**setitimer 定时器种类及信号**

1. **ITIMER\_REAL**：实际时间，信号SIGALRM
2. **ITIMER\_VIRTUAL**：用户空间时间，信号SIGVTALRM
3. **ITIMER\_PROF**：用户 + 内核时间，信号SIGPROF

**给进程组发信号 & 带数据信号**

* **进程组发信号**：kill(-pgid, sig)，负 pgid 表示进程组
* **带数据信号**：实时信号（34-64）+ sigqueue(pid, sig, val)

**可靠 / 不可靠信号**

* **不可靠信号（1-31）**：不排队，可能丢失（如SIGINT）
* **可靠信号（34-64）**：排队，不丢失（如SIGRTMIN）

**超级块（superblock）& inode**

* **超级块**：文件系统类型、块大小、inode 总数、状态
* **inode**：文件权限、所有者、类型、时间戳、数据块指针

**进程空间组成**

* **用户空间**：代码段、数据段、BSS 段、堆、栈、共享库
* **内核空间**：内核代码、数据、页表、内核栈
* **空间大小（32 位）**：用户空间 3GB，内核空间 1GB

**Linux 文件系统结构**

1. **VFS 层**：统一文件操作接口
2. **具体文件系统层**：ext4/FAT 等，管理 superblock 和 inode
3. **块设备层**：与物理存储交互

**匿名管道 & 命名管道**

* **匿名管道**：无名字，亲缘进程间通信，pipe()创建
* **命名管道**：有文件路径，任意进程通信，mkfifo()创建

**有效用户 / 实际用户 & SUID/SGID**

* **实际用户**：启动进程的用户；**有效用户**：进程运行时权限用户
* **SUID 设置**：chmod u+s 文件名，执行时有效用户变文件所有者
* **SGID 设置**：chmod g+s 文件名，执行时有效组变文件所属组

**superblock/inode/dentry/file**

* **superblock**：文件系统全局信息
* **inode**：文件元数据（权限、数据位置）
* **dentry**：目录项，文件名到 inode 映射
* **file**：进程打开的文件，包含偏移量和操作指针

**获取当前时间（C 语言）**

time\_t now = time(NULL);

struct tm \*t = localtime(&now);

int year = t->tm\_year + 1900;

int month = t->tm\_mon + 1;

int day = t->tm\_mday;

**加载线程库和数学库**：

-lm链接数学库，-lpthread链接线程库

# 链接数学库（libm.so）

gcc main.c -o main -lm

# 链接线程库（pthread）

gcc main.c -o main -lpthread

**C 语言 main 函数参数**

int main(int argc, char \*argv[]);

* **argc**：命令行参数个数（包括程序名本身）
* **argv**：字符串数组，存储参数内容
  + argv[0]：程序名
  + argv[1]~argv[argc-1]：实际参数

**字符转小写**：

char a = 'A';

a = tolower(a); // 需要头文件 <ctype.h>

**求 160 度的正弦值**：

#include <math.h>

double angle = 160.0;

double rad = angle \* M\_PI / 180.0; // 角度转弧度

double sin\_val = sin(rad); // 计算正弦值

|  |  |  |  |
| --- | --- | --- | --- |
| **函数** | **作用** | **参数 & 返回值** | **系统步骤** |
| fork() | 创建子进程 | **参数**：无 **返回值**：子进程返回 0，父进程返回子进程 PID，失败返回 - 1 | 1. 复制父进程内存空间 2. 分配新 PID 3. 父 / 子进程从 fork 后继续执行 |
| exec()族 | 执行新程序 | **参数**：程序路径、参数列表 **返回值**：不返回（成功时），失败返回 - 1 | 1. 加载新程序到当前进程内存 2. 覆盖原程序代码和数据 3. 从新程序 main 开始执行 |
| wait() | 等待子进程结束 | **参数**：int \*status（保存子进程退出状态） **返回值**：子进程 PID，失败返回 - 1 | 1. 阻塞父进程直到子进程结束 2. 回收子进程资源 3. 通过 status 获取退出状态 |
| exit(int status) | 终止当前进程 | **参数**：退出状态码（0 表示正常退出） **返回值**：不返回 | 1. 执行清理函数（atexit 注册的） 2. 关闭所有打开的文件描述符 3. 向父进程发送 SIGCHLD 信号 4. 释放进程资源 |
| getpid() | 获取当前进程 PID | **参数**：无 **返回值**：当前进程 PID |  |

**kill 与 sigqueue 的异同**

|  |  |  |
| --- | --- | --- |
| **特性** | **kill** | **sigqueue** |
| **发送信号** | 支持所有信号 | 仅支持实时信号（34-64） |
| **信号处理** | 简单发送信号，不保证排队 | 信号排队，不丢失 |

**kmalloc 与 vmalloc**

kmalloc分配**连续**物理内存

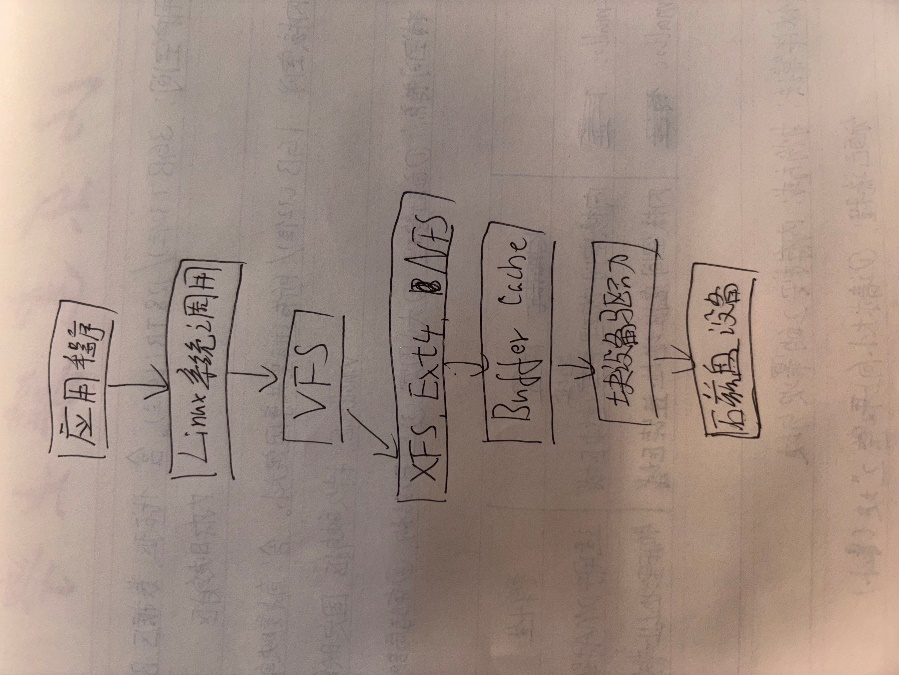
vmalloc分配**非连续**物理内存

**栈上申请 1KB 内存的程序段及特点**

char stack\_buf[1024];

**栈内存特点**：

**自动分配与释放、效率高、空间有限、后进先出。**



1. **用户空间应用程序**：通过标准库（如 open、read）调用系统接口。
2. **VFS 层**：提供统一抽象接口，管理file、dentry、inode等数据结构，屏蔽底层差异。
3. **具体文件系统层**：实现特定文件系统的逻辑（如 ext4 的块组、inode 组织）。
4. **块设备层**：管理物理块设备的读写，提供块缓存和 I/O 调度。
5. **物理存储设备**：实际的硬件存储介质。