  
非质数:46  
  
a@ubuntu: ~  
质数:47  
非质数:48  
非质数:49  
非质数:50  
非质数:51  
非质数:52  
质数:53  
非质数:54  
非质数:55  
非质数:56  
非质数:57  
非质数:58  
质数:59  
非质数:60  
质数:61  
非质数:62  
非质数:63  
非质数:64  
非质数:65  
非质数:66  
质数:67  
非质数:68  
非质数:69  
非质数:70
```



```
a@ubuntu: ~
质数:71
非质数:72
质数:73
非质数:74
非质数:75
非质数:76
非质数:77
非质数:78
质数:79
非质数:80
非质数:81
非质数:82
质数:83
非质数:84
非质数:85
非质数:86
非质数:87
非质数:88
质数:89
非质数:90
非质数:91
非质数:92
非质数:93
非质数:94
非质数:95
非质数:96
质数:97
非质数:98
非质数:99
非质数:100
a@ubuntu:~$
```

## 2. 解释程序运行过程:

no 是 1 到 100 的自然数, solveno 用来判断自然数能被 1- (no-1) 整除的个数。  
创建的两个线程分别是 tid1 和 tid2, thread1 和 thread2 是 tid1 和 tid2 的函数。  
thread1 调用 solveno 函数判断 no 是否是质数。  
如果是非质数由 thread1 输出结果。(此时 thread2 处于阻塞状态);  
如果是质数, thread1 发送信号唤醒 thread2, thread1 阻塞, 由 thread2 输出结果。  
Thread2 打印完结果之后再唤醒 thread1。no 逐次递增, 大于 100, 结束程序。

# 实验六 多线程调度

## 一 实验目的:

掌握 Linux 中线程调度策略的配置方法; 理解不同调度策略对执行过程产生的影响。

## 二 实验内容:

11. 通过系统调用接口设置线程所使用的调度策略。以多次打印线程 ID 为示例, 通过配置 FIFO 与 RR 调度策略来示例调度策略对程序执行过程的影响。
12. 解释程序运行过程。

## 1. 实验代码:

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

```

void *thread1(void *arg) {
    int i=0;
    int j;
    while(i<10) {
        printf("thread1 pid:%d, time:%d\n", getpid(), i);
        for(j=0; j<1000000; j++);
        i++;
    }
}

void *thread2(void *arg) {
    int i=0;
    int j;
    while(i<10) {
        printf("thread2 pid:%d, time=%d\n", getpid(), i);
        for(j=0; j<1000000; j++);
        i++;
    }
}

int main() {
    pthread_t pth1, pth2;
    pthread_attr_t attr1, attr2;
    pthread_attr_init(&attr1);
    pthread_attr_init(&attr2);
    pthread_attr_setschedpolicy(&attr1, SCHED_FIFO);
    pthread_attr_setschedpolicy(&attr2, SCHED_RR);

    pthread_create(&pth1, &attr1, thread1, NULL);
    pthread_create(&pth2, &attr1, thread2, NULL);

    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    printf(".....\n");

    pthread_create(&pth1, &attr2, thread1, NULL);
    pthread_create(&pth2, &attr2, thread2, NULL);

    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    pthread_attr_destroy(&attr1);
    pthread_attr_destroy(&attr2);
    return 0;
}

```

```
a@ubuntu: ~  
a@ubuntu:~$ gcc hello.c -o hello -pthread  
a@ubuntu:~$ ./hello  
thread2 pid:4089,time=0  
thread2 pid:4089,time=1  
thread1 pid:4089,time=0  
thread1 pid:4089,time=1  
thread1 pid:4089,time=2  
thread1 pid:4089,time=3  
thread2 pid:4089,time=2  
thread2 pid:4089,time=3  
thread2 pid:4089,time=4  
thread2 pid:4089,time=5  
thread2 pid:4089,time=6  
thread1 pid:4089,time=4  
thread1 pid:4089,time=5  
thread1 pid:4089,time=6  
thread2 pid:4089,time=7  
thread2 pid:4089,time=8  
thread2 pid:4089,time=9  
thread1 pid:4089,time=7  
thread1 pid:4089,time=8  
thread1 pid:4089,time=9  
.....  
thread2 pid:4089,time=0
```

2.

创建的两个线程分别是 pth1 和 pth2，thread1 和 thread2 是 pth1 和 pth2 的函数。

Thread1, thread2 分别实现多次打印线程 ID。

Attr1 和 attr2 分别用 FIFO 和 RR 调度策略。先对两个线程实施 FIFO 调度策略，再对两个线程实施 RR 调度策略，最后销毁线程，结束程序。