Concurrency and Race conditions

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

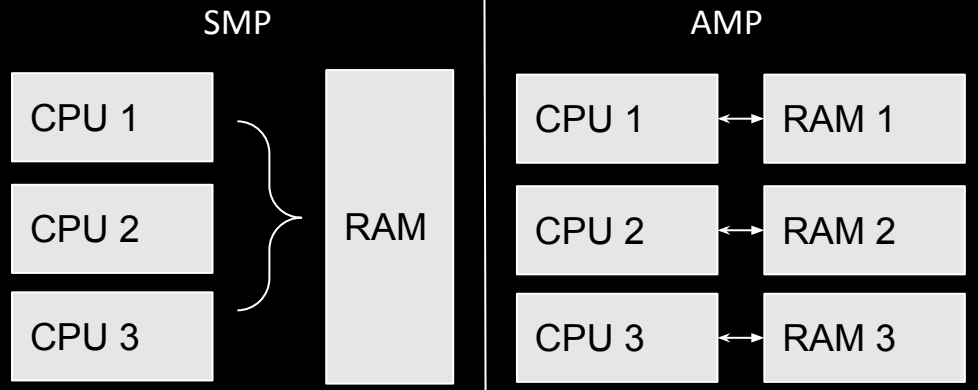
Learning objectives:

Understand concurrency concerns in the kernel.

Understand what race conditions are and how to avoid them.

Kernel Drivers and Concurrency

- concurrency bugs are “easiest to create and some of the hardest to find”
- Made ubiquitous by Symmetric Multiprocessing (SMP systems)
- Why not use AMP instead?
  - Generally not supported by Linux



Linux Device Drivers 3rd Edition Chapter 5
https://techdifferences.com/difference-between-symmetric-and-asymmetric-multiprocessing.html

scull Race Condition

```
/* follow the list up to the right position */
dptr = sculld_follow(dev, item);
if (!dptr->data) {
        dptr->data = kmalloc(qset * sizeof(void *), GFP_KERNEL);
        if (!dptr->data)
                goto nomem;
        memset(dptr->data, 0, qset * sizeof(char *));
}
```

```
kfree(dptr->data);
dptr->data=NULL;
```

- 1st process hits kmalloc
  - While in kmalloc 2nd process hits kmalloc
- 1st process completes kmalloc and continues past end of the function.
- 2nd process completes kmalloc and continues past end of the function.
- What is stored in dptr->data?
- What happens to the kmalloc from the 1st process?

Linux Device Drivers 3rd Edition Chapter 5
https://github.com/cu-ecen-5013/ldd3/blob/master/sculld/main.c#L240

scull Race Condition

- How likely is it that you'll hit this race condition? Can I just ignore it because it's so unlikely?
  - "one-in a million events can happen every few seconds"
  - Probabilities can change drastically in unexpected ways, when hardware platforms or other parts of the software changes.
  - It usually happens the first time your boss (or customer) tries to run your software!

Sources of Concurrency Issues

- Multiple user space processes are running
- SMP systems execute your code simultaneously on different processors
- Kernel code is preemptible, driver code can lose the processor at any time.
- Device interrupts are completely asynchronous.
- Your device could disappear while you are working with it.

Avoiding Race Conditions

- Avoid shared resources when possible
  - Avoid global variables
- If we have no global variables does that mean no shared resources?
  - No, since allocating memory and passing the pointer to the kernel could create a sharing situation.

Avoiding Race Conditions

- Is hardware or another resource shared beyond a single thread of execution?
- Is it possible the thread could encounter an inconsistent view of the resource?
  - If the answer to both questions is yes, you must explicitly manage access.
- Manage access through locking or mutual exclusion

Kernel Resource Requirements

- When you notify the kernel about an object it must continue to exist and function until no outside references to it exist.
  - No object can be made available to the kernel until it can function properly.
    - Your driver specific content must be initialized before providing
  - References to objects must be tracked
    - In many cases the kernel tracks references for you.