此文章已于 14:33:48 2016/6/21 发布到 MADHEX

Linux内核完全注释笔记

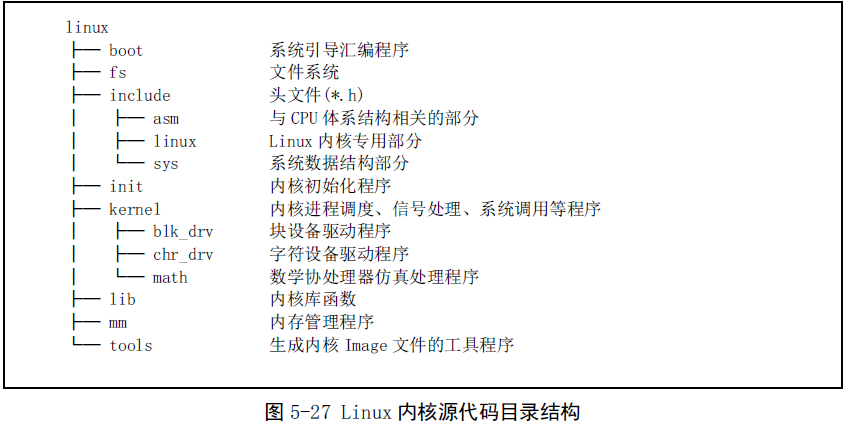
# 概述

# 微型计算机组成结构

# 内核编程语言和环境

# 80X86保护模式及其编程

# Linux内核体系结构



# 引导启动程序(boot)

# 初始化程序(init)

/\* 内核初始化主程序。初始化结束后将以任务0 (idle任务即空闲任务)的身份运行。 \*/

**void** main(**void**) /\* This really IS void, no error here. \*/

{ /\* The startup routine assumes (well, ...) this \*/

/\* 这里确实是void，并没错。在startup 程序(head.s)中就是这样假设的。

\* 参见head.s 程序第136 行开始的几行代码。 \*/

trap\_init (); /\* 陷阱门（硬件中断向量）初始化。（kernel/traps.c，181 行） \*/

tty\_init (); /\* tty 初始化。（kernel/chr\_dev/tty\_io.c，105 行） \*/

sched\_init (); /\* 调度程序初始化(加载了任务0 的tr, ldtr) （kernel/sched.c，385） \*/

buffer\_init (buffer\_memory\_end);/\* 缓冲管理初始化，建内存链表等。（fs/buffer.c，348） \*/

sti (); /\* 所有初始化工作都做完了，开启中断。 \*/

/\* 下面过程通过在堆栈中设置的参数，利用中断返回指令切换到任务0。 \*/

move\_to\_user\_mode (); /\* 移到用户模式。（include/asm/system.h，第1 行） \*/

**if** (!fork())

{ /\* we count on this going ok \*/

init();

}

/\*

\* NOTE!! For any other task 'pause()' would mean we have to get a

\* signal to awaken, but task0 is the sole exception (see 'schedule()')

\* as task 0 gets activated at every idle moment (when no other tasks

\* can run). For task0 'pause()' just means we go check if some other

\* task can run, and if not we return here.

\*/

/\* 注意!! 对于任何其它的任务，'pause()'将意味着我们必须等待收到一个信号才会返

\* 回就绪运行态，但任务0（task0）是唯一的意外情况（参见'schedule()'），因为任务0 在

\* 任何空闲时间里都会被激活（当没有其它任务在运行时），因此对于任务0'pause()'仅意味着

\* 我们返回来查看是否有其它任务可以运行，如果没有的话我们就回到这里，一直循环执行'pause()'。

\*/

**for**(;;) pause();

}

**void** init(**void**)

{

**int** pid,i;

/\* 读取硬盘参数包括分区表信息并建立虚拟盘和安装根文件系统设备。 \*/

/\* 该函数是在25 行上的宏定义的，对应函数是sys\_setup()，在kernel/blk\_drv/hd.c,71 行。 \*/

setup((**void** \*) &drive\_info);

(**void**) open("/dev/tty0",O\_RDWR,0); /\* 用读写访问方式打开设备“/dev/tty0”， \*/

/\* 这里对应终端控制台。 \*/

/\* 返回的句柄号0 -- stdin 标准输入设备。 \*/

(**void**) dup (0); /\* 复制句柄，产生句柄1 号 -- stdout 标准输出设备。 \*/

(**void**) dup (0); /\* 复制句柄，产生句柄2 号 -- stderr 标准出错输出设备。 \*/

printf("%d buffers = %d bytes buffer space**\n\r**",NR\_BUFFERS,NR\_BUFFERS\*BLOCK\_SIZE); /\* 打印缓冲区块数和总字节数，每块1024 字节。 \*/

printf("Free mem: %d bytes**\n\r**",memory\_end-main\_memory\_start); /\*空闲内存字节数。 \*/

/\* 下面fork()用于创建一个子进程(子任务)。 \*/

/\* 对于被创建的子进程，fork()将返回0 值， 对于原(父进程)将返回子进程的进程号。 \*/

/\* 所以180-184 句是子进程执行的内容。 \*/

/\* 该子进程关闭了句柄0(stdin)，以只读方式打开/etc/rc 文件，并执行/bin/sh 程序，所带参数和 \*/

/\* 环境变量分别由argv\_rc 和envp\_rc 数组给出。参见后面的描述。 \*/

**if** (!(pid=fork())) {

close(0);

**if** (open("/etc/rc",O\_RDONLY,0))

\_exit(1); /\* 如果打开文件失败，则退出(/lib/\_exit.c,10)。 \*/

execve("/bin/sh",argv\_rc,envp\_rc); /\* 装入/bin/sh 程序并执行。 \*/

\_exit(2); /\* 若execve()执行失败则退出(出错码2,“文件或目录不存在”)。 \*/

}

/\* 下面是父进程执行的语句。 \*/

/\* wait()是等待子进程停止或终止，其返回值应是子进程的进程号(pid)。这三句的作用是父进程等待子进程的结束。 \*/

/\* &i 是存放返回状态信息的位置。 如果wait()返回值不等于子进程号，则继续等待。 \*/

**if** (pid>0)

**while** (pid != wait(&i))

/\* nothing \*/;

/\* 如果执行到这里，说明刚创建的子进程的执行已停止或终止了。下面循环中首先再创建一个子进程，

\* 如果出错，则显示“初始化程序创建子进程失败”的信息并继续执行。对于所创建的子进程关闭所有

\* 以前还遗留的句柄(stdin, stdout, stderr)，新创建一个会话并设置进程组号，然后重新打开

\* /dev/tty0 作为stdin，并复制成stdout 和stderr。

\* 再次执行系统解释程序/bin/sh。但这次执行所选用的参数和环境数组另选了一套（见上面165-167 行）。

\* 然后父进程再次运行wait()等待。如果子进程又停止了执行，则在标准输出上显示出错信息“子进程pid 停

\* 止了运行，返回码是i”，然后继续重试下去…，形成“大”死循环。

\*/

**while** (1) {

**if** ((pid=fork())<0) {

printf("Fork failed in init**\r\n**");

continue;

}

**if** (!pid) {

close(0);close(1);close(2);

setsid();

(**void**) open("/dev/tty0",O\_RDWR,0);

(**void**) dup(0);

(**void**) dup(0);

\_exit(execve("/bin/sh",argv,envp));

}

**while** (1)

**if** (pid == wait(&i))

break;

printf("**\n\r**child %d died with code %04x**\n\r**",pid,i);

sync();

}

\_exit(0); /\* NOTE! \_exit, not exit() \*/

}

。