

嵌入式系统工程师

# 进程

# 大纲

- 进程概述
- 进程控制

- 进程概述
  - 进程的定义
  - 进程的状态及转换
  - 进程控制块
- 进程控制

# 进程概述

## ➤ 进程的定义

### ➤ 程序:

程序是存放在存储介质上的一个可执行文件。

### ➤ 进程:

进程是程序的执行实例，包括程序计数器、寄存器和变量的当前值。

### ➤ 程序是静态的，进程是动态的:

程序是一些指令的有序集合，而进程是程序执行的过程。进程的状态是变化的，其包括进程的创建、调度和消亡。

- 进程概述
  - 进程的定义
  - 进程的状态及转换
  - 进程控制块
- 进程控制

# 进程概述

➤ 进程整个生命周期可以简单划分为三种状态:

➤ 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。

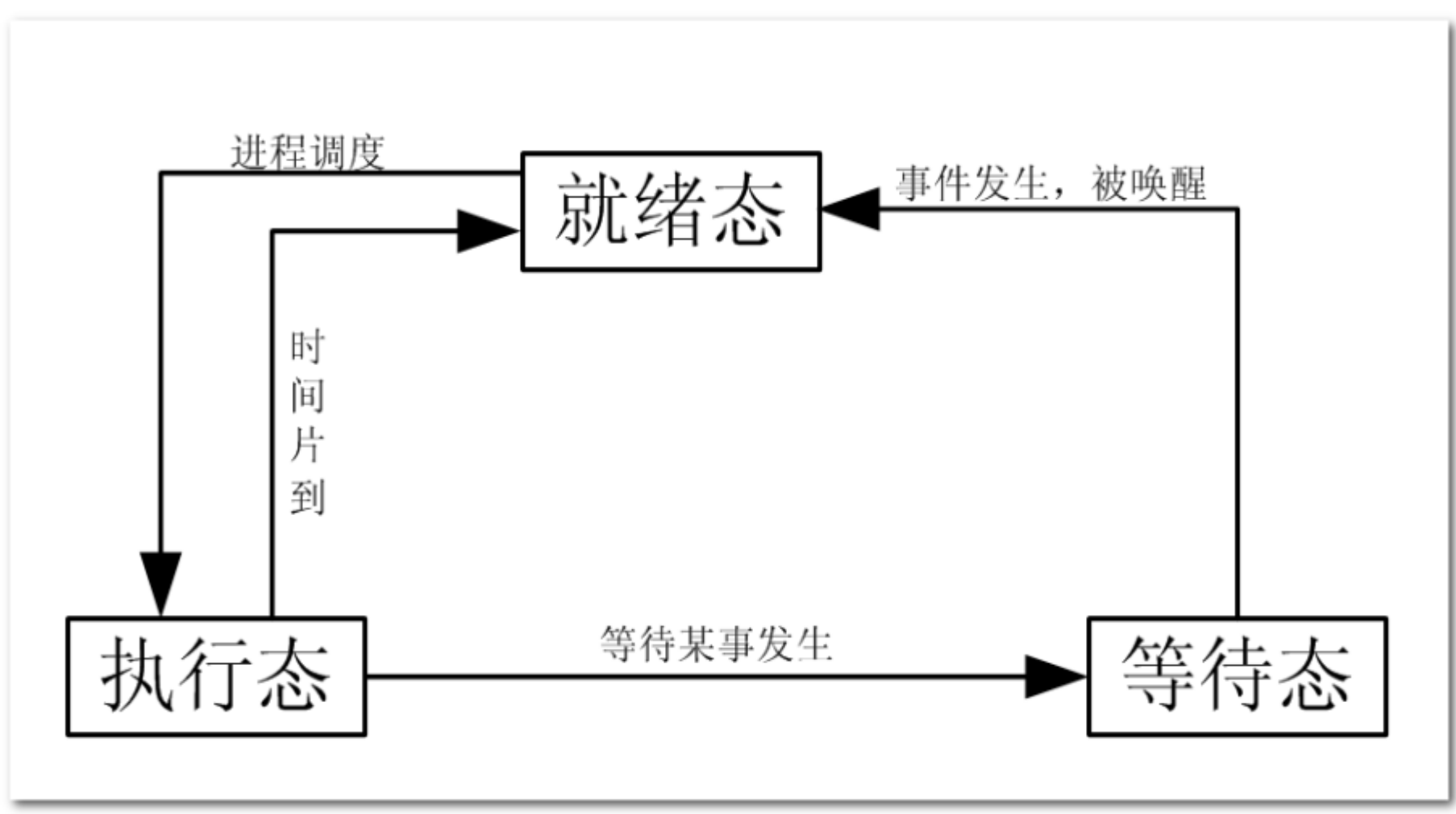
➤ 执行态:

该进程正在占用CPU运行。

➤ 等待态:

进程因不具备某些执行条件而暂时无法继续执行的状态。

# 进程概述



进程三种状态的转换关系



- 进程概述
  - 进程的定义
  - 进程的状态及转换
  - 进程控制块
- 进程控制

# 进程概述

- 进程控制块 (PCB) (Progress Control Block)
  - OS是根据PCB来对并发执行的进程进行控制和管理  
的。系统在创建一个进程的时候会开辟一段内存空间  
存放与此进程相关的PCB数据结构。
  - PCB是操作系统中最重要的记录型数据结构。PCB中  
记录了用于描述进程进展情况及控制进程运行所需  
的全部信息。
  - PCB是进程存在的唯一标志，在Linux中PCB存放在  
task\_struct结构体中。

- 进程概述
- 进程控制
  - 进程号
  - 进程的创建
  - 进程的挂起
  - 进程的等待
  - 进程的终止
  - 进程的替换

# 进程控制

- 每个进程都由一个进程号来标识，其类型为pid\_t，进程号的范围：0 ~ 32767。
- 进程号总是唯一的，但进程号可以重用。当一个进程终止后，其进程号就可以再次使用了。
- 在linux系统中进程号由0开始。

进程号为0及1的进程由内核创建。

进程号为0的进程通常是调度进程，常被称为交换进程(swapper)。进程号为1的进程通常是init进程。

除调度进程外，在linux下面所有的进程都由进程init进程直接或者间接创建。

# 进程控制

- 进程号 (PID)

标识进程的一个非负整型数。

- 父进程号 (PPID)

任何进程 (除 `init` 进程) 都是由另一个进程创建, 该进程称为被创建进程的父进程, 对应的进程号称为父进程号 (PPID)。

- 进程组号 (PGID)

进程组是一个或多个进程的集合。他们之间相互关联, 进程组可以接收同一终端的各种信号, 关联的进程有一个进程组号 (PGID) 。

- Linux操作系统提供了三个获得进程号的函数getpid()、getppid()、getpgid()。

需要包含头文件:

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

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

# 进程控制

- `pid_t getpid(void)`
  - 返回值: 本进程号 (PID)
- `pid_t getppid(void)`
  - 返回值: 调用此函数的进程的父进程号 (PPID)
- `pid_t getpgid(pid_t pid)`
  - 参数: 0当前PGID, 否则为指定进程的PGID
  - 返回值: 进程组号 (PGID)

例: [01\\_pid.c](#)

# 进程控制

- 在linux环境下，创建进程的主要方法是调用以下函数：
  - `#include <sys/types.h>`
  - `#include <unistd.h>`
  - `pid_t fork(void);`



- fork函数: 创建一个新进程

`pid_t fork(void)`

功能:

- `fork()` 函数用于从一个已存在的进程中创建一个新进程, 新进程称为子进程, 原进程称为父进程。

返回值:

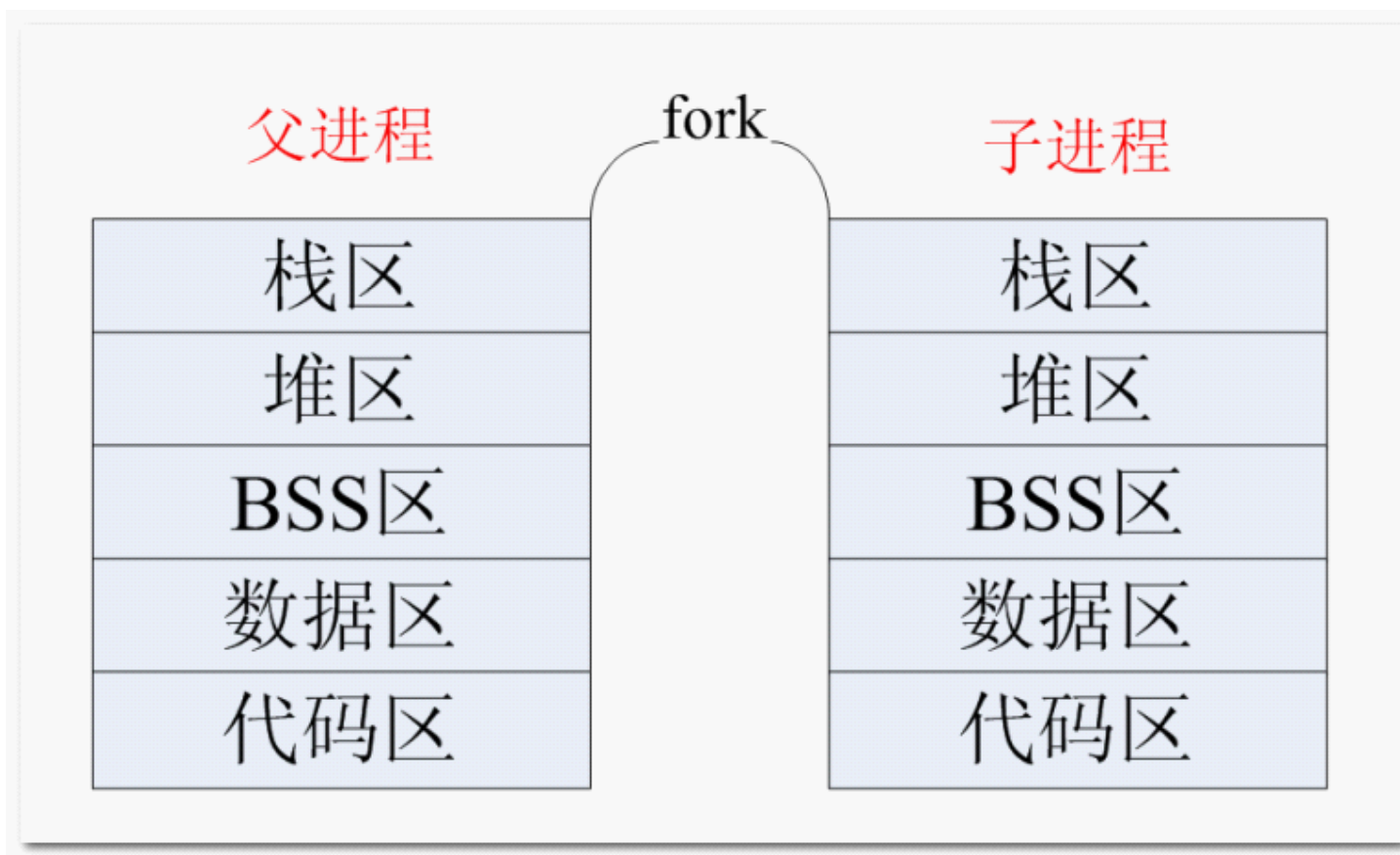
- 成功: 子进程中返回0, 父进程中返回子进程ID。
- 失败: 返回-1。

# 进程控制

- 使用fork函数得到的子进程是父进程的一个复制品，它从父进程处继承了整个进程的地址空间。
- 地址空间：
  - 包括进程上下文、进程堆栈、打开的文件描述符、信号控制设定、进程优先级、进程组号等。
- 子进程所独有的只有它的进程号，计时器等。因此，使用fork函数的代价是很大的。

# 进程控制

➤ fork函数执行结果:(复制模式)



例: [02\\_fork\\_1.c](#)

# 进程控制

例: 02\_fork\_2.c

- 从02\_fork\_2.c程序可以看出, 子进程对变量所做的改变并不影响父进程中该变量的值, 说明父子进程各自拥有自己的地址空间。
- 一般来说, 在fork之后是父进程先执行还是子进程先执行是不确定的。这取决于内核所使用的调度算法。
- 如要求父子进程之间相互同步, 则要求某种形式的进程间通信。

# 进程控制

例: 02\_fork\_3.c

➤ 提示:

➤ 标准I/O提供三种类型的缓冲:

➤ 全缓冲: (大小不定)

在填满标准I/O缓冲区后, 才进行实际的I/O操作。术语冲洗缓冲区的意思是进行标准I/O写操作。

➤ 行缓冲: (大小不定)

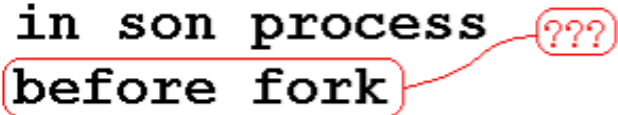
在遇到换行符时, 标准I/O库执行I/O操作。这种情况允许我们一次输入一个字符, 但只有写了一行后才进行实际的I/O操作。进程的等待

➤ 不带缓冲

# 进程控制

## ➤ 运行方法:

```
[root@localhost fork]# gcc 02_fork_3.c -o 02_fork_3
[root@localhost fork]# ./02_fork_3 交互：行缓冲
a write to stdout
before fork
in son process
in father process
[root@localhost fork]# ./02_fork_3 > test 全缓冲
[root@localhost fork]# cat test
a write to stdout
before fork
in son process
before fork
in father process
[root@localhost fork]#
```



# 进程控制

- 调用fork函数后，父进程打开的文件描述符都被复制到子进程中。在重定向父进程的标准输出时，子进程的标准输出也被重定向。
- write函数是系统调用，不带缓冲。
- 标准I/O库是带缓冲的，当以交互方式运行程序时，标准I/O库是行缓冲的，否则它是全缓冲的。

# 进程控制

- 僵尸进程 (Zombie Process)

父进程未运行结束，已运行结束的子进程。

- 孤儿进程 (Orphan Process)

父进程运行结束，但子进程未运行结束的子进程。

- 守护进程 (精灵进程) (Daemon process)

守护进程是个孤儿进程，它提供系统服务，常常在系统启动时启动，仅在系统关闭时才终止。这种进程脱离终端，在后台运行。



# 进程控制

- 进程在一定的时间内没有任何动作，称为进程的挂起

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

```
unsigned int sleep(unsigned int sec);
```

- 功能:

进程挂起指定的秒数，直到指定的时间用完或收到信号才解除挂起。

- 返回值:

若进程挂起到sec指定的时间则返回0，若有信号中断则返回剩余秒数。

- 注意:

进程挂起指定的秒数后程序并不会立即执行，系统只是将此进程切换到就绪态。

# 进程控制

- 父子进程有时需要简单的进程间同步，如父进程等待子进程的结束。
- linux下提供了以下两个等待函数wait ()、waitpid ()。
- 需要包含头文件：
  - #include <sys/types.h>
  - #include <sys/wait.h>

➤ `pid_t wait(int *status);`

功能:

等待子进程改变状态，如果子进程终止了，此函数会回收子进程的资源。

调用`wait`函数的进程会挂起，直到它的一个子进程退出或收到一个不能被忽视的信号时才被唤醒。

若调用进程没有子进程或它的子进程已经结束，该函数立即返回。

## ➤ 参数:

函数返回时，参数`status`中包含子进程退出时的状态信息。子进程的退出信息在一个`int`中包含了多个字段，用宏定义可以取出其中的每个字段。

## ➤ 返回值:

- 如果执行成功则返回子进程的进程号。
- 出错返回-1，失败原因存于`errno`中。

## ➤ 取出子进程的退出信息

### ➤ WIFEXITED(status):

如果子进程是正常终止的，取出的字段值非零。

### ➤ WEXITSTATUS(status):

取出的字段值为子进程调用exit函数返回的值（8~16位）。在用此宏前应先用宏WIFEXITED判断子进程是否正常退出，正常退出才可以使用此宏

例: 04\_wait.c

# 进程控制

➤ `pid_t waitpid(pid_t pid, int *status, int options);`

功能:

等待子进程改变状态, 如果子进程终止了, 此函数会回收子进程的资源。

返回值:

- 如果执行成功则返回子进程ID。
- 出错返回-1, 失败原因存于errno中。

# 进程控制

- 参数pid的值有以下几种类型:
  - pid>0:  
等待进程ID等于pid的子进程。
  - pid=0  
等待同一个进程组中的任何子进程，如果子进程已经加入了别的进程组，waitpid不会等待它。
  - pid=-1:  
等待任一子进程，此时waitpid和wait作用一样。
  - pid<-1:  
等待指定进程组中的任何子进程，这个进程组的ID等于pid的绝对值。

# 进程控制

- `status` 参数中包含子进程退出时的状态信息。
- `options` 参数能进一步控制 `waitpid` 的操作：
  - 0:  
同 `wait`，阻塞父进程，等待子进程退出。
  - `WNOHANG`:  
没有任何已经结束的子进程，则立即返回。
  - `WUNTRACED`  
如果子进程已暂停则马上返回，且子进程的结束状态不予以理会。



## ➤ 返回值:

如果设置了选项WNOHANG，调用waitpid时若没有任何已经结束的子进程可收集，返回0；若有已经结束的子进程可收集，则返回子进程进程号。

出错返回-1（当pid所指示的子进程不存在，或此进程存在，但不是调用进程的子进程，waitpid就会出错返回）；这时errno被设置为ECHILD。

例: [04\\_waitpid.c](#)

- 在linux下可以通过以下方式结束正在运行的进程:
  - `void exit(int value);`
  - `void _exit(int value);`

➤ `exit`函数: 结束进程执行

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

```
void exit(int value)
```

参数:

`status`: 返回给父进程的参数(低8位有效)。

➤ `_exit`函数: 结束进程执行

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

```
void _exit(int value)
```

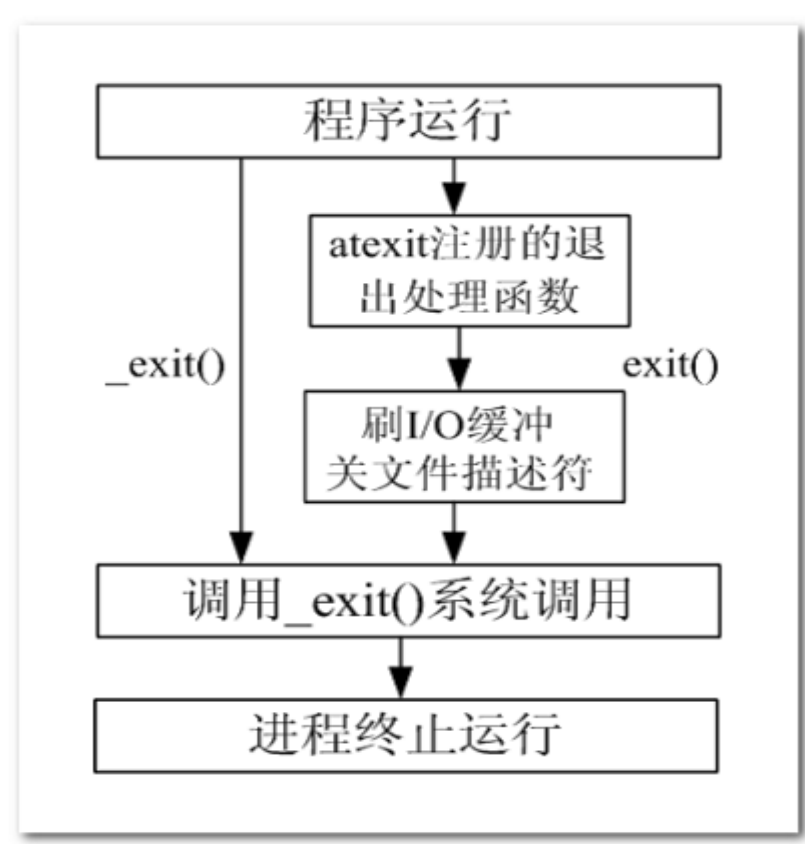
参数:

`status`: 返回给父进程的参数(低8位有效)。

# 进程控制

➤ `exit`和`_exit`函数的区别:

`exit`为库函数, 而`_exit`为系统调用



# 进程控制

- 进程在退出时可以用atexit函数注册退出处理函数。
- `#include <stdlib.h>`
- `int atexit(void (*function)(void));`

功能:

注册进程正常结束前调用的函数。

参数:

function: 进程结束前, 调用函数的入口地址。

- 一个进程中可以多次调用atexit函数注册清理函数, 正常结束前调用函数的顺序和注册时的顺序相反。

例: [05\\_atexit.c](#)

