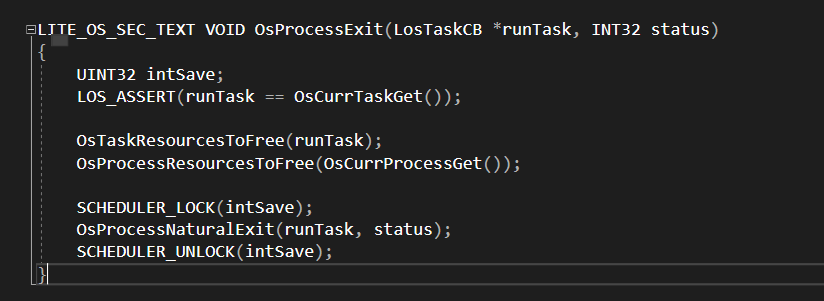
# 进程退出



该函数接受两个参数，一个是指向 LosTaskCB 结构体的指针 runTask，表示当前正在运行的任务；另一个是整型变量 status，表示进程的退出状态。

1.先使用 LOS\_ASSERT 宏判断传入的参数 runTask 是否为当前任务，确保函数被正确调用。

2.调用 OsTaskResourcesToFree 函数释放当前任务占用的资源。该函数会释放任务的所有资源，包括内存、文件描述符等等。

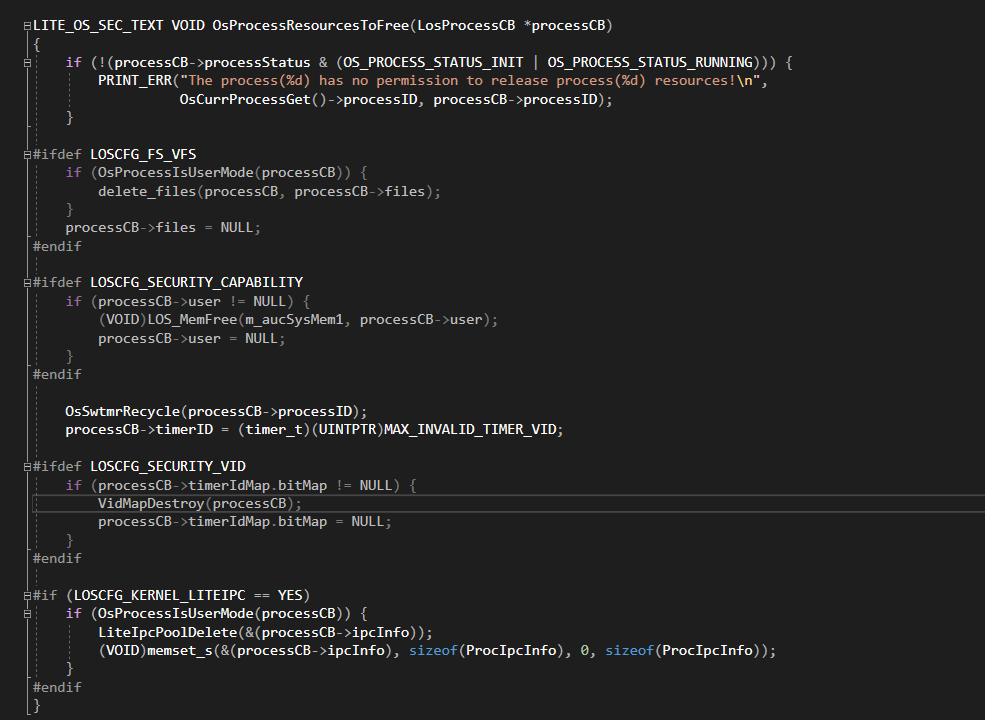
3.调用 OsProcessResourcesToFree 函数释放当前进程占用的资源。该函数会释放进程的所有资源，包括内存、文件描述符、线程等等。

4.为了避免中断干扰，使用 SCHEDULER\_LOCK 宏对调度器进行加锁。

5.在加锁的情况下，调用 OsProcessNaturalExit 函数，以自然方式（即正常退出）终止当前进程，并将退出状态设置为 status。

6.调度器解锁。

## OsProcessResourcesToFree（进程资源释放）



该函数接受一个指向 LosProcessCB 结构体的指针 processCB，表示需要释放资源的进程控制块。

函数首先检查进程的状态，判断是否具有释放资源的权限。如果进程状态不是 OS\_PROCESS\_STATUS\_INIT 或者 OS\_PROCESS\_STATUS\_RUNNING，则输出错误信息并返回。这个检查是为了确保只有初始化或者正在运行的进程才能释放资源。

接下来，根据配置选项 LOSCFG\_FS\_VFS 判断是否开启了文件系统，并且进程处于用户态。如果满足条件，调用 delete\_files 函数释放进程所打开的文件。

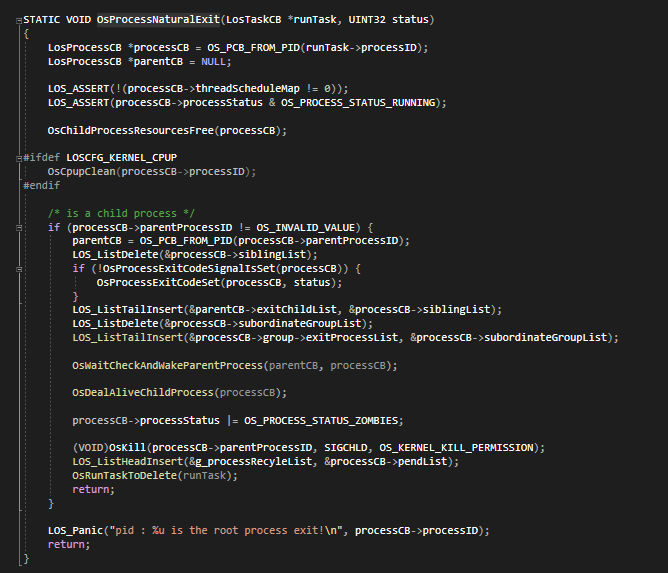
然后，根据配置选项 LOSCFG\_SECURITY\_CAPABILITY 判断是否开启了安全能力，并且进程的用户指针不为空。如果满足条件，使用 LOS\_MemFree 函数释放进程的用户指针。

接着，调用 OsSwtmrRecycle 函数回收与进程相关的软件定时器资源，并将进程的定时器ID设置为无效值。

根据配置选项 LOSCFG\_SECURITY\_VID 判断是否开启了安全VID机制，并且进程的定时器ID位图指针不为空。如果满足条件，调用 VidMapDestroy 函数销毁进程的定时器ID位图。

最后，根据配置选项 LOSCFG\_KERNEL\_LITEIPC 判断是否开启了轻量级进程间通信（LiteIPC）机制，并且进程处于用户态。如果满足条件，调用 LiteIpcPoolDelete 函数删除进程的LiteIPC池，并使用 memset\_s 函数将进程的IPC信息结构体清零。

## OsProcessNaturalExit（进程正常退出）



首先，函数通过任务控制块 runTask 中的进程ID获取进程控制块 processCB。然后，声明一个指向父进程控制块的指针 parentCB。

接下来，通过断言语句 LOS\_ASSERT 检查进程是否具有正确的状态。其中，第一个断言检查进程的线程调度映射是否为0，第二个断言检查进程的状态是否为运行状态。如果断言失败，则会触发断言错误。

然后，调用 OsChildProcessResourcesFree 函数释放子进程占用的资源。该函数用于释放子进程的文件、安全能力和定时器等资源。

如果进程的父进程ID不是无效值，表示当前进程是一个子进程。接下来，执行以下操作：

从父进程控制块中删除当前子进程的兄弟进程列表项。

如果当前子进程没有设置退出码信号，调用 OsProcessExitCodeSet 函数设置退出码。

将当前子进程的列表项插入到父进程的退出子进程列表中。

从当前子进程的组成员列表中删除当前子进程的列表项。

将当前子进程的列表项插入到当前子进程所属组的退出进程列表中。

调用 OsWaitCheckAndWakeParentProcess 函数检查并唤醒父进程。

调用 OsDealAliveChildProcess 函数处理当前子进程的存活状态。

将当前子进程的状态设置为僵尸态。

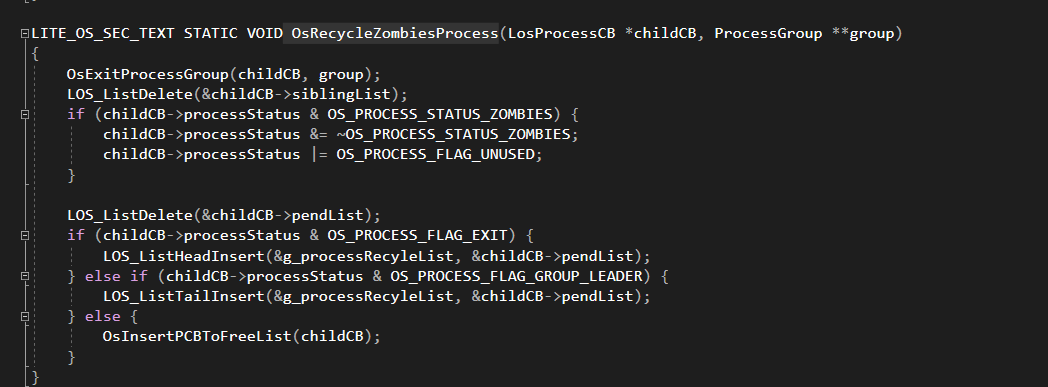
调用 OsKill 函数向父进程发送 SIGCHLD 信号，表示子进程已退出。

将当前子进程的列表项插入到全局进程回收列表中。

调用 OsRunTaskToDelete 函数删除当前任务的运行任务。

最后，如果进程的父进程ID是无效值，表示当前进程是根进程，会触发Panic并输出提示消息。

## OsRecycleZombiesProcess（回收僵尸进程）



首先，调用 OsExitProcessGroup 函数将子进程从其所属的进程组中移除，并更新进程组的状态。

然后，通过 LOS\_ListDelete 函数将子进程的兄弟链表节点删除。

接下来，判断子进程的状态是否为僵尸态（OS\_PROCESS\_STATUS\_ZOMBIES标志位）。如果是，则清除子进程的僵尸态标志位，并将其标记为未使用（OS\_PROCESS\_FLAG\_UNUSED标志位）。

再次使用 LOS\_ListDelete 函数将子进程从挂起列表中删除。

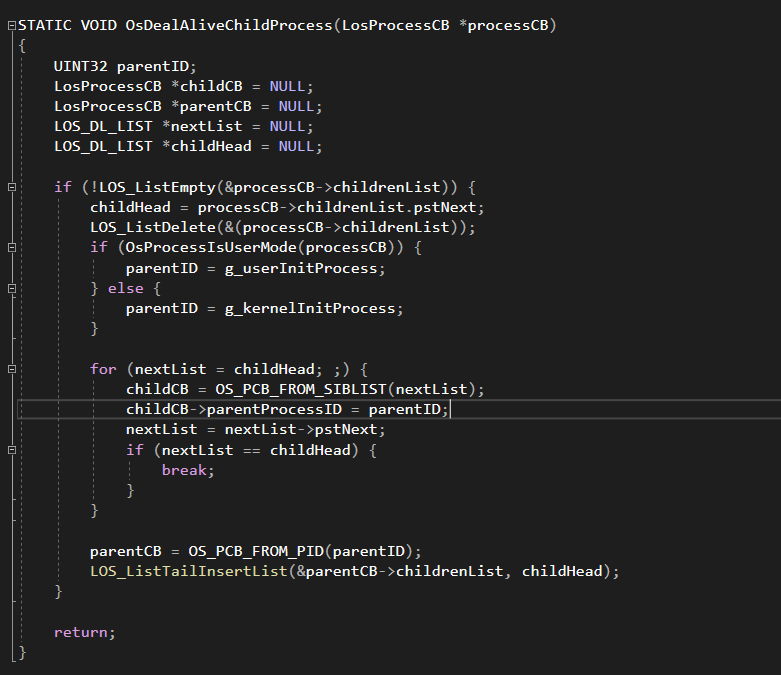
根据子进程的状态进行不同的处理：

如果子进程的状态是退出态（OS\_PROCESS\_FLAG\_EXIT标志位），则将子进程的挂起列表插入到全局的进程回收列表头部。

如果子进程的状态是组长态（OS\_PROCESS\_FLAG\_GROUP\_LEADER标志位），则将子进程的挂起列表插入到全局的进程回收列表尾部。

否则，将子进程的进程控制块插入到空闲进程控制块列表中（调用 OsInsertPCBToFreeList 函数）。

## OsDealAliveChildProcess（处理存活子进程）



首先，如果当前进程存在子进程，则从父进程的子进程链表中删除该子进程，并初始化父进程的 ID。

接下来，使用循环遍历子进程链表，将每个子进程的父进程 ID 设置为初始化时指定的父进程 ID，即 g\_userInitProcess 或 g\_kernelInitProcess。

然后，将子进程链表插入到相应父进程的子进程链表的末尾。这里使用 LOS\_ListTailInsertList 函数将整个子进程链表插入到父进程的子进程链表的末尾。

总之，这段代码实现了将父进程的子进程列表重新挂载到新的父进程上的操作。它会遍历所有子进程，将它们的父进程 ID 设置为初始化时指定的父进程 ID，然后将子进程链表插入到相应的父进程的子进程链表的末尾。