Pitfalls in Embedded Software

..and how to avoid them

Sagar Behere

31 March 2014

What is wrong with this code?

```
unsigned int count = BigValue;
for (int i = 0; i < count; i++) {
    ;
}
```

Who am I?

- 10+ years of systems development
- Diesel engines, traction control, autonomous driving
- Robotics, artificial intelligence, unmanned aerial vehicles

What will I talk about?

1. Inconsistent bugs
2. How to debug
3. Take-home puzzles

Race conditions

Thread 1:

```
global_counter += 1;
```

Thread 2:

```
global_counter = 0;
```

What if the increment operation cannot be performed atomically?

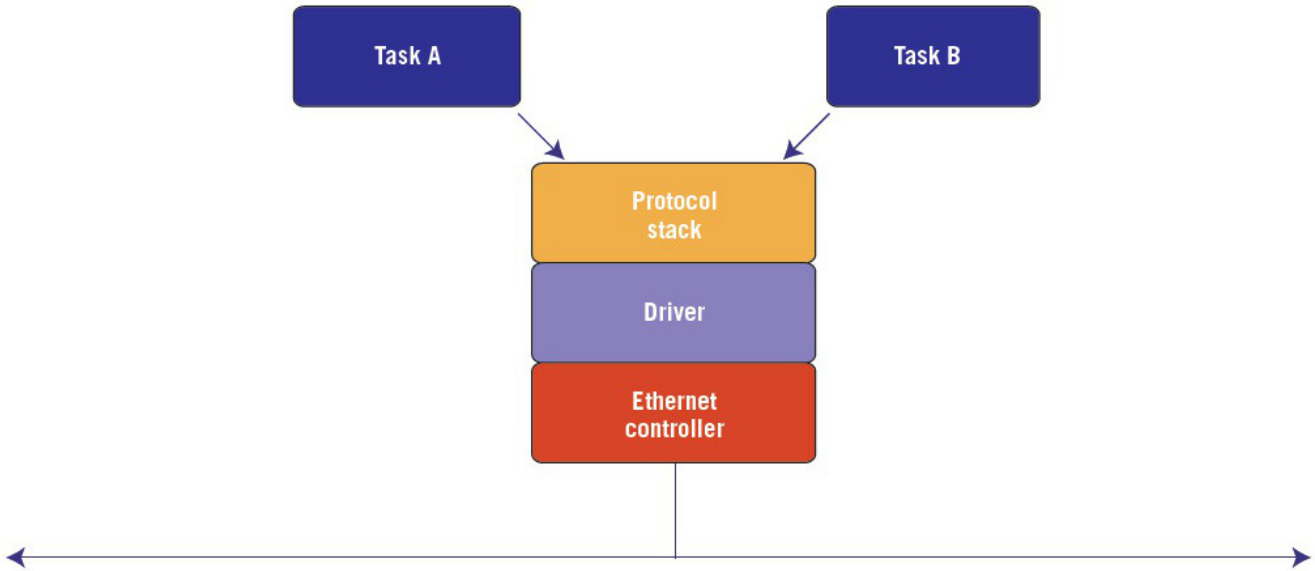
Best practices:

- Surrounded critical sections by preemption limiting mechanisms
- For ISRs: Interrupt must be disabled
- For RTOS Tasks: Mutexes

Tip

Look up Scoped Mutexes

Non-reentrant functions



ETH driver functions MUST manipulate the same global objects!

Best practice:

- Use Mutexes
  - But is that sufficient?

Missing volatile keyword

```
g_alarm = ALARM_ON;
//
// Code that does not access g_alarm
//
g_alarm = ALARM_OFF;
```

What happens when compiled with optimization enabled?

Best practices: Use volatile to declare every

- Global variable accessed by an ISR
- Global variable accessed by two or more RTOS tasks
- Pointers to memory-mapped registers
- Delay loop counters

Stack overflow

- Effects and timing both unpredictable
- Embedded systems are especially vulnerable
  - Limited RAM. No virtual memory
  - RTOS based designs typically have one-stack-per-thread; each must be correctly sized
  - Interrupt handlers may try to use those

Best practice:

1. During init, paint an unlikely memory pattern throughout the stack.
2. During runtime, supervisor task periodically checks for 'scratches in the paint' above a 'high water mark'

Heap fragmentation

1. Start with a 10KB heap
2. malloc() two blocks of 4 KB
3. free() one of the blocks
4. malloc() a block of 6 KB

What will happen?

Best practice:

- All memory requests should have the same size.

Memory leaks

- Number of malloc()s does not match number of free()s

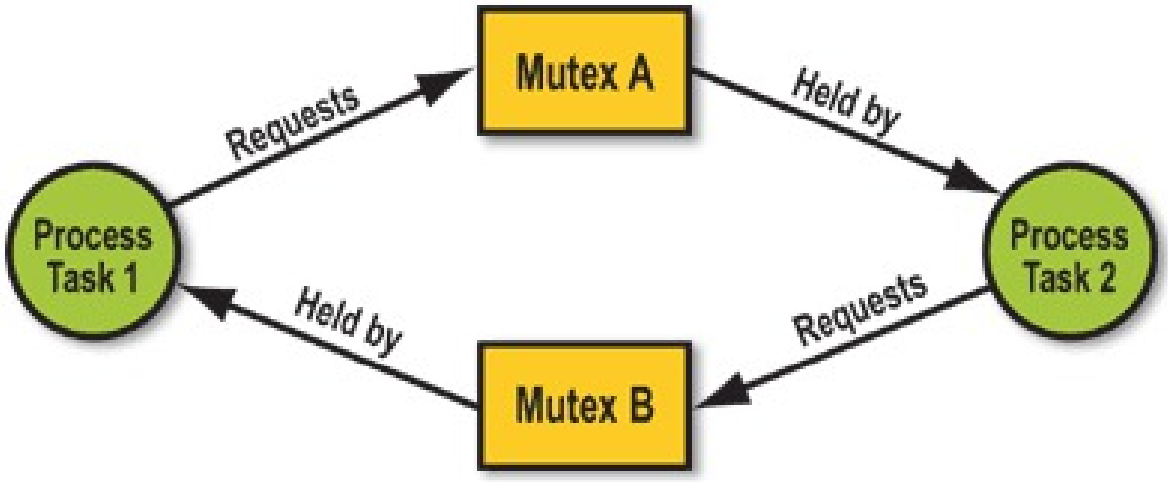
```
int *x;
x = malloc(sizeof(int));
*x = 100;
```

- Why is the above code dangerous?

Best practice:

- Design patterns: Clearly define ownership pattern or lifetime of each type of heap-allocated object.
- Valgrind!!!

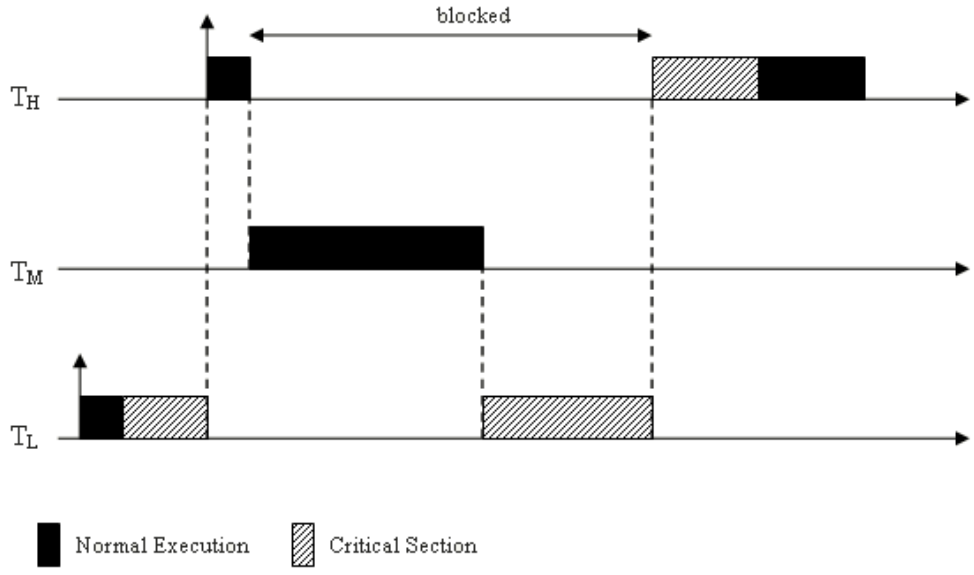
Deadlocks



Best practices:

- Do not attempt simultaneous acquisition of two or more mutexes
- Assign an ordering to all mutexes. Always acquire multiple mutexes in that same order

Priority inversion



Not always reproducible

Best practice:

1. Choose RTOS that has priority-inversion work-arounds in its API
2. Do not forget execution time cost of work-around

Jitter

An example of jitter in the timing of a 10-ms task.

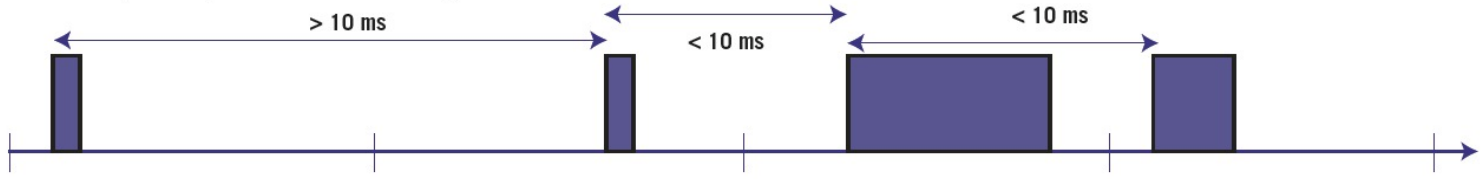


Figure 3

Best practice: Set correct relative priorities .. or cheat!

Jitter is affected by relative priority.

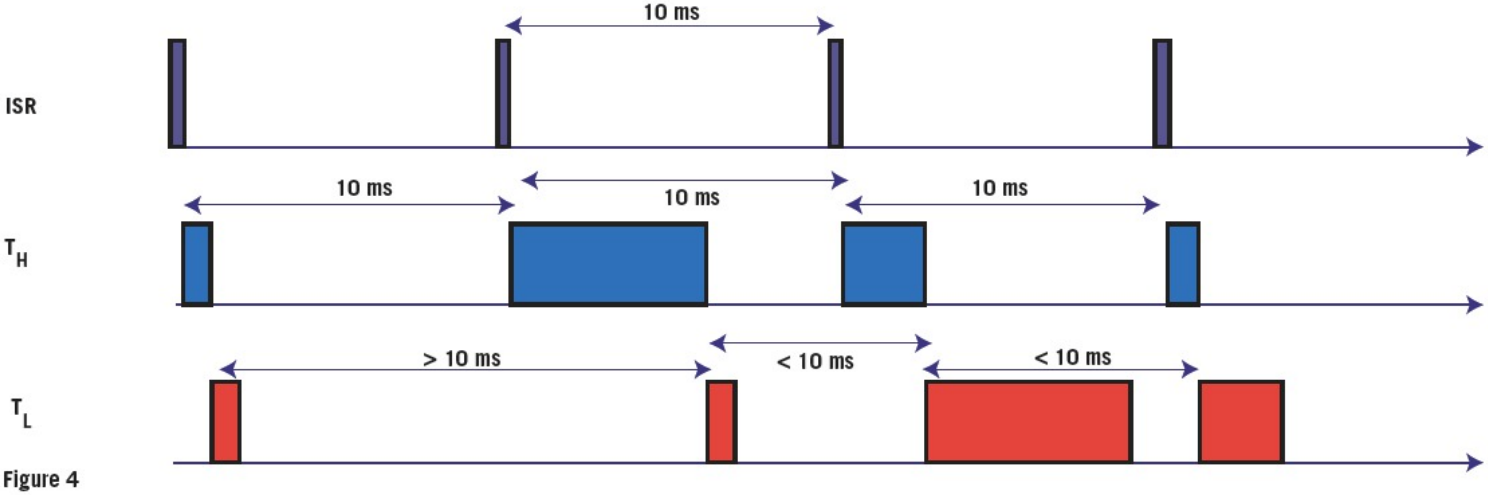
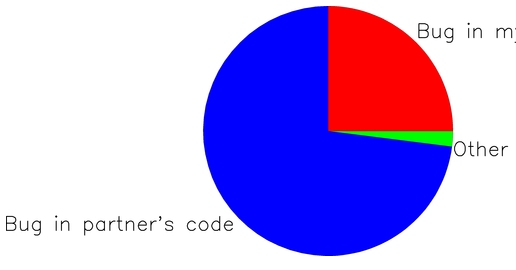
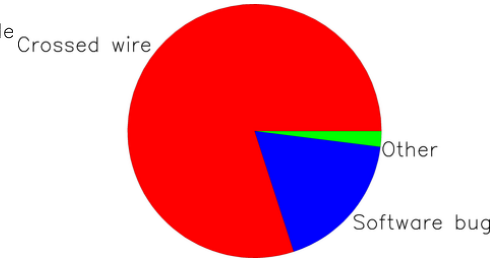


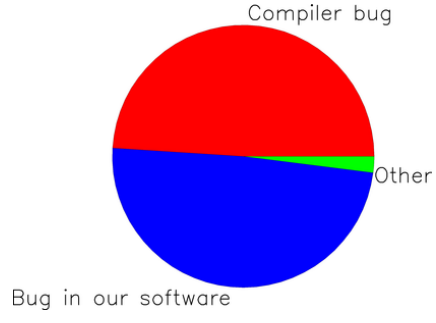
Figure 4

A scientific approach to debugging

1. Verify the bug, determine correct behavior
2. Stabilize, isolate, minimize
    2. Can you make the bug appear consistently?
    2. What is the minimum input needed to make it appear?
3. Estimate a probability distribution





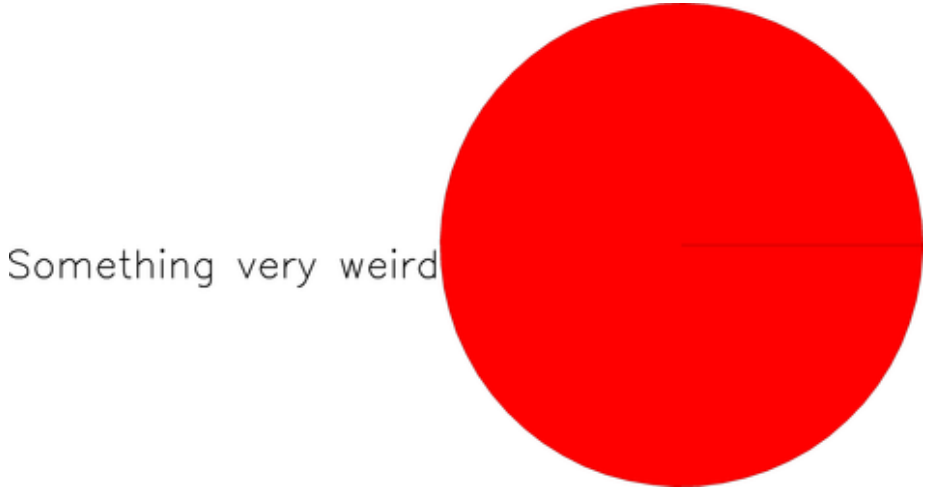


A scientific approach to debugging

4. Devise and run an experiment
5. Iterate - but remember to change one thing at a time
6. Fix bug, verify fix
7. Undo changes
8. Find the bug's parents, friends and relatives

If you are stuck

What if the probability distribution looks like this?



- Take a break
- Talk with someone
- Sit and stare at the code
- Reduce size of failure-inducing input
- Find a tool to bring out more information

Take home puzzle #1

```
 1 #include <iostream>
 2 #include <string>
 3 using namespace std;
 4 int main(void)
 5 {
 6    string s = "abc";
 7    char delim = ':';
 8    unsigned int position = s.find(delim);
 9    // if no matches are found, find() returns string::npos, else position of delim
10    // see http://www.cplusplus.com/reference/string/string/find/
11    if(string::npos != position) {
12       cout << delim << " FOUND in " << s << endl;
13    } else {
14       cout << delim << " NOT FOUND in " << s << endl;
15    }
16    return 0;
17 }
```

Take home puzzle #2

```
#include <stdio.h>

main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
)&&t == 2 ? _<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')(){}#}w')){}{nl]'/+#n';d}rw' i;# ){n\
l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

References & further reading

Material in these slides has been taken from..

- Embedded.com: Five top causes of nasty embedded software bugs
- Embedded.com: Five more top causes of nasty embedded software bugs
- John Regehr's blog post: How to debug?

..you really should read those articles!