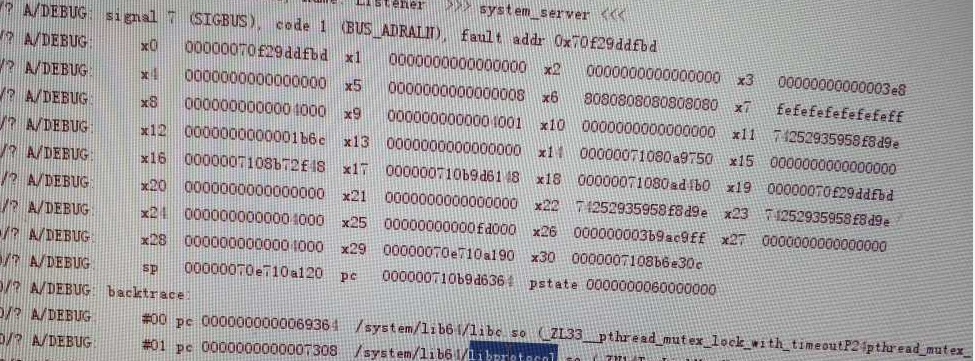
# TODO

signal 7 (SIGBUS), code 1, fault add android

<https://www.bbsmax.com/A/x9J2xALVd6/>



stCmd\* CmdQueue::waitCmd(uint16\_t seq, **int** timeoutInMs) {  
 **struct** timeval tTime;  
 **struct** timespec timer = {0, 0};  
 **long long** duration;  
 **int** ret ;  
  
 stCmd \*pCmd = NULL;  
  
 gettimeofday(&tTime,NULL);  
 duration = ((**long long**)tTime.tv\_sec)\*1000000+tTime.tv\_usec + timeoutInMs\*1000;  
 timer.tv\_sec=duration/1000000;  
 timer.tv\_nsec=(duration%1000000)\*1000;  
  
 pthread\_mutex\_lock(&cmdLock);  
 uint16\_t index = seq % MAX\_CMD\_COUNT;  
 pCmd = &mCmds[index];  
  
 ret = pthread\_cond\_timedwait(&pCmd->cond, &cmdLock, &timer);  
  
 pthread\_mutex\_unlock(&cmdLock);  
 pthread\_cond\_destroy(&pCmd->cond);  
 **if** (!ret)  
 **return** pCmd;  
 **return** NULL;  
}

<https://blog.csdn.net/fingding/article/details/36653139>

#pragma pack

<https://www.jianshu.com/p/90a6eef329ec>

#include **<pthread.h>在哪个库中？**

<https://blog.csdn.net/koozxcv/article/details/51728793>

android C++常用库

# 常量方法

system/connectivity/wificond/ipc\_constants.h

**namespace** android {  
**namespace** wificond {  
**namespace** ipc\_constants {  
  
**extern const char** kServiceName[];  
  
} *// namespace ipc\_constants*} *// namespace wificond*} *// namespace android*

system/connectivity/wificond/ipc\_constants.cpp

**namespace** android {  
**namespace** wificond {  
**namespace** ipc\_constants {  
  
**const char** kServiceName[] = **"wificond"**;  
  
} *// namespace ipc\_constants*} *// namespace wificond*} *// namespace android*

## 使用

**using** android::wificond::ipc\_constants::kServiceName;

**void** RegisterServiceOrCrash(**const** android::sp<android::IBinder>& service) {  
 android::sp<android::IServiceManager> sm = android::defaultServiceManager();  
 CHECK\_EQ(sm != NULL, **true**) << **"Could not obtain IServiceManager"**;  
  
 CHECK\_EQ(sm->addService(android::String16(kServiceName), service),  
 android::NO\_ERROR);  
}

# 多线程变编程

pthread\_cond\_t

pthread\_mutex\_t： 互斥锁，多线程中对共享变量的包保护

pthread\_cond\_t： 线程间同步，一般和pthread\_mutex\_t一起使用，以防止出现逻辑错误，即如果单独使用条件变量，某些情况下（条件变量前后出现对共享变量的读写）会出现问题

pthread\_mutexattr\_init(&tAttr);  
pthread\_mutexattr\_settype(&tAttr,PTHREAD\_MUTEX\_RECURSIVE);  
pthread\_mutex\_init(&tLock, &tAttr);  
pthread\_mutexattr\_destroy(&tAttr);

*PTHREAD\_MUTEX\_RECURSIVE*

如果一个线程对这种类型的互斥锁重复上锁，不会引起死锁，一个线程对这类互斥锁的多次重复上锁必须由这个线程来重复相同数量的解锁，这样才能解开这个互斥锁，别的线程才能得到这个互斥锁。如果试图解锁一个由别的线程锁定的互斥锁将会返回一个错误代码。如果一个线程试图解锁已经被解锁的互斥锁也将会返回一个错误代码。这种类型的互斥锁只能是进程私有的（作用域属性为*PTHREAD\_PROCESS\_PRIVATE*）。

# Wunused-parameter

/pipe/mainpipe.c:6:10: error: unused parameter 'argc' [-Werror,-Wunused-parameter]

LOCAL\_CFLAGS += -Wno-unused-parameter