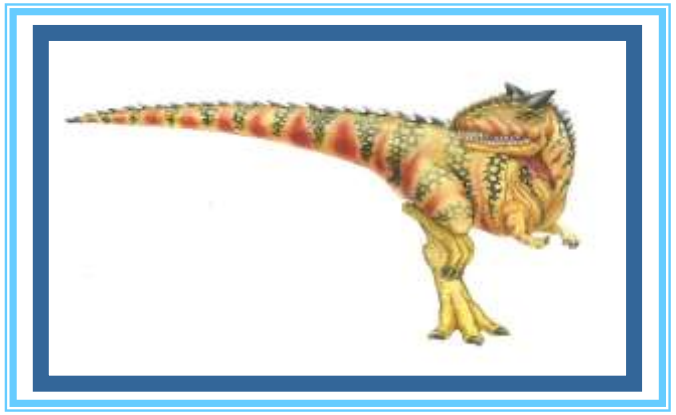
实验习题讲解



Xidian University, Cyber Engineering, Pengbin Feng© 2024

实验1 系统调用基础

- 系统调用基础（p96，2.26 ）

  - 在 2.3 节中，我们描述了**一个将一个文件的内容复制到目标文件的程序**。该程序首先提示用户输入源文件和目标文件的名称。使用 Windows 或 **POSIX API** 编写此程序。请务必包括所有必要的错误检查，包括确保源文件是否存在。
  - 正确设计和测试程序后，如果你使用支持它的系统，请使用跟踪系统调用的实用程序运行该程序。 Linux 系统提供 ***strace*** 实用程序，Solaris 和 Mac OS X 系统使用 dtrace 命令。由于 Windows 系统不提供此类功能，您将不得不使用调试器跟踪此程序的 Windows 版本。

实验1 系统调用基础

- 程序编写
  - 拷贝文件功能
  - 模仿cp命令的执行效果
    - 错误检查（参数检查，程序逻辑，函数返回值判断等）
    - 只考虑文件，不考虑目录
- strace
  - 查看所编写程序执行过程中调用的系统调用
  - strace用来判断一个程序执行出错的原因

实验2 多进程编程

- 多进程编程（**p157，project1**）

Project 1 — UNIX Shell和历史记录

- 该项目包括设计一个 C 程序**作为一个 shell 接口，它接受用户命令，然后在单独的进程中执行每个命令**。这个项目可以在任何 Linux、UNIX 或 Mac OS X 系统上完成。
- shell 界面给用户一个提示，然后输入下一个命令。下面的示例说明了提示符 osh> 和用户的下一个命令：cat prog.c 。 （此命令使用 UNIX cat 命令在终端上显示文件 prog.c。）

```
osh> cat prog.c
```

实验2 多进程编程

- 实现 shell 接口的一种技术是让父进程首先**读取用户在命令行中输入的内容**（在本例中为 cat prog.c），然后**创建一个单独的子进程来执行该命令**。除非另有说明，否则父进程在继续之前等待子进程退出。这在功能上类似于图 3.10 中所示的新流程创建。但是，UNIX shell 通常还允许**子进程在后台或同时运行**。为此，我们在命令末尾添加一个与号（&）。因此，如果我们将上面的命令重写为

```
osh> cat prog.c &
```

父进程和子进程将同时运行。

- 单独的子进程是使用 **fork()** 系统调用创建的，用户的命令是使用系统调用 **exec()** 系列之一执行的（如第 3.3.1 节所述）。

实验2 多进程编程

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

#define MAX_LINE 80 /* The maximum length command */

int main(void)
{
char *args[MAX_LINE/2 + 1]; /* command line arguments */
int should_run = 1; /* flag to determine when to exit program */

   while (should_run) {
      printf("osh>");
      fflush(stdout);

      /**
       * After reading user input, the steps are:
       * (1) fork a child process using fork()
       * (2) the child process will invoke execvp()
       * (3) if command included &, parent will invoke wait()
       */
   }

    return 0;
}
```

实验2 多进程编程

- 用户在 osh> 提示符下输入命令 ps -ael，存储在 args 数组中的值是：

```
args[0] = "ps"
args[1] = "-ael"
args[2] = NULL
```

- 这个 args 数组将被传递给 execvp() 函数，它具有以下原型：

```
execvp (char *command, char *params[]) ;
```

- 这里，command 表示要执行的命令，params 存储此命令的参数。对于这个项目，execvp() 函数应该被调用为 execvp(args [0], args)。请务必检查用户是否包含 & 以确定父进程是否要等待子进程退出。

实验2 多进程编程

- 创建历史记录
  - 下一个任务是修改 shell 接口程序，使其提供**history**功能，允许用户访问最近输入的命令。通过使用该功能，用户**最多可以访问 10 个命令**。命令会从1开始连续编号，超过10会继续编号。例如，如果用户输入了35条命令，则最近的10条命令将编号为26到35。

```
history     6 ps
            5 ls -l
            4 top
            3 cal
            2 who
            1 date
```

1. 当用户输入 !! 执行历史记录中的最新命令。
2. 当用户输入单 ! 后跟整数 N，执行历史记录中的第 N 个命令。

实验2 多进程编程

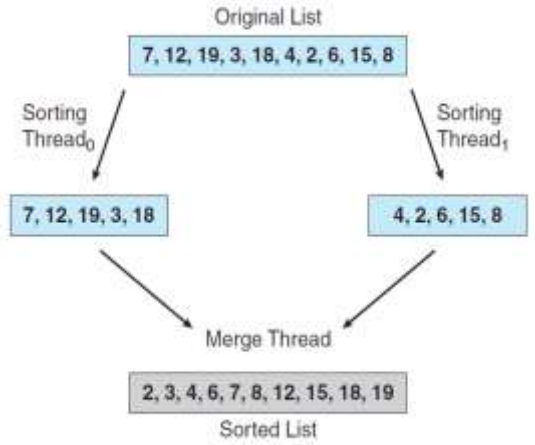
- 该程序还应该管理基本的错误处理。
  - 如果历史中没有命令，输入！！应该会产生一条消息“No commands in history”。
  - 如果没有与用单个！输入的数字对应的命令，程序应该输出“No such command in history”。

实验3 多线程编程

- 多线程编程（**p199，project2 ）**

Project 2 —— 多线程排序应用

- 编写一个**多线程排序程序**，其工作方式如下：一个整数列表被分成两个大小相等的较小列表。两个单独的线程（我们将其称为 ***sorting threads***）使用您选择的排序算法对每个子列表进行排序。这两个子列表随后由第三个线程——a ***merging thread***——它将两个子列表合并为一个排序列表。



实验4 多线程与信号量编程

- 多线程与信号量编程（**p253，project3**）
  - 生产者—消费者问题

在第 5.7.1 节中，我们提出了一种使用有界缓冲区的信号量解决方案来解决生产者消费者问题。你将使用图 5.9 和 5.10 中所示的生产者和消费者进程来设计有界缓冲区问题的编程解决方案。5.7.1 节中介绍的解决方案使用三个信号量：**empty** 和 **full**，它们计算缓冲区中空槽和满槽的数量，以及 **mutex**，它是一个二进制（或互斥）信号量，用于保护实际插入或删除缓冲区中的项目。你将使用标准计数信号量表示**empty**和**full**，并使用互斥锁而不是二进制信号量来表示**mutex**。**生产者和消费者作为单独的线程运行**——将产品移入和移出与**empty**、**full**和**mutex**结构同步的缓冲区。可以使用 **Pthreads** 或 Windows API 实现。

实验4 多线程与信号量编程

```c
#include <pthread.h>

pthread_mutex_t mutex;

/* create the mutex lock */
pthread_mutex_init(&mutex,NULL);
```

```c
/* acquire the mutex lock */
pthread_mutex_lock(&mutex);

/* critical section */

/* release the mutex lock */
pthread_mutex_unlock(&mutex);
```

```c
#include <semaphore.h>
sem_t sem;

/* Create the semaphore and initialize it to 1 */
sem_init(&sem, 0, 1);
```

```c
/* acquire the semaphore */
sem_wait(&sem);

/* critical section */

/* release the semaphore */
sem_post(&sem);
```

实验4 多线程与信号量编程

```
/* buffer.h */
typedef int buffer_item;
#define BUFFER_SIZE 5
```

```
#include "buffer.h"

/* the buffer */
buffer_item buffer[BUFFER_SIZE];

int insert_item(buffer_item item) {
   /* insert item into buffer
    return 0 if successful, otherwise
    return -1 indicating an error condition */
}

int remove_item(buffer_item *item) {
   /* remove an object from buffer
    placing it in item
    return 0 if successful, otherwise
    return -1 indicating an error condition */
}
```

实验4 多线程与信号量编程

```
#include "buffer.h"

int main(int argc, char *argv[]) {
  /* 1. Get command line arguments argv[1],argv[2],argv[3] */
  /* 2. Initialize buffer */
  /* 3. Create producer thread(s) */
  /* 4. Create consumer thread(s) */
  /* 5. Sleep */
  /* 6. Exit */
}
```

实验4 多线程与信号量编程

- 生产者和消费者线程
  - 生产者线程将在**随机休眠一段时间和将随机整数插入缓冲区之间交替**。随机数将使用 rand() 函数产生，该函数产生介于 O 和 RAND_MAX 之间的随机整数。消费者也会随机休眠一段时间，醒来后会尝试从缓冲区中删除一个项目。生产者和消费者线程的概要如图 5.26 所示。
  - 如前所述，您可以使用 **Pthreads** 或 Windows API 解决此问题。在以下部分中，我们将提供有关每个选择的更多信息。

实验4 多线程与信号量编程

```
#include <stdlib.h> /* required for rand() */
#include "buffer.h"

void *producer(void *param) {
   buffer_item item;

   while (true) {
      /* sleep for a random period of time */
      sleep(...);
      /* generate a random number */
      item = rand();
      if (insert_item(item))
         fprintf("report error condition");
      else
         printf("producer produced %d\n",item);
}
```

```
void *consumer(void *param) {
   buffer_item item;

   while (true) {
      /* sleep for a random period of time */
      sleep(...);
      if (remove_item(&item))
         fprintf("report error condition");
      else
         printf("consumer consumed %d\n",item);
}
```

实验5 文件系统

- 文件系统（**p583，programming problem** ）

- 下面的练习检查了 UNIX 或 Linux 系统上**文件和 inode 之间的关系**。在这些系统上，文件用 inode 表示。也就是说，一个 inode 是一个文件（反之亦然）。可以在随本文提供的Linux虚拟机上完成此练习。也可以在任何 Linux、UNIX 或 Mac OS X 系统上完成练习，但需要创建两个名为 file1.txt 和 file3.txt 的简单文本文件，其内容是唯一的句子。

实验6 Linux操作系统实例研究报告

- 阅读教材第18章（Linux案例），并在互联网上查阅相关资料，对照操作系统课程中所讲的原理（进程管理，存储管理，文件系统，设备管理），了解Linux操作系统实例
- 形成一份专题报告
  - 可以是全面综述性报告
  - 可以是侧重某一方面的报告（进程调度，进程间通信，存储管理，文件系统，安全）
- 阅读教材第19章（可选，Windows案例）

实验7 Linux内核编译

- 下载、编译内核源代码[1][2]
- 启动测试所编译出来的内核
- 使用Clang编译内核*
- 成功配置Linux Kernel静态分析工具[3] *

[1] https://www.kernel.org/doc/html/latest/kbuild/makefiles.html
[2] https://0xax.gitbooks.io/linux-insides/content/Misc/linux-misc-2.html
[3] https://github.com/umnsec/crix

参考

- https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2014_2015/pk1415-introduction.pdf
- https://linuxhint.com/linux-kernel-tutorial-beginners/