

操作系统进程管理

上机实验1~4



操作系统一实验1

实验1 掌握Linux基本命令 和开发环境

- 1. 掌握常用的Linux shell命令；
- 2. 掌握编辑环境VIM；
- 3. 掌握编译环境gcc及跟踪调试工具gdb
- 4. 通过man fork()了解fork系统调用，并调试一个fork()的面试题

操作系统一实验1

面试题：请问下面的程序一共输出多少个“-”？

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    int i;
    for(i=0; i<2; i++){
        fork();
        printf("-");
    }
    wait(NULL);
    wait(NULL);
    return 0;
}
```

操作系统一实验2

实验2 进程

■ 目的

通过观察、分析实验现象，深入理解进程及进程在调度执行和内存空间等方面的特点，掌握在POSIX 规范中fork和kill系统调用的功能和使用。

■ 实验前准备

学习man 命令的用法，通过它查看fork 和kill 系统调用的在线帮助，并阅读参考资料，学会fork 与kill 的用法。复习C 语言的相关内容。

操作系统一实验2

```
/* POSIX 下进程控制的实验程序残缺版 */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <ctype.h>
/* 允许建立的子进程个数最大值 */
#define MAX_CHILD_NUMBER 10
/* 子进程睡眠时间 */
#define SLEEP_INTERVAL 2
int proc_number=0; /* 子进程的自编号，从0开始 */
void do_something();
```

操作系统一实验2

```
main(int argc, char* argv[])
{
    /* 子进程个数 */
    int child_proc_number = MAX_CHILD_NUMBER;
    int i, ch;
    pid_t child_pid;
    pid_t pid[10]={0}; /* 存放每个子进程的id */
    if (argc > 1) /* 命令行参数第一个参数表示子进程个数*/
    {
        child_proc_number = atoi(argv[1]);
        child_proc_number= (child_proc_number > 10) ? 10 :
            child_proc_number;
    }
    .....
```

操作系统一实验2

```
for (i=0; i<child_proc_number; i++) {  
    /* 填写代码，建立child_proc_number个子进程要执行  
     * proc_number = i;  
     * do_something();  
     * 父进程把子进程的id保存到pid[i] */  
}  
/* 让用户选择杀死进程，数字表示杀死该进程，q退出 */  
while ((ch = getchar()) != 'q') {  
    if (isdigit(ch)) {  
        /* 填写代码，向pid[ch-'0']发信号SIGTERM,  
         * 杀死该子进程 */  
    }  
}  
/* 在这里填写代码，杀死本组的所有进程 */  
return;  
}
```

操作系统一实验2

```
void do_something() {  
    for(;;) {  
        printf("This is process No.%d and its pid is %d,  
               proc_number, getpid());  
        sleep(SLEEP_INTERVAL); /* 主动阻塞两秒钟 */  
    }  
}
```

kill()函数用于删除执行中的程序或者任务。调用格式为：
：

```
kill(int PID, int IID);
```

**其中：PID是要被杀死的进程号，IID为向将被杀死的
进程发送的中断号。**

操作系统一实验2

■实验过程

先猜想一下这个程序的运行结果。假如运行“`./process 20`”，输出会是什么样？然后按照注释里的要求把代码补充完整，运行程序。可以多运行一会儿，并在此期间启动、关闭一些其它进程，看`process`的输出结果有什么特点，记录下这个结果。开另一个终端窗口，运行“`ps aux|grep process`”命令，看看`process`究竟启动了多少个进程。回到程序执行窗口，按“数字键+回车”尝试杀掉一两个进程，再到另一个窗口看进程状况。按`q`退出程序再看进程情况。

操作系统一实验2

实验2 进程

■实验报告

回答下列问题，写入实验报告。

1. 你最初认为运行结果会怎么样？
2. 实际的结果什么样？有什么特点？试对产生该现象的原因进行分析。
3. proc_number 这个全局变量在各个子进程里的值相同吗？为什么？
4. kill 命令在程序中使用了几次？每次的作用是什么？执行后的现象是什么？
5. 使用kill 命令可以在进程的外部杀死进程。进程怎样能主动退出？这两种退出方式哪种更好一些？
6. 把你的程序源代码附到实验报告后。

操作系统一实验3

实验3 线程

■ 目的

通过观察、分析实验现象，深入理解线程及线程在调度执行和内存空间等方面的特点，并掌握线程与进程的区别。掌握POSIX 规范中pthread_create() 函数的功能和使用方法。

■ 实验前准备

阅读参考资料，了解线程的创建等相关系统调用

◦

操作系统一实验3

```
/* POSIX 下线程控制的实验程序残缺版 */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <ctype.h>
#include <pthread.h>

#define MAX_THREAD 3 /* 线程的个数 */
unsigned long long main_counter, counter[MAX_THREAD];
/* unsigned long long 是比 long 还长的整数 */
void* thread_worker(void*);
```

操作系统一实验3

```
int main(int argc, char* argv[])
{
    int i, rtn, ch;
    pthread_t pthread_id[MAX_THREAD] = {0}; /* 存放线程id*/
    for (i=0; i<MAX_THREAD; i++)  {
        /* 在这里填写代码，用pthread_create建一个普通的线程，
        线程id存入pthread_id[i]， 线程执行函数是thread_worker
        并i作为参数传递给线程 */
    }
    do {/* 用户按一次回车执行下面的循环体一次。按q退出 */
        unsigned long long sum = 0;
        ..... /* 求所有线程的counter的和 */
    }
}
```

操作系统一实验3

```
for (i=0; i<MAX_THREAD; i++) /* 求所有counter的和 */
    sum += counter[i];
    printf("%llu ", counter[i]);
}
printf("%llu/%llu", main_counter, sum);
} while ((ch = getchar()) != 'q');
return 0;
}

void* thread_worker(void* p) {
    int thread_num;
    /* 在这里填写代码，把main中的i的值传递给thread_num */
    for(;;) /* 无限循环 */
        counter[thread_num]++;
        /* 本线程的counter加一 */
        main_counter++;
        /* 主counter 加一 */
    }
}
```

操作系统一实验3

实验3 线程

■实验过程

按照注释里的要求把代码补充完整，正确编译程序后，先预计一下这个程序的运行结果。具体的结果会是什么样？运行程序。开另一个终端窗口，运行“ps aux”命令，看看thread的运行情况，注意查看thread的CPU占用率，并记录下这个结果。

```
extern int pthread_create ((pthread_t * __thread,
    __const pthread_attr_t * __attr,
    void *(* __start_routine) (void *),
    void * __arg));
```

第一个参数为指向线程标识符的指针，第二个参数用来设置线程属性，第三个参数是线程运行函数的起始地址，最后一个参数是运行函数的参数。

操作系统一实验3

实验3 线程

■实验报告

回答下列问题，写入实验报告。

1. 你最初认为前三列数会相等吗？最后一列斜杠两边的数字是相等，还是大于或者小于关系？
2. 最后的结果如你所料吗？有什么特点？对原因进行分析。
3. thread 的CPU 占用率是多少？为什么会这样？
4. thread_worker() 内是死循环，它是怎么退出的？你认为这样退出好吗？
5. 把你的程序源代码附到实验报告后。并请保留源代码，下次实验需要使用。

操作系统一实验4

实验4 互斥

■ 目的

通过观察、分析实验现象，深入理解理解互斥锁的原理及特点掌握在POSIX 规范中的互斥函数的功能及使用方法。

■ 实验前准备

准备好上节实验完成的程序thread.c。

阅读参考资料，了解互斥锁的加解锁机制及相关的系统调用。

■ 实验内容

找到thread.c 的代码临界区，用临界区解决 main_counter 与sum 不同步的问题。

互斥锁的类型

PTHREAD_MUTEX_NORMAL:不检测死锁，如果等待一个已经锁定的互斥量将会一直等待，即使是同一个线程锁定互斥量两次也会造成死锁，解除有其他线程锁定的互斥量将会引起不确定行为

PTHREAD_MUTEX_ERRORCHECK:检测错误，一个线程重新锁定同一个锁会返回EDEADLK，如果解锁由其它线程锁定的互斥量或者没有锁定的互斥量就会返回错误

PTHREAD_MUTEX_RECURSIVE:线程可以多次锁定同一个互斥锁，并且需要解锁和锁定次数对应，尝试解除没有锁定的互斥锁和解除由其他线程锁定的互斥锁将会引起错误。

PTHREAD_MUTEX_DEFAULT:重复锁定一个锁会导致不确定行为，其他和NORMAL相同。(一般来说它会映射到PTHREAD_MUTEX_NORMAL)

操作系统一实验4

```
/* POSIX 下线程死锁的演示程序 */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <ctype.h>
#include <pthread.h>

#define LOOP_TIMES 10000
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
/*用宏PTHREAD_MUTEX_INITIALIZER来初始化*/
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
void* thread_worker(void*);  
void critical_section(int thread_num, int i);
```

操作系统一实验4

```
int main(void) {  
    int rtn, i;  
    pthread_t pthread_id = 0; /* 存放子线程的id */  
    rtn = pthread_create(&pthread_id, NULL,  
                        thread_worker, NULL );  
    if(rtn != 0) {  
        printf("pthread_create ERROR!\n");  
        return -1;  
    }  
    for (i=0; i<LOOP_TIMES; i++) {  
        pthread_mutex_lock(&mutex1);  
        pthread_mutex_lock(&mutex2);  
        critical_section(1, i);  
        pthread_mutex_unlock(&mutex2);  
        pthread_mutex_unlock(&mutex1);  
    }  
}
```

操作系统一实验4

```
pthread_mutex_destroy(&mutex1);
pthread_mutex_destroy(&mutex2);
return 0;
}
void* thread_worker(void* p) {
    int i;
    for (i=0; i<LOOP_TIMES; i++) {
        pthread_mutex_lock(&mutex2);
        pthread_mutex_lock(&mutex1);
        critical_section(2, i);
        pthread_mutex_unlock(&mutex2);
        pthread_mutex_unlock(&mutex1);
    }
}
void critical_section(int thread_num, int i) {
    printf("Thread%d: %d\n", thread_num, i);
}
```

操作系统一实验4

实验4 互斥

■实验过程

仔细阅读程序，编译程序后，先预计一下这个程序的运行结果。运行程序。若程序没有响应，按ctrl+c 中断程序运行，然后再重新运行，如此反复若干次，记录下每次的运行结果。若产生了死锁，请修改程序，使其不会死锁。

■实验报告

回答下列问题，写入实验报告。

1. 你预想deadlock.c 的运行结果会如何？
2. deadlock.c 的实际运行结果如何？多次运行每次的现象都一样吗？为什么会这样？
3. 把修改后的两个程序的源代码附在实验报告后。