课程目标：掌握linux下并发程序设计

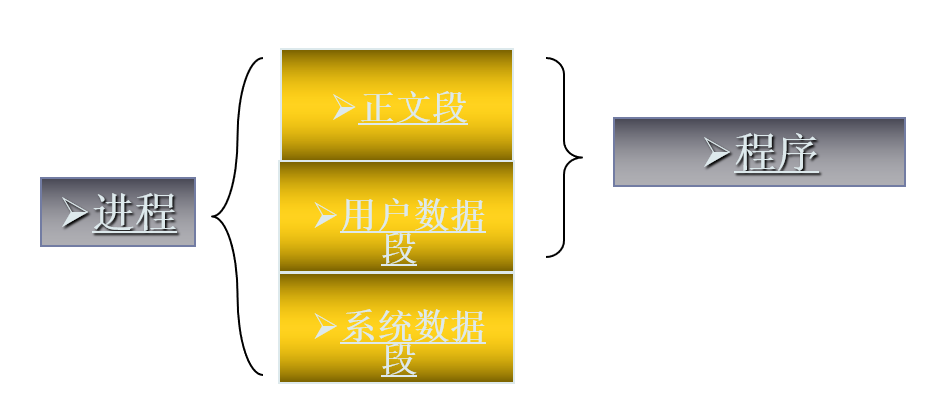
**一、进程基础**

**1.1 基本概念**

**1.1.1 什么是进程**

在 Linux 系统中，​​进程是程序的一次动态执行过程。

程序是静态的代码文件（如 /bin/ls），而进程是程序运行时的实例，包含程序运行所需的所有资源（如内存、文件描述符、寄存器状态等）



**1.1.2 PID 与PPID**

PID（Process ID）​ 是操作系统为每个进程分配的​​唯一数字标识符​​，用于在系统中唯一标识一个进程。它是进程的“身份证号”，贯穿进程的整个生命周期（从创建到终止）。

PPID（Parent Process ID) 是​​创建当前进程的父进程​​id。每个进程（除系统初始进程外）都有且仅有一个父进程。pstree 可直观展示进程的父子层级。

**1.1.3 进程类型**

Linux 进程按​​用户交互方式​​和​​运行目标​​可分为三大类：交互进程、批处理进程、守护进程。

交互进程

交互进程是​​直接与用户终端交互​​的进程，用户通过键盘输入指令，进程实时输出结果。

批处理进程

批处理进程是​​非交互式​​的自动化任务，通常按计划或批量处理数据，无需用户实时干预。

守护进程

守护进程是​​长期运行在后台​​的系统服务进程，不与任何终端直接关联。

**1.1.4 进程状态**

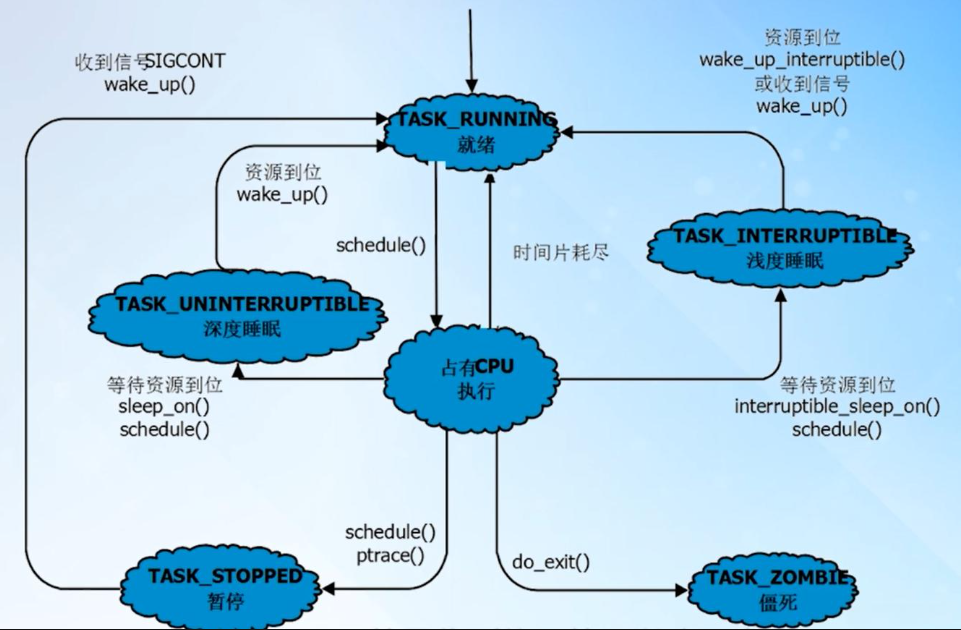
Linux 进程的状态由内核维护，可通过 ps 或 top 命令查看。常见状态如下

|  |  |  |  |
| --- | --- | --- | --- |
| 状态 | **状态符** | **全称** | **含义** |
| 运行与就绪 | R | Running/Runnable | 运行中（正在 CPU 执行）或就绪（等待 CPU 分配时间片）。 |
| 等待态 | S | Sleeping | 可中断睡眠（等待事件，如用户输入、信号；可被信号唤醒）。 |
| D | Disk Sleep | 不可中断睡眠（深度睡眠，通常因 I/O 操作阻塞；无法被信号唤醒）。 |
| 停止态 | T | Stopped | 暂停（被SIGSTOP、SIGTSTP等信号暂停，如Ctrl+Z终止前台进程）。 |
| 僵尸态 | Z | Zombie | 僵尸态 |

状态修饰符

|  |  |
| --- | --- |
| < | 高优先级的 |
| N | 低优先级的 |
| s | 包含子进程 |
| l | 多线程 |
| + | 前台运行 |

进程状态切换图



**1.2 进程相关命令**

ps 查看系统中的进程，支持多种选项组合

// 示例：

ps -ef

ps aux

ps -el

top 动态显示系统中的进程

nice 按用户指定的优先级运行进程

nice -n priority cmd

//示例：

nice -n 10 ./a.out //指定优先级10

sudo nice -n -10 ./a.out

//注意：Linux 进程的调度优先级由 nice 值决定（范围：-20 到 19），值越小优先级越高（默认值为 0）

renice 改变正在运行进程的优先级

renice -n priority pid

//示例：

sudo renice -n 10 pid //改变进程的优先级

kill 向进程发信号

//示例：

kill -l //产看信号列表

kill pid //结束进程

kill -14 pid //发送14号信号 给指定进程

kill -19 pid //挂起该进程

kill -18 pid //继续运行

kill %1 //结束作业号为1的进程

jobs 管理终端下所有作业

//示例：

jobs //列出所有作业的状态

jobs -l //显示详细信息

bg 将挂起的进程在后台执行

bg [作业号]

//示例：

bg 1 //将作业 1 放到后台继续运行

fg 把后台运行的进程放到前台运行

//示例：

fg 1 //将作业 1 恢复到前台

**1.3 重要概念**

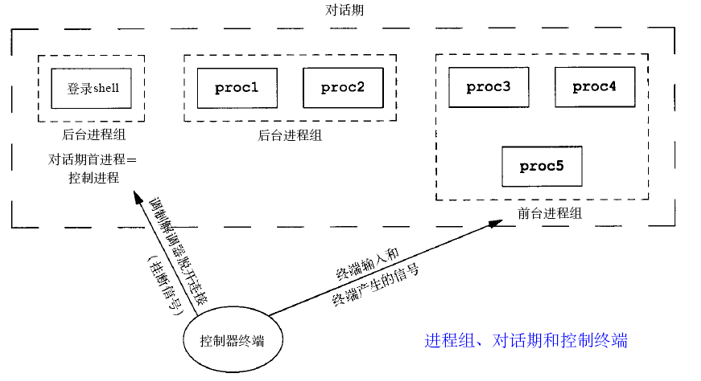
孤儿进程： 父进程 先于子进程退出

僵尸进程：子进程 先于父进程退出，没有回收资源

进程：相对于线程，独立性更好

进程组：是​​一组逻辑上相关的进程的集合​​，每个进程组有一个唯一的 ​​进程组 ID（PGID）

会话：一个或多个进程组的集合，用于管理进程与终端的关联。



PCB（Process Control Block）：进程控制块，task\_struct结构体，是操作系统管理进程的“神经中枢”，它通过集中存储进程的状态、资源和上下文信息，支撑了进程的创建、调度、同步和终止等核心功能。

写时拷贝：fork() 是用于创建子进程的系统调用。传统上，创建子进程需要​​完全复制父进程的内存空间​​，但这会导致显著的性能开销（尤其是对大内存进程）。为了优化这一过程，现代操作系统普遍采用​​写时拷贝（Copy-On-Write, COW）​​技术，仅在需要修改内存时才实际复制数据。

**二、进程相关函数**

**2.1 基本函数**

**1. fork：创建子进程**

|  |
| --- |
| 函数原型 |
| pid\_t fork(void); |

它通过​​复制父进程​​的方式生成一个几乎完全相同的子进程，实现进程的“分叉”。

一次调用，两次返回，通过返回值区分子父进程。

**2. exit/\_exit：结束进程**

|  |  |  |
| --- | --- | --- |
| 函数原型 | 类型 |  |
| void exit(int status); | 库函数 | 清理缓存 |
| void \_exit(int status); | 系统调用 | 不清理 |

进程完成任务或需要退出时，需调用终止进程。exit 和 \_exit 都能终止进程，差异在是否清理缓存。

**3. wait/waitpid：回收子进程资源**

创建子进程，子进程先于父进程

|  |
| --- |
| 函数原型 |
| pid\_t wait(int \*wstatus); |
| pid\_t waitpid(pid\_t pid, int \*wstatus, int options); |

父进程创建子进程后，需通过 wait 或 waitpid 回收子进程的资源（如 PID、退出状态），否则子进程会成为​​僵尸进程​​（Zombie Process，仅保留 PCB，占用系统资源）。

父进程通过 wait/waitpid 获取子进程的 status，可以通过以下宏进行解析。

WIFEXITED(status)： 是否正常退出

WEXITSTATUS(status)：正常退出时的返回值（即 exit 或 main 的 return 值）。

WIFSIGNALED(status)：是否因信号终止（如 kill 命令）。

WTERMSIG(status)：导致终止的信号编号。

**4. exec 函数族：启动程序（替换原进程）**

exec 函数族的作用是​​替换当前进程空间​​，使其执行新的程序（原进程的代码、数据被覆盖，但 PID 不变）。

执行成功不会返回，失败返回-1。

exec 家族有多个版本(库函数6个，系统调用1个)，核心区别是参数传递方式。

|  |  |
| --- | --- |
| 函数原型 | 参数说明 |
| int execl(const char \*pathname, const char \*arg, ...); | pathname：路径名  arg：程序名  ...：参数，NULL结束 |
| int execlp(const char \*file, const char *arg, ..*); | file：路径名  arg：程序名  ...：参数，NULL结束 |
| int execle(const char \*pathname, const char \*arg, ..., char \*const envp[]); | pathname：路径名  arg：程序名  envp: {环境变量，NULL结束 } |
| int execv(const char \*pathname, char \*const argv[]); | pathname：路径名  argv：{程序名，参数，NULL结束} |
| int execvp(const char \*file, char \*const argv[]); | file：路径名  argv：{程序名，参数，NULL结束} |
| int execvpe(const char \*file, char \*const argv[],char \*const envp[]); | file：路径名  argv：{程序名，参数，NULL结束}  envp: {环境变量，NULL结束 } |

差异说明

|  |  |
| --- | --- |
| l | 参数逐个罗列 |
| p | 包含环境变量PATH(不需要填写完整路径) |
| v | 采用数组传参 |
| e | 可以指定环境变量，通过数组传入 |

**2.2 守护进程相关函数**

在 Linux 系统中，​​守护进程（Daemon）​​ 是一种在后台长期运行的特殊进程，通常不与任何终端直接关联，负责提供系统级服务或完成特定任务（如网络服务、日志管理、定时任务等）。

特征：

脱离控制终端（TTY）

会话组领导者（Session Leader）

无关联终端，I/O 重定向到空设备或文件

通常以 root 权限运行

父进程为 init (PID=1)

创建守护进程，往往需要用到以下函数

setsid();    //创建新会话

chdir();   //改变工作目录，避免守护进程因原工作目录所在的文件系统被卸载而无法运行

umask();     //设置权限掩码，避免继承父进程的文件权限掩码，确保守护进程创建文件时的权限可控。

示例代码

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <time.h>

#include <string.h>

int main(void)

{

    pid\_t pid;

    // 1.创建子进程，父进程退出

    pid = fork();

    if (pid == -1)

    {

        perror("fork");

        exit(-1);

    }

    else if (pid > 0)

    {

        exit(-1);

    }

    setsid();    // 2.创建新会话

    chdir("/");  // 3.改变工作目录

    umask(0002); // 4.设置权限掩码

    // 5.关闭没有使用的文件

    int fd = STDERR\_FILENO;

    for (int i = 0; i <= fd; i++)

    {

        close(fd);

    }

    // 采用守护进程 将时间写入文件

    fd = open("log.txt",O\_WRONLY|O\_CREAT,0666);

    if(fd==-1){

        exit(-1);

    }

    time\_t time\_val;

    char buf[256];

    while(1){

        time\_val = time(NULL);

        snprintf(buf,sizeof(buf),"time is %s",ctime(&time\_val));

        write(fd,buf,strlen(buf));

        sleep(1);

    }

    return 0;

}

**三、函数备注**

pid\_t fork(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 创建子进程

\* Input:

\* Return: 失败返回-1，子进程返回0，父进程返回子进程id;

\* Others: 成功返回两次

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int execl(const char \*pathname, const char \*arg, .../\* (char \*) NULL \*/);

int execlp(const char \*file, const char \*arg, .../\* (char \*) NULL \*/);

int execle(const char \*pathname, const char \*arg, .../\*, (char \*) NULL, char \*const envp[] \*/);

int execv(const char \*pathname, char \*const argv[]);

int execvp(const char \*file, char \*const argv[]);

int execvpe(const char \*file, char \*const argv[],char \*const envp[]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： exec函数族，有多个版本，功能类似：启动程序(替换进程)

l:列出参数

v:将参数存入数组中

p : 带PATH变量(不用填写完整的路径)

e : 携带环境变量(可以指定环境变量)

\* Return: 注意：成功不会返回，失败返回-1

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int system(const char \*command);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 执行命令

\* Input: command：命令

\* Return: 失败返回-1 ，成功返回 command执行状态

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void exit(int status);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 结束进程（清空IO缓存）

\* Input: status：进程结束状态

\* Return:

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \_exit(int status);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 结束进程

\* Input: status：进程结束状态

\* Return:

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

pid\_t wait(int \*status);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 回收子进程资源（阻塞）

\* Input:

status：保存子进程状态的地址, NULL:不保存

\* Return: 成功返回进程id; 失败返回-1

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

pid\_t waitpid(pid\_t pid, int \*status, int options);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description： 回收子进程资源（阻塞 or 非阻塞）

\* Input: pid=-1 ：等待任意子进程

pid=>0 ：等待与pid相等的子进程

pid==0 ：等待组ID等于调用进程组ID的任一进程。

pid<-1 ：等待组ID等于pid绝对值的任一进程。

status：保存子进程状态的地址, NULL:不保存

options： 0：阻塞版本 WNOHANG:非阻塞版本

\* Return: 成功返回进程id;

阻塞版本： 失败返回-1

非阻塞版本： 失败返回-1，未成功回收到资源 0

\* Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WIFEXITED(status) // 正常终止子进程 return true

WEXITSTATUS(status) // 获取正常退出的返回值

WIFSIGNALED(status) // 异常退出（被信号结束）return true

WTERMSIG(status) // 获取异常退出 signal num

**作业**

1.实现一个简单的shell

shell的工作原理：当用输入ls -l时，shell 进程会创建子进程，通过子进程通过exec执行命令。