嵌入式linux应用期末复习

第一章 Linux基础--第四章 嵌入式系统基础

1. Windows和Linux的区别

   Linux的文件系统和Windows的文件系统有很大的区别：

   - Windows的文件系统是以驱动器的盘符为基础的，而且每一个目录与相应的分区对应。
   - 而Linux恰好相反，文件系统是一颗文件树，且他的所有文件和外部设备都是以文件的形式挂在这个文件树上。
   - 总之，在Windows下，目录结构属于分区；Linux下分区属于目录结构。
   - Linux中把每一个分区和某一个目录相对应，以后再对这个目录的操作就是对这个分区的操作，这就实现了硬件管理手段和软件目录管理手段的统一。这个把分区和目录对应的过程叫做挂载（Mount），而这个挂载在文件树中的位置就是挂载点。

2. SWAP交换分区及其作用

   在硬件条件有限的情况下，为了运行大型的程序， Linux在硬盘上划出一个区域来当作临时的内存，而Windows操作系统把这个区域叫做虚拟内存，Linux把它叫做交换分区swap，在安装Linux建立交换分区时，一般将其设为内存大小的2倍。

3. 编译的过程：

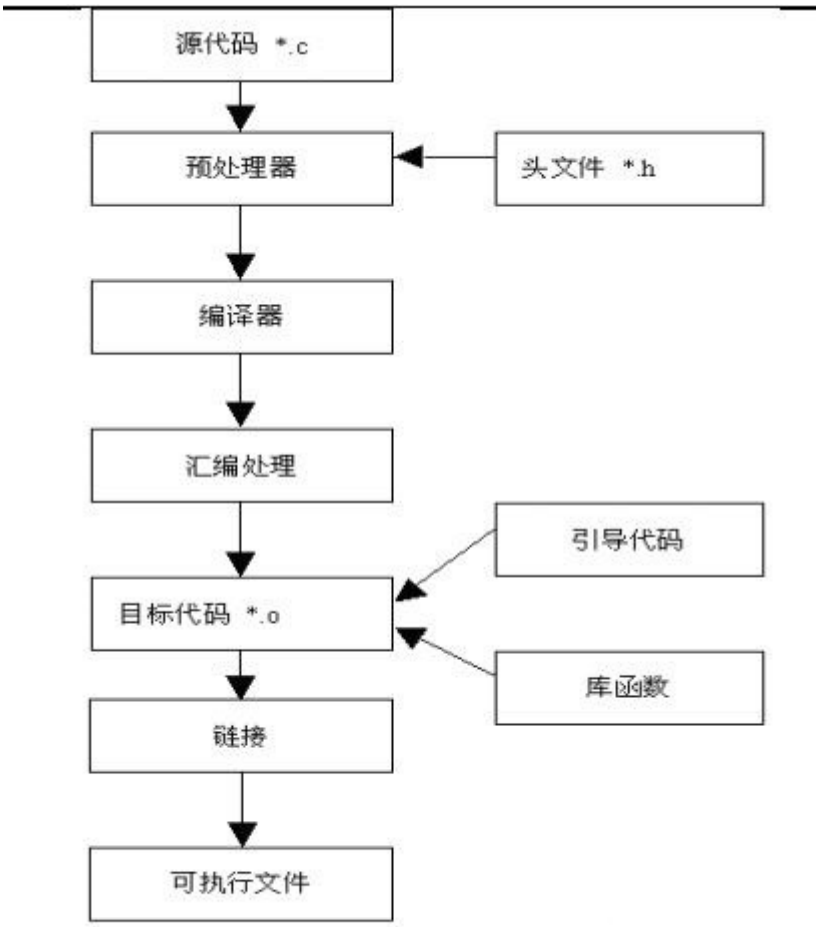


图 3.1　编译过程

4. vi的三种模式：(1) 命令行模式 (2) 插入模式 (3) 底行模式

   vi 文件名 进入命令行模式，按 ‘i’进入插入模式 ，按esc 退出插入模式，最后输入：wq保存

5. gcc的编译流程：

   预处理、编译、汇编、链接

   gcc -g hello.c -o hello

   ```
   -c:只编译不链接，生成目标文件".o"
   -S:只编译不汇编，生成汇编代码；
   -E:只进行预编译，不做其他处理；
   -g:在可执行程序中包含了标准调试信息。
   ```

6. 静态库和动态库定义以及建立P64

   静态库是一系列的目标文件(.o文件)的归档文件(.a)，如果在编译某个程序是链接静态库，则连接器将会搜索静态库，从中提取他所需要的目标文件并直接复制到该程序的可执行文件之中。

   动态库在程序编译时并不会被连接到目标代码中，而是在程序运行时才被载入。

7. gdb调试器

   ```
   gcc -g test.c -o test
   gdb test  //进入调试器
   l        //表示插入载入的文件按行显示
   b 6      //设置断点在第6行
   r        //运行调试程序
   p n      //查看变量n
   p i      //查看变量i
   s        //进入函数单步运行
   n        //不进入函数单步运行
   c        //恢复程序运行
   ```

8. Make 工程管理器

   方便工程编译，依赖于makefile文件，当文件夹中有makefile才能编译

9. autotools
   - autoscan
   - autoconf
   - autoheader
   - automake
   - 运行configure

第六章 文件I/O编程

1. 系统调用

   所谓系统调用是指操作系统提供给用户程序调用的一组“特殊”接口，用户

   程序可以通过这组“特殊”接口来获得操作系统内核提供的服务。

2. 用户编程接口

   在linux中，用户编程接口（API）遵循了在UNIX中最流行的应用编程界面标准。

   描述了操作系统的系统调用编程接口（实际上就是API）用于保证应用程序可以在源代码一级上在多种操作系统上移植运行。主要通过C库实现的。

3. 系统命令

   他实际上是一个可执行的程序，它的内部引用了用户编程接口（API）来实现相应的功能。

4. 文件描述符

   一个进程启动都会打开3个文件：标准输入(0)，标准输出(1)和标准出错处理(2)。

   宏替换：STDIN_FILENO、STDOUT_FILENO、STDERR_FILENO。

5. copy_file.注释:(P157)

0	无任何权限	1	只执行
2	只写	3	写和执行
4	只读	5	读和执行
6	读写	7	读写和执行

```
/* 以只写方式打开目标文件，若此文件不存在则创建，访问权限值为644，
即user:读写、group:读、others:读权限*/（下面分别是usr读写，grp读，oth读）
dest_file = open(DEST_FILE_NAME, O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
```

```
/* 将源文件的读写指针移到最后10KB的起始位置*/
lseek(src_file, -OFFSET, SEEK_END);/*以SEEK_END为起点(文件的结尾)，向前偏移10K字节，定为当前文件读写位置*/
OFFSET：设置每一次读写操作移动的距离，可正可负（-OFFSET向前，+OFFSET向后，define OFFSET 10240 /*大小是10k*/）
SEEK_SET:当前位置的开头；SEEK_CUR：当前位置为文件指针位置；SEEK_END:文件的结尾
```

1. 文件锁
   - 上文件锁是为了避免共享的资源产生竞争的方法。(分为建议性锁和强制性锁)

     上锁的函数有fcntl()和lockf()。

- fcntl()函数
  - *IF_SETLKW：在无法获取锁时,进入睡眠；如果可获得锁或捕捉到信号，则返回*
  - *IF_GETLK:根据lock参数值，决定是否上锁；只有处于解锁状态，才能上锁*

可以施加建议性锁还可以加强制性锁，还能对文件的某一记录上锁，即记录锁。

记录锁还分为读取锁和写入锁，读取锁又称共享锁，它能够使多个进程都能在文件的同一个

部分建立读取锁。而写入锁又称排斥锁，在任何一个时刻，只能有一个进程在文件的某一

部分上排斥锁。在文件的同一部分不能同时建立读取锁和写入锁。

- lock()函数
  - F_RDLCK ：给文件上读取锁(共享锁)
  - F_WRLCK ：给文件上写入锁(排斥锁)
  - F_UNLCK：给文件解锁

```
lock_set.c 判断是否上锁
fcntl(fd, F_SETLKW, &lock)
/*F_SETLKW：在无法获取锁时,进入睡眠；如果可获得锁或捕捉到信号，则返回*/
/*F_GETLK:根据lock参数值，决定是否上锁；只有处于解锁状态，才能上锁*/
/*&lock:取flock结构体变量lock的首地址*/
```

```
write_lock.c
fd = open("hello",O_RDWR | O_CREAT, 0644);
/* 首先只读写权限打开文件，文件权限是664(user 6;group 4,other 4) */
/*O_CREAT:文件不存在创建以新文件*/
lock_set(fd, F_WRLCK);/* 给文件上写入锁 */
getchar();
lock_set(fd, F_UNLCK);/* 给文件解锁 */
```

```
read_lock.c
lock_set(fd, F_RDLCK);/* 给文件上读取锁，上锁后程序处于等待键盘输入字符的等待状态； */
getchar();/*取键盘键入的字符，可以让程序继续执行下去*/
lock_set(fd, F_UNLCK);/* 给文件解锁 */
```

- select()函数
  - FD_ZERO(&inset); *//初始化文件描述符集inset，并在该集合中加入相应的三个文件描述符：标准输入，in1和in2的文件描述符*
  - FD_SET()：将一个文件描述符加入文件描述集中。
  - FD_ISSET(fds[0],&inset) 如果文件描述符fd为fd_set集中的一个元素，则返回值非零值，可以用于调用select()之后测试文件描述符集中的文件描述符是否有变化。

```
while(FD_ISSET(fds[0],&inset)||FD_ISSET(fds[1],&inset)||FD_ISSET(fds[2], &inset))
    {
        tmp_inset = inset;/*文件描述符集合的备份，避免每次进行初始化*/
        res = select(maxfd + 1, &tmp_inset, NULL, NULL, &tv);
         /*maxfd+1:需要监视的文件描述符最大值+1，减少监控二进制位数，提高效率*/
         /* &tmp_inset：由select()监视的读文件描述符集合*/
         /*NULL:由select()监视的写文件描述符集合*/
         /*NULL:由select()监视的异常出错文件描述符集合*/
         /*&tv:若等待了tv时间还没有检测到任何文件描述符准备好，则立即返回*/
         /*select()执行成功则返回准备好的文件描述符的数目，如果返回-1代表出错
该函数执行过后只有数据输入的文件描述符才能继续存在tmp_inset文件描述符集当中，没有输入的则清除出
tmp_inset*/
```

第七章 进程控制开发

1. 三种进程状态的转化：

   - 执行态：该进程正在运行，即进程正在占用CPU。
   - 就绪态：进程已经具备执行的一切条件，正在等待分配CPU的处理时间片。
   - 等待态：进程不能使用CPU，若等待事件发生（等待的资源分配到）则可将其唤醒。

2. fork()函数的作用：

   fork()函数用于从已存在的进程中创建一个新进程。新进程称为子进程，而原进程称为父进程。

   实际上是在父进程中执行fork()函数时，父进程会复制出一个子进程，而且父子进程的代码从fork()函数的返回开始分别在两个地址空间中同时运行。（返回值为0：子进程；返回值为大于0的整数：父进程；-1：出错）

3. exec函数族：

   exec函数族就提供了一个在进程中启动另一个程序执行的方法。它可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，其他全部被新的进程替换了。

4. exit()和_exit() 函数：

   在于exit()函数在调用exit系统之前要检查文件的打开情况，把文件缓冲区中的内容写回文件，而_exit()函数直接将进程关闭，缓冲区中的数据就会丢失。

5. wait()和waitpid() 函数：

   wait()函数是用于使父进程（也就是调用wait()的进程）阻塞，直到一个子进程结束或者该进程接到了一个指定的信号为止。如果该父进程没有子进程或者他的子进程已经结束，则wait()就会立即返回。

   waitpid()的作用和wait()一样，但它并不一定要等待第一个终止的子进程，它还有若干选项。

   例题：

```
pr = waitpid(pc, NULL, WNOHANG);
/*若子进程还未退出，则父进程暂停1s*/
    if (pr == 0)
    {
        printf("The child process has not exited\n");
         sleep(1);
    }
换成waitpid(pid, NULL, 0);/*阻塞父进程，等待子进程退出，此时父进程不能做什么。*/
```

6. Linux守护进程

   编写守护进程的步骤：

   - 创建子进程，父进程退出；
   - 在子进程中创建新会话；
   - 改变当前目录为根目录；
   - 重设文件权限掩码；
   - 关闭文件描述符；

第八章进程间通信

1. 管道
   - 用于具有亲缘关系的进程之间的通信（也就是父子或者兄弟进程之间）
   - 半双工通信模式，有固定的读写段
   - 管道可以看作是一种特殊的文件，她不属于文件系统，只存在于内核的内存空间。
   - 创建管道的步骤：
     - 用pipe()函数创建一个管道；
     - 在通过fork()函数创建一个子进程(子进程继承父进程所创建的管道)；
     - 父子进程关闭不需要的文件描述符close()；
     - 然后父子进程之间就建立起了一条“子进程写入父进程读取”的通道。
   - 有名管道使用步骤：
     - 创建管道文件mkfifo();
     - 在某个进程中以只写方式打开管道文件, 并写管道;
     - 在某个进程中以只读方式打开管道文件, 并读管道;
     - 关闭管道文件;

     ```
     -对于读进程
     - 若该管道是阻塞打开，且当前FIFO内没有数据，则对读进程而言将一直阻塞到有    数据写入。
     - 若该管道是非阻塞打开，则不论FIFO内是否有数据，读进程都会立即执行读操    作。即如果FIFO内没有数据，则读函数将立刻返回0。
     ```

     ```
     -对于写进程
     - 若该管道是阻塞打开，则写操作将一直阻塞到数据可以被写入。
     -若该管道是非阻塞打开而不能写入全部数据,则写操作进行部分写入或者调用失败。
     ```

   - 管道的工作流程：

```
/* 创建一子进程 */
    if ((pid = fork()) == 0)
/*子进程关闭写描述符，并通过使子进程暂停1秒等待父进程已关闭相应的读描述符 */
        close(pipe_fd[1]);/*pipe_fd[1]：子进程的写操作，为‘0’时表示读*/
        sleep(DELAY_TIME * 3);

        /* 子进程读取管道内容 */
        if ((real_read = read(pipe_fd[0], buf, MAX_DATA_LEN)) > 0)
        /* 关闭子进程读描述符 */
        close(pipe_fd[0]);
    else if (pid > 0)/*(当pid的值是0时表示是子进程，大于1时表示父进程)*/
/* 父进程关闭读描述符，并通过使父进程暂停1秒等待子进程已关闭相应的写描述符 */
        close(pipe_fd[0]);
        sleep(DELAY_TIME);

        /* 父进程向管道中写入字符串 */
        if((real_write = write(pipe_fd[1], data, strlen((const char*)data))) !=  -1)

        /*关闭父进程写描述符*/
        close(pipe_fd[1]);

        waitpid(pid, NULL, 0);
    /*父进程阻塞，等待子进程退出，不退出父进程不能干任何事*/
```

- 标准流管道：
  - 一系列的创建过程合并到一个函数popen()中完成
  - 大大减少代码的编写量，
  - 用popen()创建的管道必须使用标准I/O函数进行操作
  - 将管道的读端或写端分别映射显示器或键盘的管道就是标准流管道。

1. 信号
   - kill()函数

     不仅可以中止进程（实际上发出SIGKILL信号），也可以向进程发送其他信号。
   - raise()函数

     允许进程自身发送信号（实际上发出SIGSTOP信号），让进程暂停。
   - 信号量处理函数
     - signal函数：使用函数处理时，只需要指出要处理的信号和处理函数即可

       SIG_IGN（忽略信号），SIG_DFL(采用系统默认的方式处理。)
     - sigaction函数：更健壮，更新的信号处理函数，sa_handler是一个函数指针，指定信号处理函数。
   - 信号集函数组

     sigemptyset()：将信号集合初始化为空。

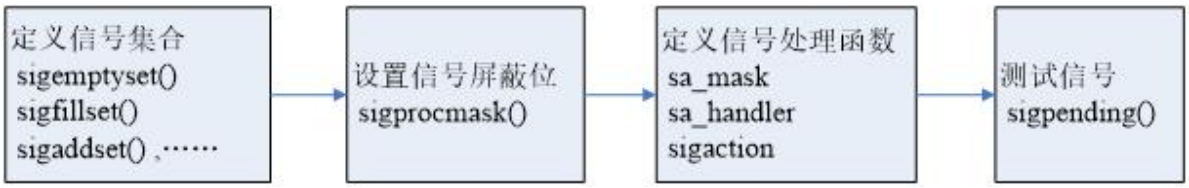
     sigfillset()：将信号集合初始化为包含所有已定义的信号的集合。

     sigaddset()：将指定信号加入到信号集合中去。

     sigdelset()：将指定信号从信号集合中删去。

     sigismember()：查询指定信号是否在信号集合之中。

一般信号操作处理流程：



初始化信号-> 设置信号集屏蔽字,目前处于set信号集的信号处于阻塞状态，进程当前得不到这些信号

2. 信号量

进程之间的互斥与同步关系存在的根源在于临界资源，访问临界资源的代码叫做临界区，临界区本身也会成为临界资源。

- P操作：如果有可用的资源（信号量值>0），则占用一个资源（给信号量值减去一，进入临界区代码
- V操作：如果在该信号量的等待队列中有进程在等待资源，则唤醒一个阻塞进程。如果没有进程等待它，则释放一个资源（给信号量值加一）

4.共享内存

P263代码，P277 程序

5.消息队列

消息队列的实现包括创建或打开消息队列(msgget())、添加消息(msgsnd())、读取消息(msgrcv())和控制消息队列(msgctl())这四种操作。