嵌入式技术

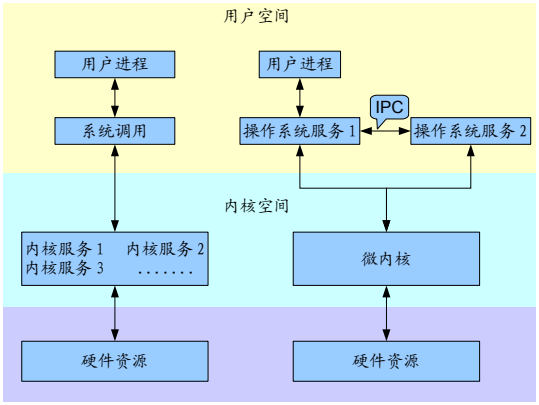
GNU/Linux 进程

邢超

<EC.1>

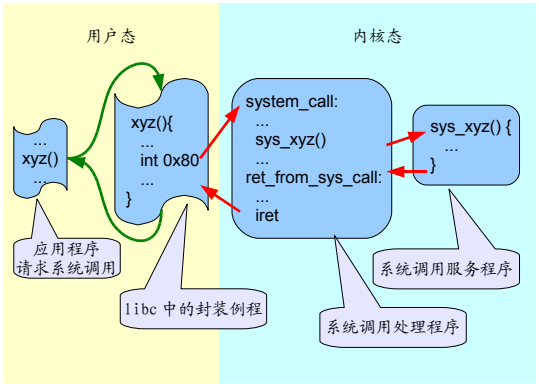
1 Linux 内核特点

宏内核结构与微内核结构



<EC.2>

系统调用



<EC.3>

中断处理与定时器

- 中断
  - 外部设备生成的中断
  - 软件程序生成的中断 (陷阱 trap)
- 中断描述符表 (IDT)
  - 中断编号
  - 入口指针

<EC.4>

2 进程调度

进程

- 进程是执行程序的一个实例
- 进程和程序的区别
  - 几个进程可以并发的执行一个程序，这些进程共享内存中程序正文的单一副本，但每个进程有自己的单独的数据和堆栈区
  - 一个进程可以顺序的执行几个程序，一个进程可以在任何时刻执行新的程序，并且在它的生命周期中可以运行几个程序

<EC.5>

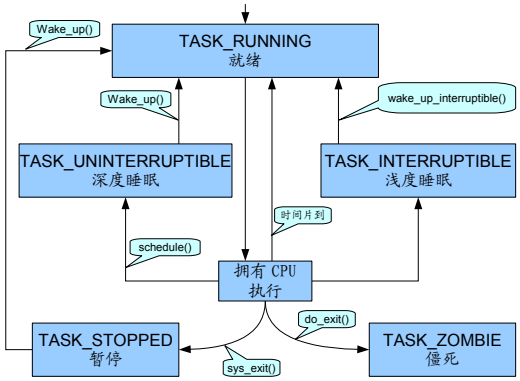
进程状态

- 就绪态：
  - 进程已经获得所有所需的其他资源；
  - 正在申请处理机资源，准备运行。
- 阻塞态 (休眠状态，或等待状态)：
  - 进程因需要等待所需资源而放弃处理机；
  - 或者进程本不拥有处理机，且其他资源也没有满足 (从而即使得到处理机资源也不能开始运行)。
- 运行态：

- 指进程得到了处理机，不需要等待其他任何资源，正在执行的状态；
- 此时进程才可使用申请的资源

<EC.6>

进程调度



<EC.7>

进程编号

- 进程以进程号 PID(process ID) 作为标识。任何对进程的操作都要有相应的 PID 号。
- 每个进程都属于一个用户，进程要配备其所属的用户编号 UID。
- 每个进程都属于多个用户组，所以进程还要配备其归属的用户组编号 GID 的数组。

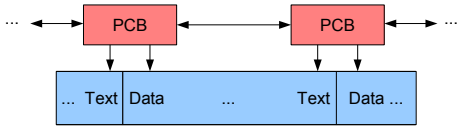
<EC.8>

进程切换: 进程上下文

- 运行进程的环境称为进程上下文 (context)，包含了进程执行需要的所有信息
- 包括：
  - 用户地址空间：包括程序代码，数据，用户堆栈等
  - 控制信息：进程描述符，内核堆栈等
  - 硬件上下文
    * 通用寄存器, 如 eax，ebx 等
    * 系统寄存器, 如 eip，esp，cr3 等等

<EC.9>

进程表 (process table)



PCB(Process Control Block) 包括进程的编号、状态、优先级以及正文段和数据段中数据分布的大概情况。
正文段(text segment)存放该进程的可执行代码。
数据段(data segment)存放进程静态产生的数据结构

<EC.10>

进程的创建

- 系统启动时，自行创建了 0 号进程 (init_task())。作为一切其他进程的父进程。
  - 系统调用 fork
  - 系统调用 vfork
  - 系统调用 clone
- 执行一个新程序
  - 系统调用 exec
- 终止进程
  - 系统调用 exit
  - 进程也可以因收到信号而终止

<EC.11>

fork.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv){
  printf("father start\n");
  pid_t self=getpid();
  pid_t pid=fork();//child process
      start here
  printf("child start\n");
  if(pid>0){
    printf("Father: %d, child:%d\n",
        self,pid);
    exit(0);
  }
  if(pid==0){
    printf("Child: %d, father: %d\n",(
        int)getpid(),self);
    exit(0);
  }
  if(pid<0){
    printf("Error, Create Failure!\n")
        ;
    exit(1);
  }
}
```

<EC.12>

fork

- 调用 fork 的进程称为父进程
- 新进程是子进程
- 子进程的地址空间是父进程的复制，一开始也是运行同一程序。
- 为父子进程返回不同的值

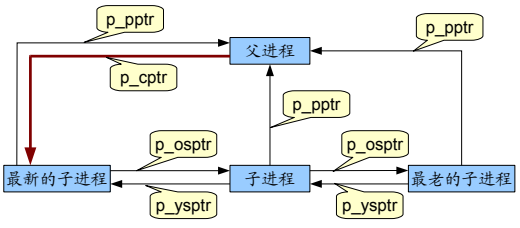
<EC.13>

写时复制技术

- 背景
  - 很多情况下，子进程从 fork 返回后很多会调用 exec 来开始执行新的程序
  - 这种情况下，子进程根本不需要读或者修改父进程拥有的所有资源。
  - 所以 fork 中地址空间的复制依赖于 Copy On Write 技术，降低 fork 的开销
- 实现方法
  - 写时复制技术允许父子进程能读相同的物理页。
  - 只要两者有一个进程试图写一个物理页，内核就把这个页的内容拷贝到一个新的物理页，并把这个新的物理页分配给正在写的进程

<EC.14>

进程间关系



<EC.15>

3 信号

Linux 系统信号

- 信号主要用于通知进程异步事件的发生。
- 进程可以用 kill 或 killpg 系统调用向另一个进程发信号。
- 进程可以通过提供信号处理函数来取代对于任意信号的缺省反应，这种缺省反应一般都是终止进程。
- 信号发生时，内核中断当前的进程，进程执行处理函数来响应信号，结束后恢复正常的进程处理。

<EC.16>

4 进程间通信

进程间通信

- 管道机制：
  - 源进程向管道写数据，而内核会自动将这些数据引导向目标进程
  - 写之前必须关闭读管道，反之亦然。
- 先入先出机制：
  - FIFO 为“first in，first out”的简写，指一个在磁盘上的文件，它可以被所有进程所共享。
  - FIFO 与一般文件不同，它还使用了内核中的缓冲区，所以在效率上要比一般共享文件快得多。
  - FIFO 和管道都可以使用 read() 和 write() 调用来进行读写操作。
- IPC(Interprocess Communication) 机制：
  - 信号量：用于同步
  - 消息队列：由内核创建并维护的一个数据结构，
  - 共享内存：将一段内存区映射到一个进程的地址空间来实现，进程间通信不再涉及到内核

<EC.17>

5 信号

Linux 系统信号

- 信号主要用于通知进程异步事件的发生。
- 进程可以用 kill 或 killpg 系统调用向另一个进程发信号。
- 进程可以通过提供信号处理函数来取代对于任意信号的缺省反应，这种缺省反应一般都是终止进程。
- 信号发生时，内核中断当前的进程，进程执行处理函数来响应信号，结束后恢复正常的进程处理。

<EC.18>

Linux 系统信号

信号	说明	信号	说明
SIGHUP	连接断开终止进程	SIGINT	响应键盘中断终止
SIGQUIT	中断退出信号	SIGILL	非法终止
SIGTRAP	硬件故障终止进程	SIGIOT	硬件故障终止进程
SIGBUS	硬件故障终止进程	SIGFPE	算术异常浮点例外
SIGKILL	非法硬件指令	SIGUSR1	用户定义信号 1
SIGUSR2	用户定义信号 2	SIGPIPE	写至无读进程的管道，终止进程
SIGALRM	超时退出	SIGTERM	终止信号
SIGCHLD	子进程改变状态	SIGCONT	使挂起进程继续
SIGSTOP	停止进程，不能被捕获	SIGTSTP	终端挂起符
SIGTTIN	后台从控制终端读	SIGTTOU	后台向控制终端写
SIGURG	紧急状况	SIGXCPU	超过 CPU 限制退出
SIGXFSZ	超过文件长度限制	SIGVTALRM	虚拟时间闹钟
SIGPROF	梗概时间超时	SIGWINCH	终端窗口大小改变
SIGIO	异步 I/O	SIGPWR	电源失效，重启

<EC.19>

信号产生条件

- 用户按下特定的键后，将向该终端前台进程组发送信号
- 硬件异常会产生信号：如被 0 除、无效内存引用等。
- kill(2) 系统调用允许进程向其他进程发送任意信号
- kill(1) 命令允许用户向进程发送任意信号。
- 软件设置的条件，如 SIGALARM。

<EC.20>

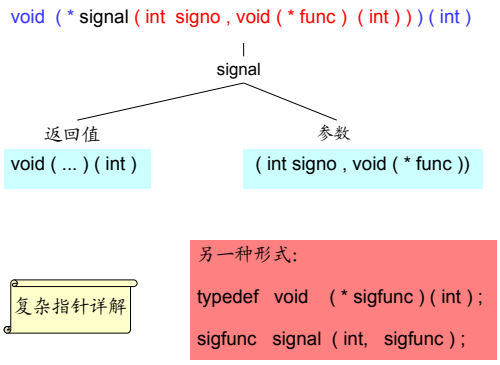
信号捕捉

void (*signal (int signo，void (*func)(int)))(int)

- 其中 signo 是系统信号表中的信号名；
- func 的值是 SIG_IGN 或 SIG_DFL 或者是接到此信号后需要调用的函数地址
  - 当 func 为 SIG_IGN 时则向内核表示忽略此信号。
  - 当 func 为 SIG_DFL 表示接到此信号的动作是系统默认动作。
  - 当 func 为函数地址时为捕捉信号。出错则返回 SIG_ERR，成功返回信号处理配置。

<EC.21>

复杂指针详解



<EC.22>

发送信号

- kill(pid_t pid，int sig)；
  - pid>0 是将信号发送到 PID 为 pid 的进程
  - pid=0 信号发送到与发送进程在同一进程组的进程
  - pid<-1 将信号发送到进程组 ID 等于 -pid 的进程
  - pid=-1Linux 发到进程表中除第一个进程外的进程
  - sig=0 时，不发送任何信号，但仍然执行错误检查
- int raise(int sig)；//向当前进程发送信号，
  - 等价于 kill(getpid()，sig)，成功返回为 0，出错为 -1。
- unsigned int alarm(unsigned int seconds)；
  - 此函数用来设置一个时间值 (闹钟时间)，当所设置的值被超过后，产生 SIGALRM 信号，默认动作是终止进程。
- int pause(void)；
  - 可以使进程挂起，直到捕捉到一个信号。
- unsigned int sleep(unsigned int seconds)；
  - 此函数挂起调用中的进程，直到过了预定时间或者是收到一个信号并从信号处理程序返回。

<EC.23>

alarm.c

```
#include <unistd.h>
#include <signal.h>
void handler(){
        printf("hello\n");
}
main(){
 int i;
 signal(SIGALRM, handler);
 alarm(2);
 for(i=1;i<4;i++){
   printf("sleep␣%d␣␣\n",i);
   sleep(1);
 }
}
```

<EC.24>

6 思考

思考

- 进程与程序的区别
- 进程间通信的实现

<EC.25>