# 文件操作

## Unlink

unlink() 函数删除文件 —— 不能删除文件夹

# Sleep

 unsigned sleep(unsigned seconds);

scandir函数：读取特定的目录数据  
表头文件：#include <dirent.h>  
定义函数：int scandir(const char \*dir, struct dirent \*\*namelist, nt (\*select) (const struct dirent \*), nt         
              (\*compar) (const struct dirent \*\*, const struct dirent\*\*));  
函数说明：  
         scandir()会扫描参数dir指定的目录文件，经由参数select指定的函数来挑选目录结构至参数namelist数组中，最后再调用  
参数compar指定的函数来排序namelist 数组中的目录数据。每次从目录文件中读取一个目录结构后便将此结构传给参数select所指的  
函数， select函数若不想要将此目录结构复制到namelist数组就返回0，若select为空指针则代表选择所有的目录结构。scandir()会  
调用 qsort()来排序数据，参数compar则为qsort()的参数，若是要排列目录名称字母则可使用alphasort(). 结构dirent定义请参考  
readdir()  
返回值 ：成功则返回复制到namelist数组中的数据结构数目，有错误发生则返回-1  
错误代码：ENOMEM 核心内存不足  
Example  
#include <dirent.h>  
main()  
{  
    struct dirent \*\*namelist;  
    int n;  
    n = scandir(".", &namelist, 0, alphasort);  
    if (n < 0)  
        perror("scandir");  
    else  
    {  
        while(n--)  
        {  
            printf("%s\n", namelist[n]->d\_name);  
            free(namelist[n]);  
        }  
        free(namelist);  
    }  
}

# 基本数据操作

## strncmp

两者不同之处是，strncmp函数是指定比较size个字符，strcmp函数比较整个字符，直到出现不同的字符或遇"**\0"**为止

### 函数声明

**int strncmp(const char \*str1, const char \*str2, size\_t n)**

### 参数

* **str1**-- 要进行比较的第一个字符串。
* **str2**-- 要进行比较的第二个字符串。
* **n**-- 要比较的最大字符数。[3]

### 返回值

该函数返回值如下：

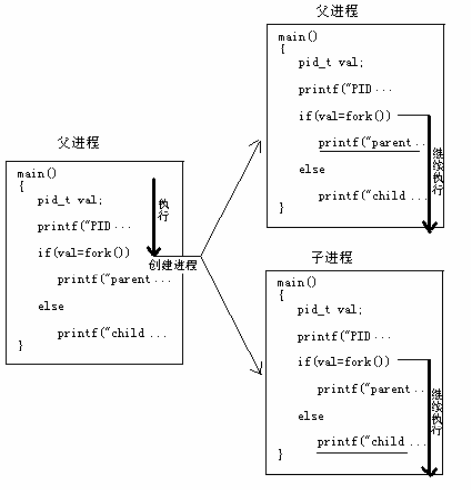
* 如果返回值 < 0，则表示 str1 小于 str2。
* 如果返回值 > 0，则表示 str2 小于 str1。
* 如果返回值 = 0，则表示 str1 等于 str2

# Fork

设计哲学：其中父进程返回子进程pid，这是由于一个进程可以有多个子进程，但是却没有一个函数可以让一个进程来获得这些子进程id，那谈何给别人你创建出来的进程。而子进程返回0，这是由于子进程可以调用getppid获得其父进程进程ID

一个现有的进程可以调用fork函数创建一个新进程。原型如下：

1. #include<unistd.h>
3. pid\_t fork(**void);**
5. 返回值:自进程中返回0，父进程返回进程id，出错返回-1



参考：

浅析fork()和底层实现

https://www.cnblogs.com/tp-16b/p/9005079.html

# 可变参数列表技巧-写日志

TraceUtils

**static void** atraceFormatBegin(**const char**\* fmt, ...) {

**const int** BUFFER\_SIZE = 256;  
 va\_list ap;  
 **char** buf[BUFFER\_SIZE];  
 va\_start(ap, fmt); //获取可变参数列表的第一个参数的地址，list是va\_list的指针，format可变参数为最左边的  
 vsnprintf(buf, BUFFER\_SIZE, fmt, ap); //将格式化数据从可变参数列表写入大小缓冲区  
 va\_end(ap); //清空可变参数列表  
 // ATRACE\_BEGIN(buf);  
}

BL\_Log::Output( BL\_LOG\_KIND\_POWERMANAGER, "%s", msg);

frameworks/base/libs/hwui/OpenGLRenderer.cpp

## 参考工程

frameworks\base\libs\hwui\utils\TraceUtils.h

frameworks\av\media\libnbaio\NBLog.cpp

消息机制

frameworks/native/vulkan/libvulkan/debug\_report.cpp

# 单例模式

ANDROID\_SINGLETON\_STATIC\_INSTANCE( GraphicBufferMapper )

把下面 C++ 代码片段中的 Singleton 替换成实际类名, 快速得到一个单例类:

## 源码分析

class Singleton {

public:

static Singleton& Instance() {

static Singleton theSingleton;

return theSingleton;

}

/\* more (non-static) functions here \*/

private:

Singleton(); // ctor hidden

Singleton(Singleton const&); // copy ctor hidden

Singleton& operator=(Singleton const&); // assign op. hidden

~Singleton(); // dtor hidden

};

## 使用方法

#include **<utils/Singleton.h>**

**namespace** android {

**class** LayoutEngine : **public** Singleton<LayoutEngine> {

*// ---------------------------------------------------------------------------*.cpp文件ANDROID\_SINGLETON\_STATIC\_INSTANCE(LayoutEngine)