

# 第 4 章 进程管理

月出皓兮 苏曙光老师的课堂笔记

2022 年 3 月 19 日

## 目录

<b>1 进程 (process) 概念</b>	<b>3</b>
1.1 进程的定义	3
1.2 进程的特征	3
1.3 进程的类型	3
1.3.1 按使用资源的权限	3
1.3.2 按对 CPU 的依赖性	3
1.4 进程的状态	3
1.4.1 进程的三态模型 Running, Ready, Block	3
1.4.2 进程的五态模型 New, Exit	4
1.4.3 进程的七态模型 Suspend, Resume	4
1.4.4 进程状态的变迁	4
1.4.5 Linux 进程的状态	4
1.4.6 Linux 显示 process 状态	6
<b>2 进程控制块 (Process Control Block, PCB)</b>	<b>6</b>
2.1 PCB 的定义	6
2.2 PCB 中的基本成员	6
2.3 Linux 的进程控制块 PCB: task_struct	6
2.4 使用 SOURCE INSIGHT 阅读 Linux2.6 内核源码	7
2.4.1 进程家族关系相关的成员变量	8
2.4.2 和内存相关的成员变量	8
2.4.3 和文件相关的成员变量	8

2.4.4	和进程标识相关的成员变量	8
2.5	使用 SOURCE INSIGHT 阅读 Linux0.11 内核源码	9
2.6	进程的上下文（约等于 PCB）	9
2.7	分时系统的进程切换过程	9
<b>3</b>	<b>进程控制</b>	<b>10</b>
3.1	创建进程	10
3.1.1	参数	10
3.1.2	过程	10
3.2	撤消进程	10
3.2.1	参数	10
3.2.2	过程	10
3.3	阻塞进程	11
3.3.1	引起阻塞的时机/事件	11
3.3.2	参数	11
3.3.3	过程	11
3.4	唤醒进程	11
3.4.1	引起唤醒的时机/事件	11
3.4.2	参数	11
3.5	进程控制原语	12
3.6	Windows 进程控制	12
3.6.1	WINDOWS 通过编程启动一个程序	12
3.6.2	CreateProcess	12

# 1 进程 (process) 概念

## 1.1 进程的定义

进程是程序在某个**数据集合**上的一次**运行活动**。

数据集合：软/硬件环境，多个进程共存/共享的环境

## 1.2 进程的特征

1. 动态性：进程是程序的一次执行过程，动态产生/消亡
2. 并发性：进程可以同其他进程一起向前推进
3. 异步性：进程按各自速度向前推进（必要的时候需要进行同步）
4. 独立性：进程是系统分配资源和调度 CPU 的单位；

## 1.3 进程的类型

### 1.3.1 按使用资源的权限

系统进程：指系统内核相关的进程。

用户进程：运行于用户态的进程。

### 1.3.2 按对 CPU 的依赖性

偏 CPU 进程：计算型进程

偏 I/O 进程：侧重于 I/O 的进程

## 1.4 进程的状态

### 1.4.1 进程的三态模型 Running, Ready, Block

运行状态 (Running)：进程已经占有 CPU，在 CPU 上运行。

就绪状态 (Ready)：具备运行条件但由于无 CPU，暂时不能运行。

阻塞状态 (Block) (等待状态 (Wait))：因为等待某项**服务完成**或**信号来到**而不能运行的状态，例如等待：系统调用，I/O 操作，合作进程的服务或信号。

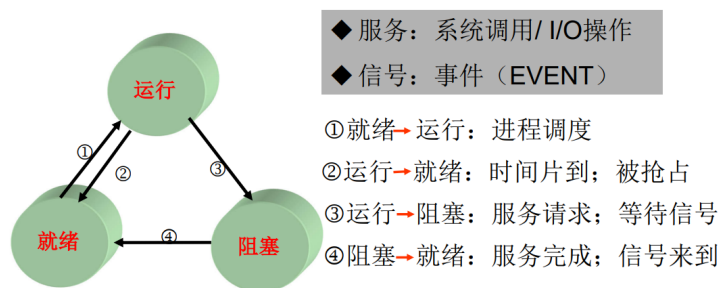


图 1: 三态模型的变化

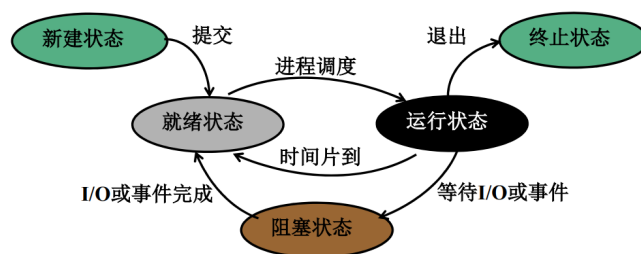


图 2: 具有新建 (new) 和终止 (terminate) 状态的进程状态

#### 1.4.2 进程的五态模型 New, Exit

新建状态 (New)：对应于进程被创建时的状态，尚未进入就绪队列。

终止状态 (Exit)：处于终止态的进程不再被调度执行，下一步将被系统撤销，最终从系统中消失。

#### 1.4.3 进程的七态模型 Suspend, Resume

挂起与解挂 Suspend, Resume

#### 1.4.4 进程状态的变迁

进程的状态可以依据一定的条件相互转化。

#### 1.4.5 Linux 进程的状态

可运行态 (TASK\_RUNNING)：就绪与运行。

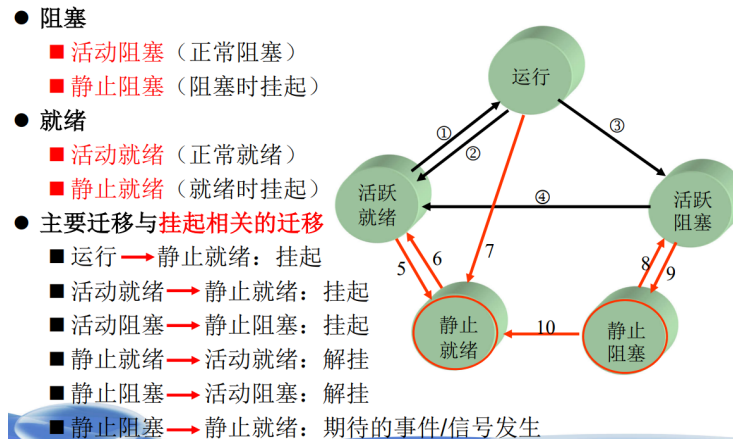


图 3: 支持挂起 (suspend) 和解挂 (resume) 操作的进程状态

睡眠态/阻塞态/等待态:

深度睡眠 (TASK\_UNINTERRUPTIBLE) 不能被其他进程通过信号和时钟中断唤醒。

浅度睡眠 (TASK\_INTERRUPTIBLE) 可被其他进程的信号或时钟中断唤醒。

僵死态 (TASK\_ZOMBIE): 进程终止执行, 释放大部分资源。

挂起态 (TASK\_STOPPED): 进程被挂起。

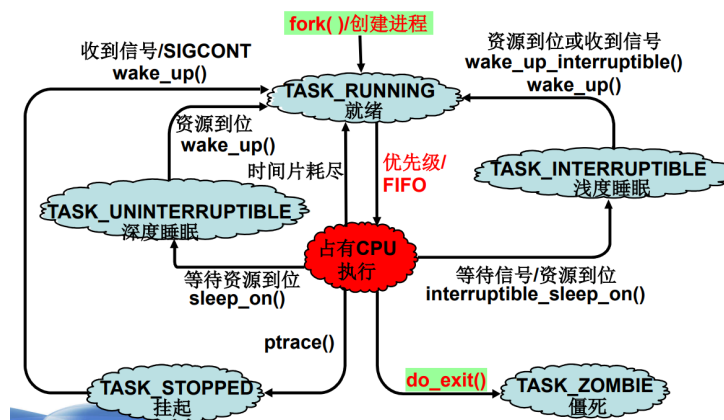


图 4: Linux 进程状态的转换

### 1.4.6 Linux 显示 process 状态

ps 命令。ps aux

## 2 进程控制块 (Process Control Block, PCB)

### 2.1 PCB 的定义

描述进程的状态、资源、和相关进程的关系的一种数据结构。

PCB 是进程的标志：创建进程时创建 PCB；进程撤销后 PCB 同时撤销。

进程 = 程序 + PCB。每当程序运行一次，创建一次 PCB，即进程运行一次。

### 2.2 PCB 中的基本成员

1. name (ID): 进程名称 (标识符)
2. status: 状态
3. next: 指向下一个 PCB 的指针
4. start\_addr: 程序地址
5. priority: 优先级
6. cpu\_status: 现场保留区 (堆栈)
7. comm\_info: 进程通信机制
8. process\_family: 家族
9. own\_resource: 资源清单

### 2.3 Linux 的进程控制块 PCB: task\_struct

基本内容包括

1. 进程状态

2. 调度信息
3. 标识符
4. 内部进程通信信息
5. 链接信息
6. 时间和计时器
7. 文件系统

## 8. 虚拟内存信息

## 9. 处理器信息

## 2.4 使用 SOURCE INSIGHT 阅读 Linux2.6 内核源码

```

struct task_struct {
    volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    void *stack;
    atomic_t usage; /* 有几个进程正在使用该结构 */
    unsigned int flags;
    unsigned int ptrace;

    int lock_depth; /* BKL lock depth */

    int oncpu; /* 在哪个CPU上运行 */

    int prio, static_prio, normal_prio; /* 静态优先级, 动态优先级 */
    struct list_head run_list;
    const struct sched_class *sched_class; /* 与调度相关的函数 */
    struct sched_entity se; /* 调度实体 */

    unsigned int policy; /* 调度策略 */
    cpumask_t cpus_allowed;
    unsigned int time_slice;
    struct sched_info sched_info; /* 调度相关的信息, 如在CPU上运行的时间/在队列中等待的时间等。 */

    struct list_head tasks; /* 任务队列 */
    struct list_head ptrace_children;
    struct list_head ptrace_list;

    struct mm_struct *mm, *active_mm;
    /* mm是进程的内存管理信息, active_mm指向结构进程当前活动的地址空间 */

    /* task state */
    struct linux_binfmt *binfmt;
    int exit_state; /* 进程退出时的状态 */
    int exit_code, exit_signal; /* 进程退出时发出的信号 */
    unsigned int personality;
    unsigned did_exec:1;
    pid_t pid; /* 进程ID */
    pid_t tgid; /* 进程组ID */
    /* 进程家族关系相关的成员变量 */
    struct task_struct *real_parent; /* real parent process (when being debugged) */
    struct task_struct *parent; /* parent process */
    struct list_head children; /* list of my children */
    struct list_head sibling; /* linkage in my parent's children list */
    struct task_struct *group_leader; /* threadgroup leader */

    /* PID/PID hash table linkage. */
    struct pid_link pids[PIDTYPE_MAX];
    struct list_head thread_group;

    unsigned int rt_priority;
    cputime_t utime, stime, utimescaled, stimescaled;
    cputime_t gtime;
    cputime_t prev_utime, prev_stime;

    /* process credentials */
    uid_t uid, euid, suid, fsuid;
    gid_t gid, egid, sgid, fsgid;
    struct group_info *group_info;

    /* file system info */
    int link_count, total_link_count;
    struct sysv_sem sysvsem;
    /* CPU-specific state of this task */
    struct thread_struct thread;
    /* filesystem information */
    struct fs_struct *fs; /* 文件系统信息 */
    /* open file information */
    struct files_struct *files; /* 使用的文件 */
    /* namespaces */
    struct nsproxy *nsproxy;

    /* signal handlers */
    /* 信号通信机制 */
    struct signal_struct *signal; /* pending sigs */
    struct sighand_struct *sighand;

    sigset_t blocked, real_blocked; /* max=sked sigs */
    sigset_t saved_sigmask;
    struct sigpending pending;

    tty_struct *tty; /* 终端 */

```

此处仅展示了部分信息。

#### 2.4.1 进程家族关系相关的成员变量

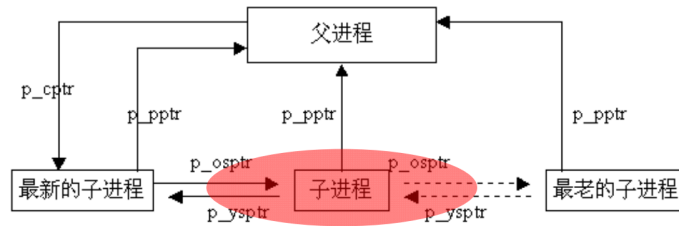


图 5: 进程家族关系

#### 2.4.2 和内存相关的成员变量

`p->mm` 指向 `mm_struct` 结构进程的地址空间。

`p->active_mm` 指向 `mm_struct` 结构进程的当前活动地址空间。

#### 2.4.3 和文件相关的成员变量

`p->fs`, 文件系统信息: `root` 目录和挂载点; 当前工作目录和挂载点。

`p->files` 字段包含了文件句柄表

#### 2.4.4 和进程标识相关的成员变量

LINUX 进程的标识:

PID: 进程 ID, 每个进程有唯一编号: PID

PPID: 父进程 ID

PGID: 进程组 ID

UNIX 进程的用户标识:

UID: 用户 ID

GID: 用户组 ID

除 `init` 进程外, 每个进程都可用 `kill` 命令杀死。



## 2.5 使用 SOURCE INSIGHT 阅读 Linux0.11 内核源码

当读者使用盗版 SOURCE INSIGHT 时，注意断网使用。

```

struct task_struct {
/* these are hardcoded - don't touch */
    long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    long counter; /* 信号通信相关 */
    long priority;
    long signal; /* 信号通信相关 */
    struct sigaction sigaction[32]; /* 信号安装函数 */
    long blocked; /* bitmap of masked signals */
/* various fields */
    int exit_code;
    unsigned long start_code, end_code, end_data, brk, start_stack;
    long pid, father, pgrp, session, leader;
    unsigned short uid, euid, suid; /* ID */
    unsigned short gid, egid, sgid; /* ID */
    long alarm;
    long utime, stime, cutime, cstime, start_time;
    unsigned short used_math;
/* file system info */
    int tty; /* -1 if no tty, so it must be signed */ // 终端
    unsigned short umask;
    struct m_inode * pwd; /* 与文件目录相关 */
    struct m_inode * root; /* 与文件目录相关 */
    struct m_inode * executable; /* 与文件目录相关 */
    unsigned long close_on_exec;
    struct file * filp[NR_OPEN]; /* 打开的文件列表 */
/* ldt for this task 0 - zero 1 - cs 2 - ds&ss */
    struct desc_struct ldt[3]; /* 保护模式下，当前状态代码运行的空间 */
/* tss for this task */
    struct tss_struct tss; /* 上下文 */
} /* end task_struct */

```

## 2.6 进程的上下文（约等于 PCB）

Context，进程运行环境，约等于 PCB。

## 2.7 分时系统的进程切换过程

进程的上下文在 CPU 中交换

换出进程的上下文离开 CPU（到栈 +PCB 上去）

换入进程的上下文进入 CPU（从栈 +PCB 上来）

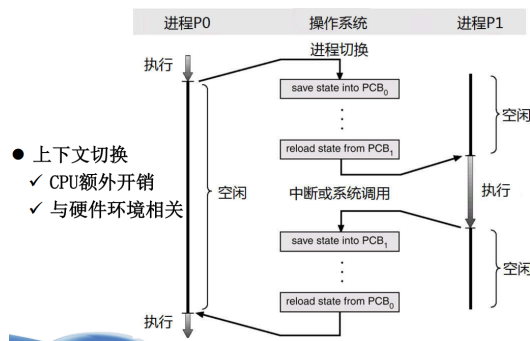


图 6: 进程切原的过程

## 3 进程控制

在进程生存全期间，对其全部行为的控制。主要包括创建进程，撤消进程，阻塞进程，唤醒进程。

### 3.1 创建进程

创建一个具有指定标识（ID）的进程。

#### 3.1.1 参数

进程标识、优先级、进程起始地址、CPU 初始状态、资源清单等。

#### 3.1.2 过程

1. 创建一个空白 PCB
2. 赋予进程标识符 ID
3. 为进程分配空间
4. 初始化 PCB 的 CPU 状态，内存，优先级，进程状态等。
5. 插入相应的进程队列（**就绪队列**）

### 3.2 撤消进程

撤消一个指定的进程，收回进程所占有的资源，撤消该进程的 PCB。  
大概率出现于正常结束，异常结束，外界干预这三种情况。

#### 3.2.1 参数

被撤消的进程名（ID）

#### 3.2.2 过程

1. 在 PCB 队列中检索出该 PCB
2. 获取该进程的状态。
3. 若该进程处在运行态，立即终止该进程。是否需要撤销其子进程？若是，则递归地撤销其子进程。若不是，将子进程挂接到 init 进程下。
4. 释放进程占有的资源
5. 将进程从 PCB 队列中移除

### 3.3 阻塞进程

停止进程执行，变为阻塞。

#### 3.3.1 引起阻塞的时机/事件

请求系统服务（由于某种原因，OS 不能立即满足进程的要求）  
启动某种操作（进程启动某操作，阻塞等待该操作完成）  
新数据尚未到达（A 进程要获得 B 进程的中间结果，A 进程等待）  
无新工作可作（进入 idle 进程/pause( )，等待新任务到达）

#### 3.3.2 参数

阻塞原因  
不同原因构建有不同的阻塞队列。

#### 3.3.3 过程

停止运行  
将 PCB “运行态”改“阻塞态”  
插入对应的阻塞队列  
转调度程序

### 3.4 唤醒进程

唤醒处于阻塞队列当中的某个进程。

#### 3.4.1 引起唤醒的时机/事件

系统服务由不满足到满足  
I/O 完成  
新数据到达  
进程提出新请求（服务）

#### 3.4.2 参数

被唤醒进程的标识

### 3.5 进程控制原语

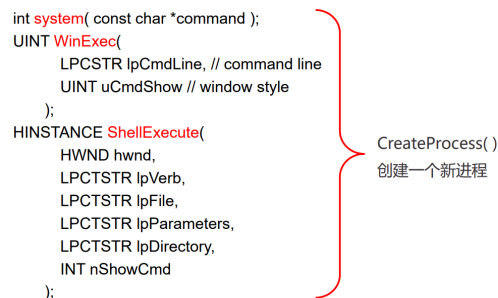
由若干指令构成的具有特定功能的函数，具有原子性，其操作不可分割。

进程控制，要不然全部完成，要不然就不完成。

创建原语，撤消原语，阻塞原语，唤醒原语。

### 3.6 Windows 进程控制

#### 3.6.1 WINDOWS 通过编程启动一个程序



```

int system( const char *command );
UINT WinExec(
    LPCSTR lpCmdLine, // command line
    UINT uCmdShow // window style
);
HINSTANCE ShellExecute(
    HWND hwnd,
    LPCTSTR lpVerb,
    LPCTSTR lpFile,
    LPCTSTR lpParameters,
    LPCTSTR lpDirectory,
    INT nShowCmd
);

```

CreateProcess()  
创建一个新进程

图 7: WINDOWS 通过编程启动一个程序

应注意的是，CreateProcess 只能创建 32 位进程，在 Win10 中运行存在兼容性问题。

#### 3.6.2 CreateProcess

```

BOOL CreateProcess(
    LPCTSTR lpApplicationName, // 可执行程序名
    LPTSTR lpCommandLine, //[可执行程序名] 程序参数，例如打开的
    文件等
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags, //创建标志
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,

```

---

```
LPSTARTUPINFO lpStartupInfo,  
LPPROCESS_INFORMATION lpProcessInformation  
); // lpProcessInformation : 接收新进程的识别信息  
创建进程内核对象，创建虚拟地址空间  
装载 EXE 和/或 DLL 的代码和数据到地址空间中  
创建主线程和线程内核对象  
启动主线程，进入主函数 (main)
```