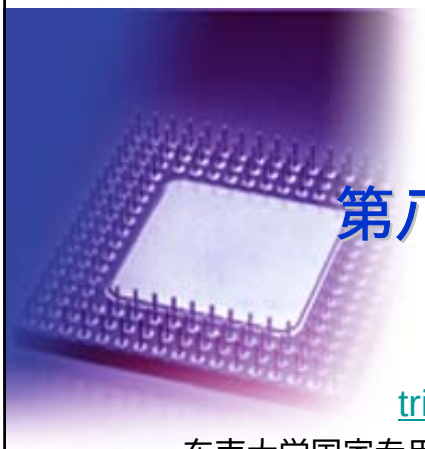
CNASIC



第八讲 代码的调试

凌 明

trio@seu.edu.cn

东南大学国家专用集成电路系统工程技术研究中心

CNASIC

目 录

CNASIC

Bugs vs Debugging

- 没有Bug的就不是软件☺
- 核心的问题是：
  - 怎样发现程序错误的根源?
  - 怎样在软件中自动地查出这个错误?
  - 怎样修正这个错误?
  - 怎样避免这个错误?

CNASIC

初学者的困惑

- 在错误面前一筹莫展
  - 拼命的单步，但却不知道该关心什么？
  - 根本就不单步跟踪程序，或者不敢往下层函数跟踪
- 总是发现编译器的“Bug”
- 随便的，没有目的的修改代码，祈求奇迹的出现

CNASIC

Debugging的手段和工具

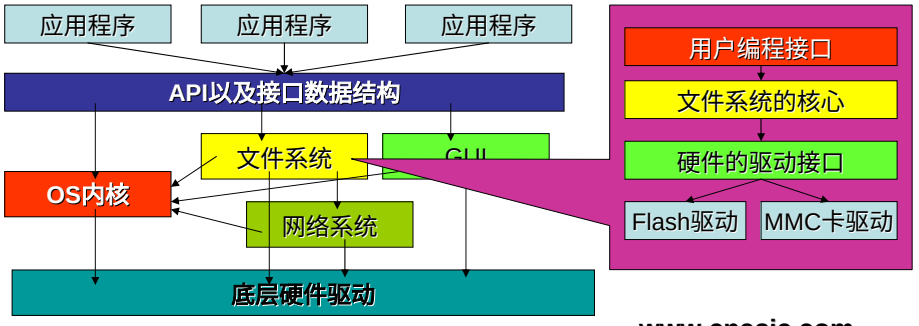
- 一般的Debug工具都提供以下功能
  - 断点与数据观测点（Break point and Watch point）
  - 单步 (Step)
  - 寄存器的观察与修改(Register)
  - 变量的观察与修改 (Watch)
  - 内存观察与修改 (Memory)
  - 调用栈 (Call Stack)

程序的可观测性与可控制性

CNASIC

Bug的定位－关注接口

- 好的软件架构总是基于层次性的架构，通过相对单纯的数据结构和接口函数与外界交互。



CNASIC

Bug的定位－关注接口

- 因次Bug的定位首先应该从上层逐渐往下层排查，将断点设在上层函数的入口，单步执行跟踪程序的流程，特别关注底层函数的执行是否正确（主要是观察他的返回值是否正确），将搜索的方位逐渐缩小，最后定位在一个函数内部。

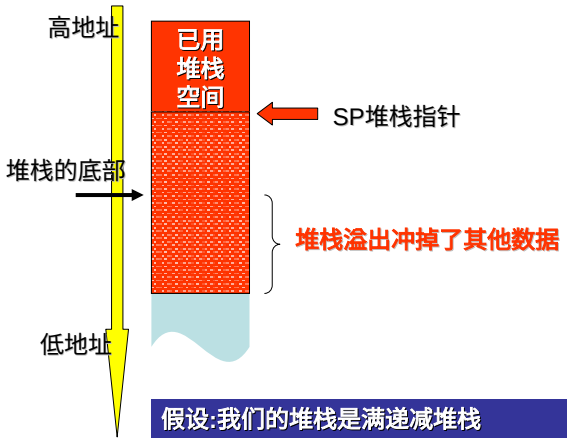
CNASIC

Bug的定位－关注内存的访问越界

- C语言的灵活性，指针的应用，以及C语法的宽容性很容易造成代码的错误，这其中最主要的就是内存单元的溢出
  - 堆栈溢出
  - 缓冲区溢出
  - 数组越界
- 因此在将错误局限在一个函数中的时候，应该关注内存的问题了

CNASIC

堆栈溢出



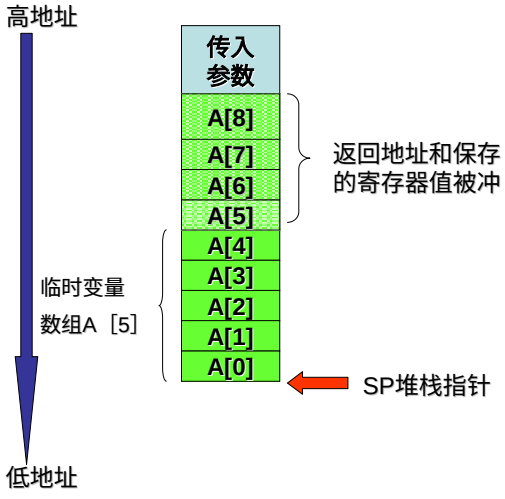
主要现象:
1,某些全局变量莫名其妙被改
2,其他任务工作不正常,比如:
函数调用不正常(Prefech Abort)
局部变量莫名其妙被改
其他所有的死机与崩溃

主要原因:
1,任务的堆栈预留的太小
2,任务中开设了大的临时变量
3,过深的函数递归,消耗了堆栈

假设:我们的堆栈是满递减堆栈

CNASIC

堆栈的缓冲区溢出

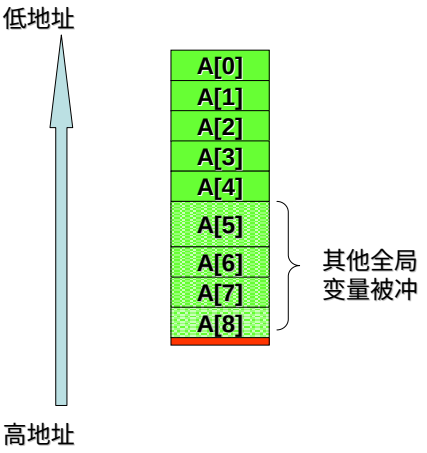


主要现象:
1,某些局部变量莫名其妙被改
2,函数返回的时候崩溃

主要原因:
1,临时数组变量越界
2,注意 strcpy();sprintf();
memcpy()函数的目的缓冲区
是否越界，strycpy()函数的
原字符串是否正常以'\0'结
束；memcpy()函数的拷贝数
是否正常

CNASIC

全局变量和动态缓冲区溢出



主要现象:
1,某些全局变量莫名其妙被改
2,如果被冲的部分是动态分配的缓冲区，一方面会造成其他数据的错误；另一方面会造成无法释放被冲的动态内存从而造成内存泄漏

主要原因:
1,全局数组变量越界
2,注意 strcpy();sprintf();memcpy()函数的目的缓冲区是否越界，strycpy()函数的原字符串是否正常以'\0'结束；memcpy()函数的拷贝数是否正常

CNASIC

关注边界情况

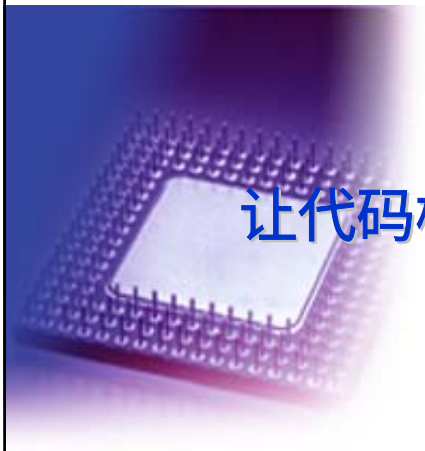
- 程序中需要考虑的边界情况
  - 数组的上限
  - 循环的次数
  - 链表的头部和尾部在插入新节点，或删除的情况下的特殊处理
  - 输入参数的极限情况（0，指针为空，复数，最大的情况等等）

CNASIC

Bug的修正

- 别急着改，想想，再想想，想清楚了再动手。
- 考虑所作的修改可能会对系统造成的新的影响是什么？
- 我的修改会对其他人的代码造成影响吗？
- 是否会对全局的数据结构或者函数接口定义作修改？如果是，如何通知所有的其他人？
- 修改完了，应该有详细的文档，代码注释，并对修改过的代码进行回归测试

CNASIC



让代码检查自己的错误

CNASIC 利用断言－ASSERT

- aasert 是个只有定义了DEBUG 才起作用的宏，如果其参数的计算结果为假，就中止调用程序的执行

```
#ifdef DEBUG
        void _Assert(char* , unsigned); /* 原型*/
        #define ASSERT(f) \
        if(f)  NULL; \
        else \
                _Assert(__FILE__ , __LINE__)
#else
        #define ASSERT(f)  NULL
#endif
```

```
void _Assert(char* strFile, unsigned uLine)
{
fflush(stdout);
fprintf(stderr, "\nAssertion failed: %s, line %u\n"\
,strFile, uLine);
fflush(stderr);
abort();
}
```

CNASIC 利用断言－ASSERT

```
/* memcpy ——— 拷贝不重叠的内存块*/
void memcpy(void* pvTo, void* pvFrom, size_t size)
{
        void* pbTo = (byte*)pvTo;
        void* pbFrom = (byte*)pvFrom;
        ASSERT(pvTo != NULL && pvFrom != NULL);
        ASSERT(pbTo>=pbFrom+size || pbFrom>=pbTo+size);
        while(size-->0)
                *pbTo++ = *pbFrom++;

        return(pvTo);
}
```

CNASIC

利用调试宏

- 参数的合法性检查
- 堆栈的监控（High Water Mark）
- 内存数据结构的监控（Audit）
- 调试信息的记录与输出 (对网络等不能单步跟踪的调试非常有用)

CNASIC

其他的方法和工具

- 好的架构设计与数据结构设计
- 代码检查（Code Review ）
- 编译器的警告与Lint工具
- 好的 Coding Style