CSE 451: Operating Systems

Lab Section: Week 2

Today

- Project 1
- Project 2
- User space vs. kernel space
  (this may be useful for tomorrow's quiz☺)

Office Hours

- Gary Kimura
  - MWF 12:00-1:00 CSE 476
  - (after class)
- Mark Zbikowski
  - MWF 9:30-10:30 CSE 591
  - (before class)

Project 1

- Due January 26, 11:59 pm
    - same time as Project 2
    - you can resubmit until then
- Asynchronous I/O extra credit
    - this is actually a lot of work
    - if you want to do it:

      ... read Chapter 9 of Windows Internals

      ... especially "I/O Processing"
- Questions?

Project 1

```
NTSTATUS
NTReadFile(...) {
    ...
    CSE451Info.readcalls++;
    return status;
}
```

This is broken when run on a multiprocessor. Why?

Project 1

```
NTSTATUS
NTReadFile(...) {
   ...
   int tmp = CSE451Info.readcalls;
   CSE451Info.readcalls = tmp+1;
   return status;
}
```

`CSE451Info.readcalls++`

Project 1

Thread 1

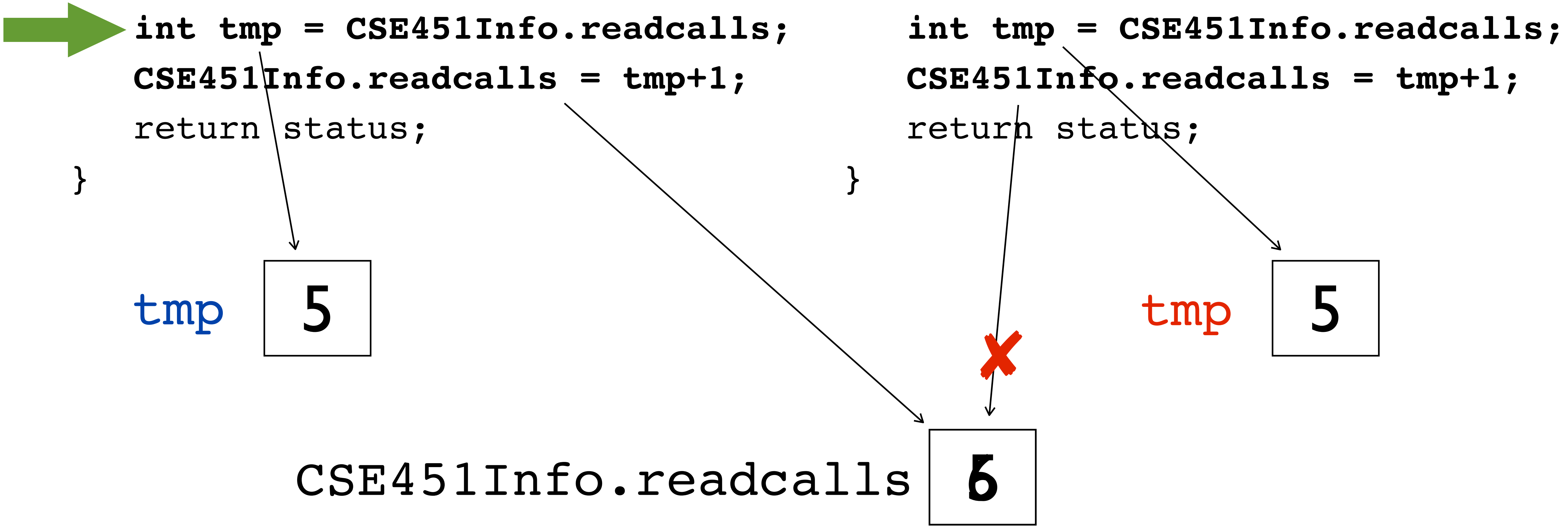
`NTReadFile("foo.txt")`

```
NTSTATUS
NTReadFile(...) {
    ...
    int tmp = CSE451Info.readcalls;
    CSE451Info.readcalls = tmp+1;
    return status;
}
```

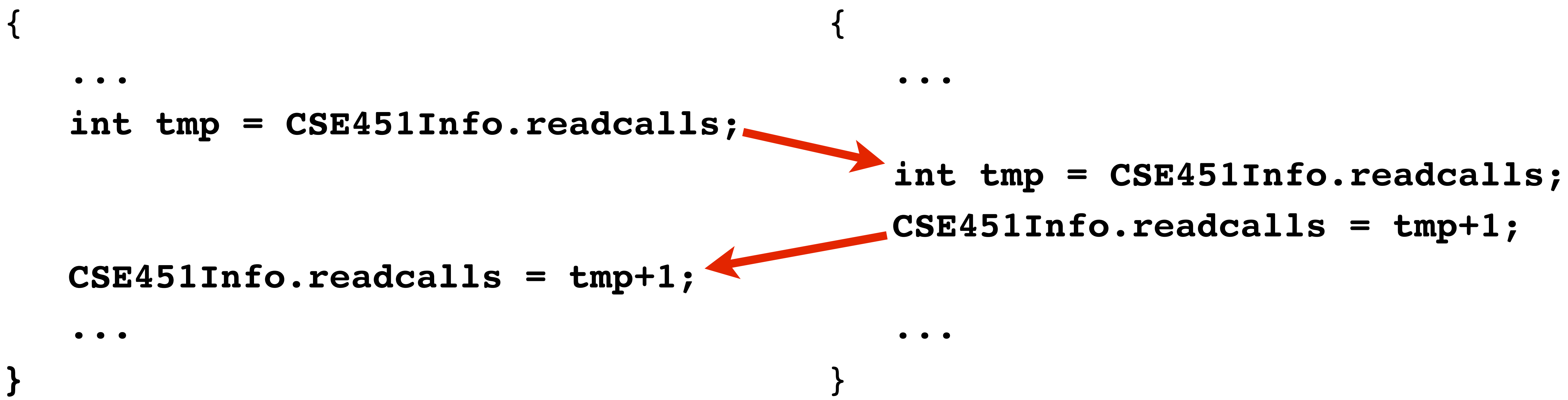
Thread 2

`NTReadFile("bar.c")`

```
NTSTATUS
NTReadFile(...) {
    ...
    int tmp = CSE451Info.readcalls;
    CSE451Info.readcalls = tmp+1;
    return status;
}
```



Project 1



Thread 1

```
{
    ...
    int tmp = CSE451Info.readcalls;


    CSE451Info.readcalls = tmp+1;
    ...
}
```

Thread 2

```
{
    ...

    int tmp = CSE451Info.readcalls;
    CSE451Info.readcalls = tmp+1;

    ...
}
```

Visualizing the bug as
a bad interleaving

How can we fix this data race?

(you need to do this for Project 2)

- Use a *mutex*
  - short for “MUtual EXclusion”
  - Acquire(mutex) begins a critical section
  - Release(mutex) ends a critical section

- Sometimes called a *lock*
  - Lock(lock) same as Acquire(mutex)
  - Unlock(lock) same as Release(mutex)

- Let’s see how it works ...

How can we fix this data race?

(you need to do this for Project 2)

Thread 1

```
{
    ...
    Acquire(&SomeMutex);
    int tmp = CSE451Info.readcalls;
    CSE451Info.readcalls = tmp+1;
    Release(&SomeMutex);
    ...
}
```

Thread 2

```
{
    ...









    Acquire(&SomeMutex);
    int tmp = CSE451Info.readcalls;
    CSE451Info.readcalls = tmp+1;
    Release(&SomeMutex);
    ...
}
```

Cannot complete until Thread 1 releases SomeMutex

Project 2

- Two goals
  - make Project 1 thread safe
  - support event histories (see project doc)
- Due January 26, 11:59 pm
  - available on course website

Today

- ~~Project 1~~
- ~~Project 2~~
- User space vs. kernel space
  (this may be useful for tomorrow's quiz☺)

User space vs. Kernel space

User
Space

process

process

process

Kernel
Space

- Isolate processes
- Multiplex hardware safely

Hardware

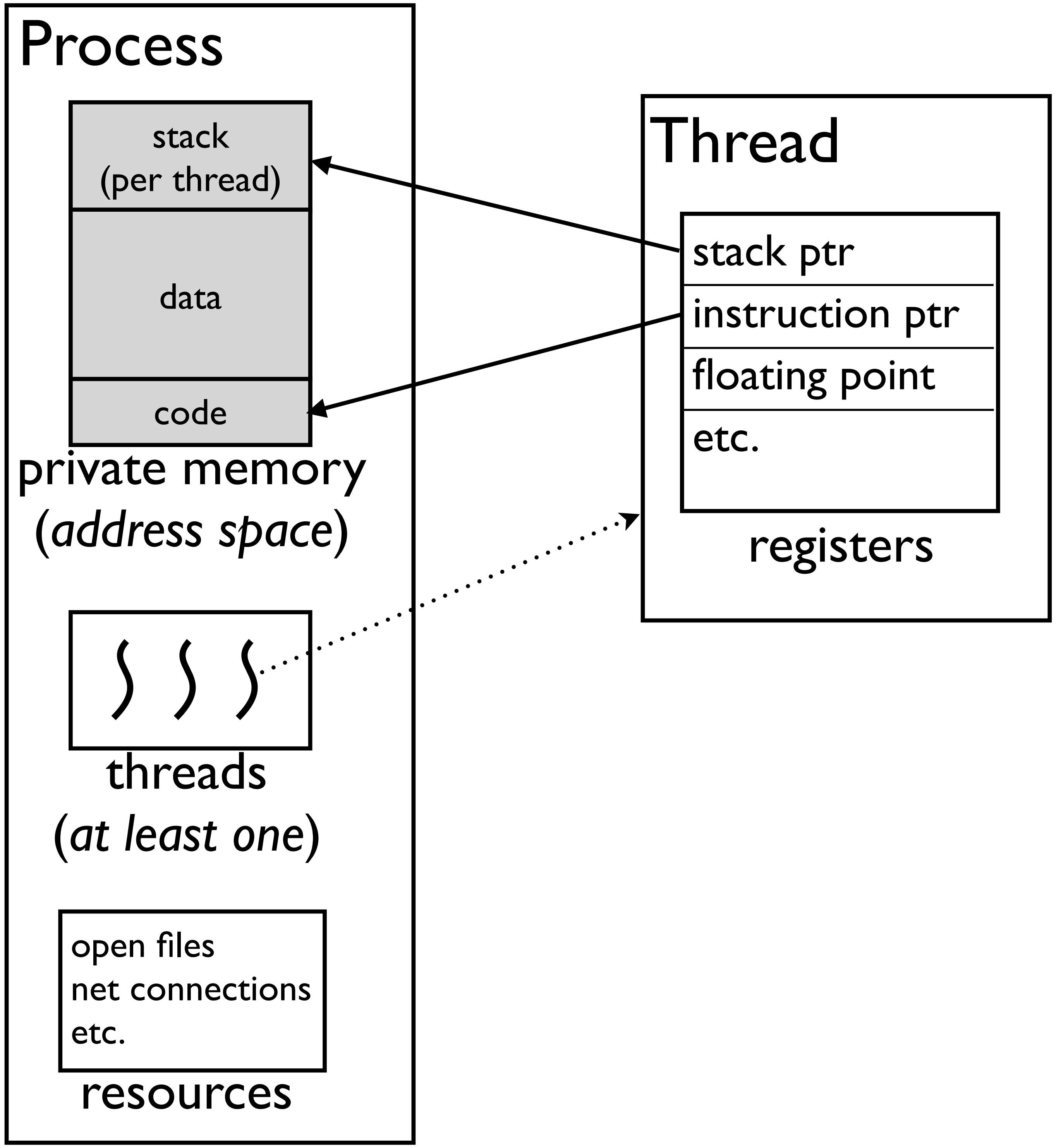
cpus

hard disks

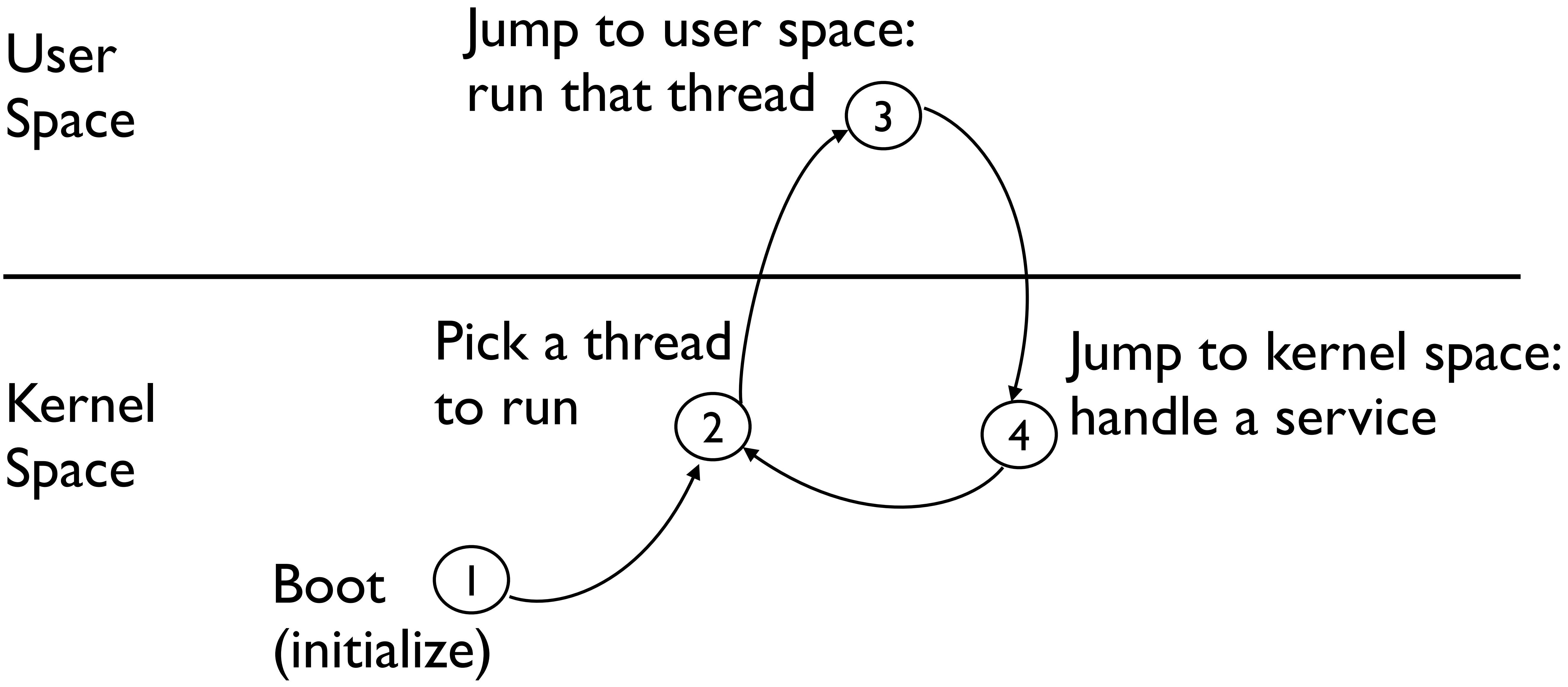
memory

devices

What's in a process?



Life of an Operating System



User Space

Kernel Space

Jump to user space: run that thread

Pick a thread to run

Jump to kernel space: handle a service

Boot (initialize)

What causes a transition to kernel mode?

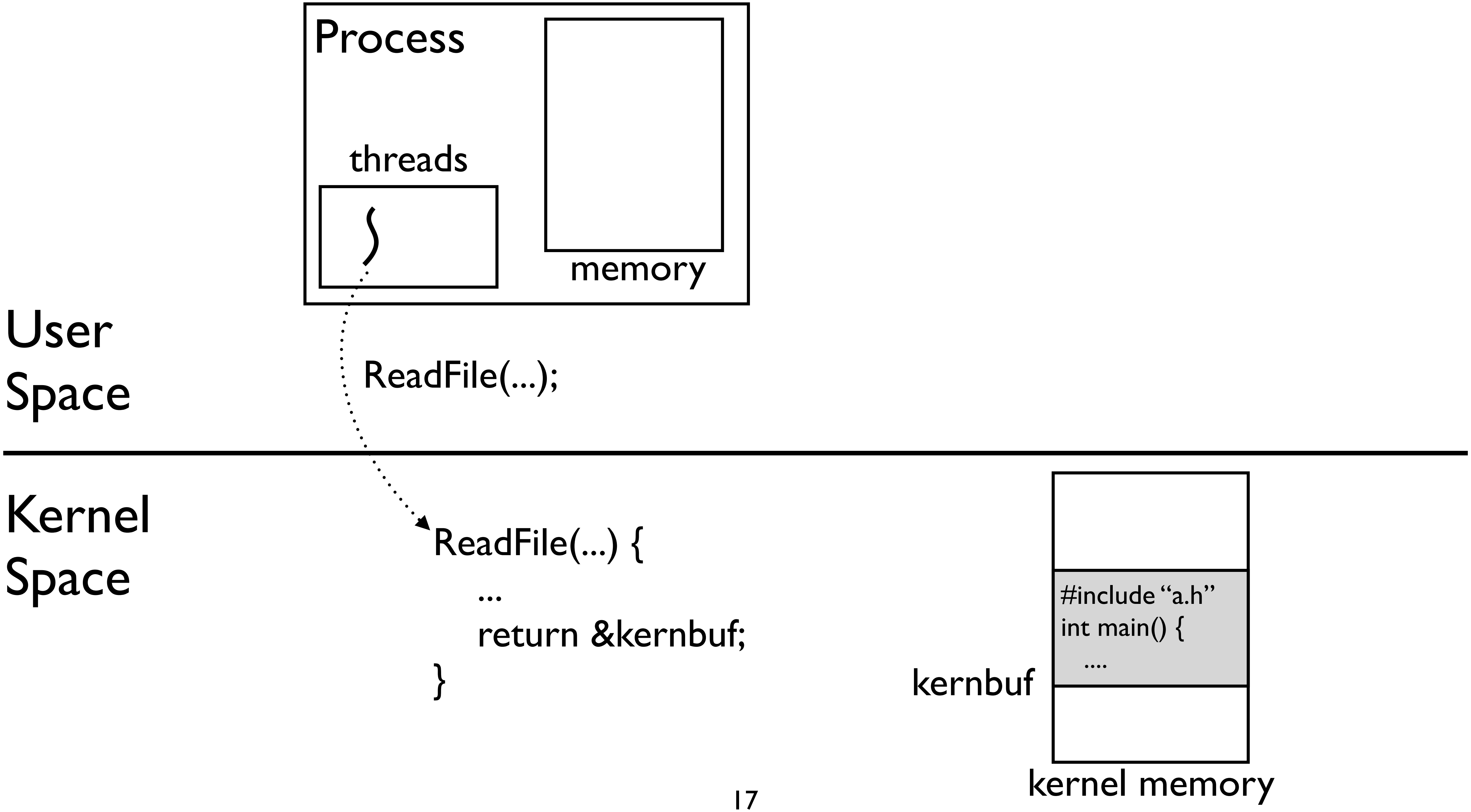
- Interrupts
  - caused by hardware ("asynchronous")
  - e.g.: timer fires, network packet arrives, etc.

- Exceptions
  - caused by software
  - traps: *expected* or *intentional* exceptions

    e.g., making a system call
  - faults: *unexpected* or *error* exceptions

    e.g., segfault (p=NULL; *p)
    divide by zero
    try to execute a privileged instruction in user mode

How do we pass data to/from a system call?

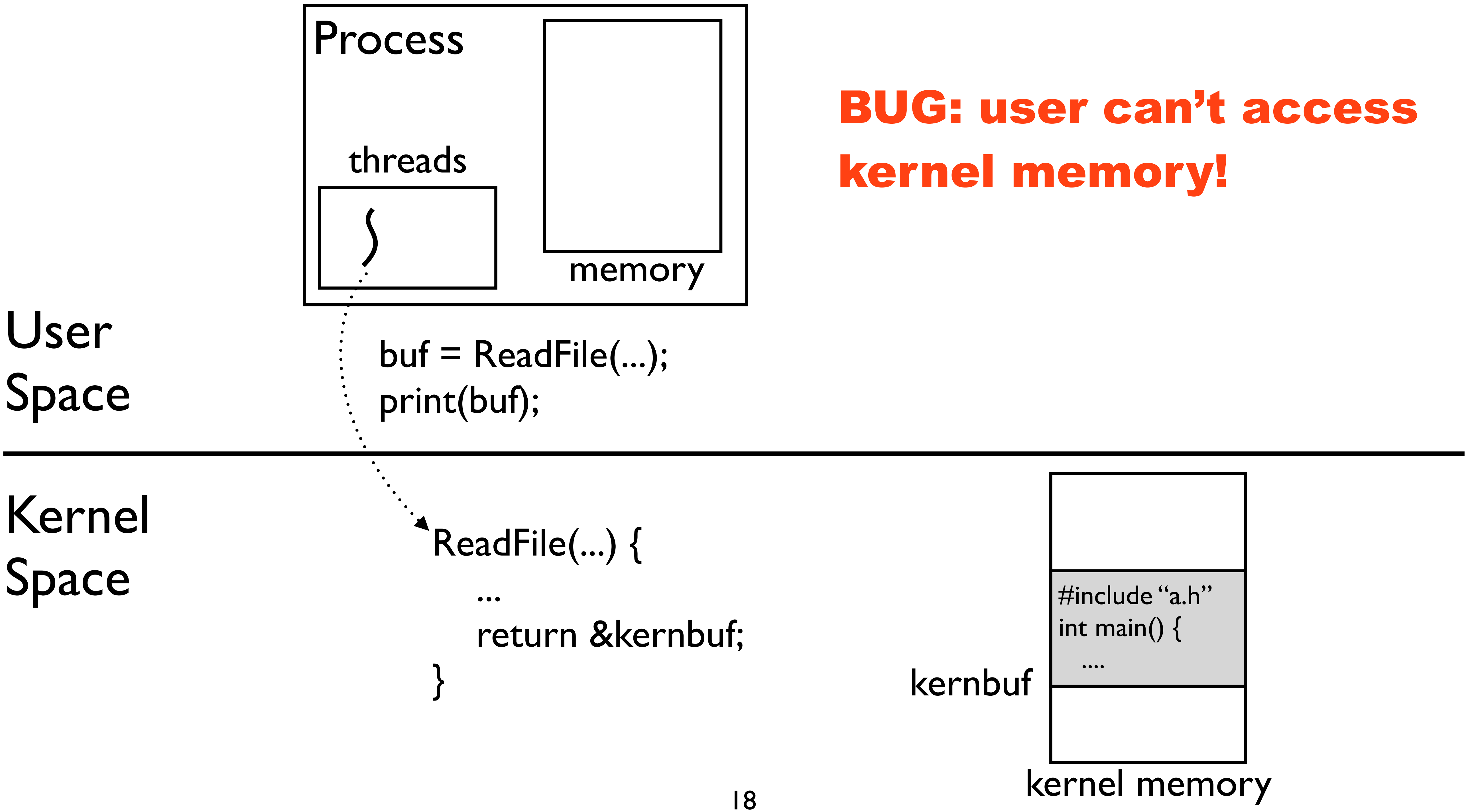


```
ReadFile(...);
```

```
ReadFile(...) {
    ...
    return &kernbuf;
}
```

```
#include "a.h"
int main() {
   ....
```

How do we pass data to/from a system call?



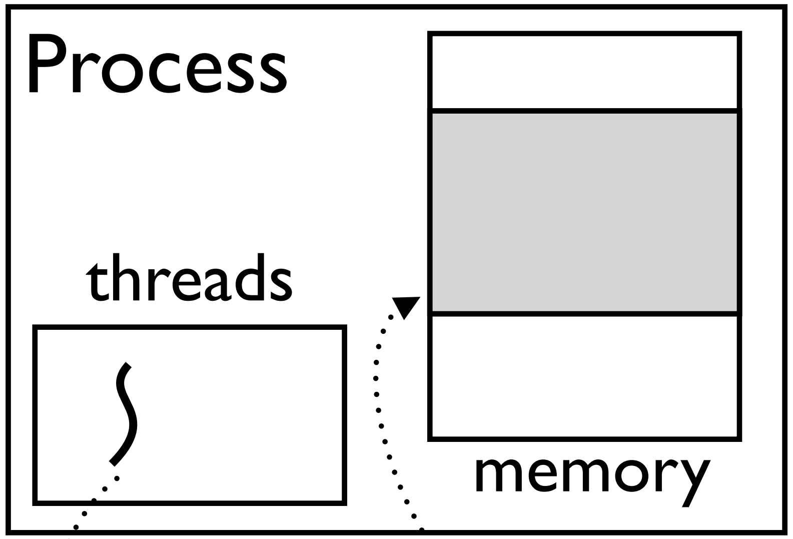
```
buf = ReadFile(...);
print(buf);
```

```
ReadFile(...) {
    ...
    return &kernbuf;
}
```

```
#include "a.h"
int main() {
   ....
```

BUG: user can't access kernel memory!

How do we pass data to/from a system call?



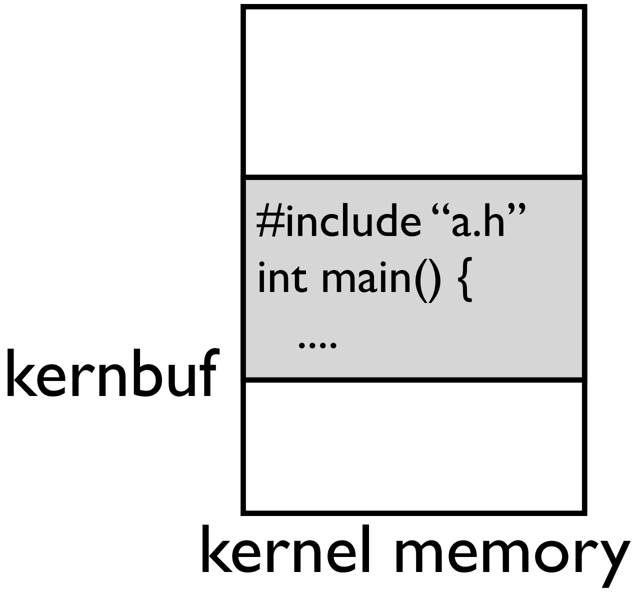
OK: kernel can access user memory

User Space

```
ReadFile(&buf, ...);
print(buf);
```

Kernel Space

```
ReadFile(char* userbuf,  ...) {
    ...
    memcpy(userbuf, kernbuf, sz);
    return;
}
```



How do we pass data to/from a system call?

Evil user program:

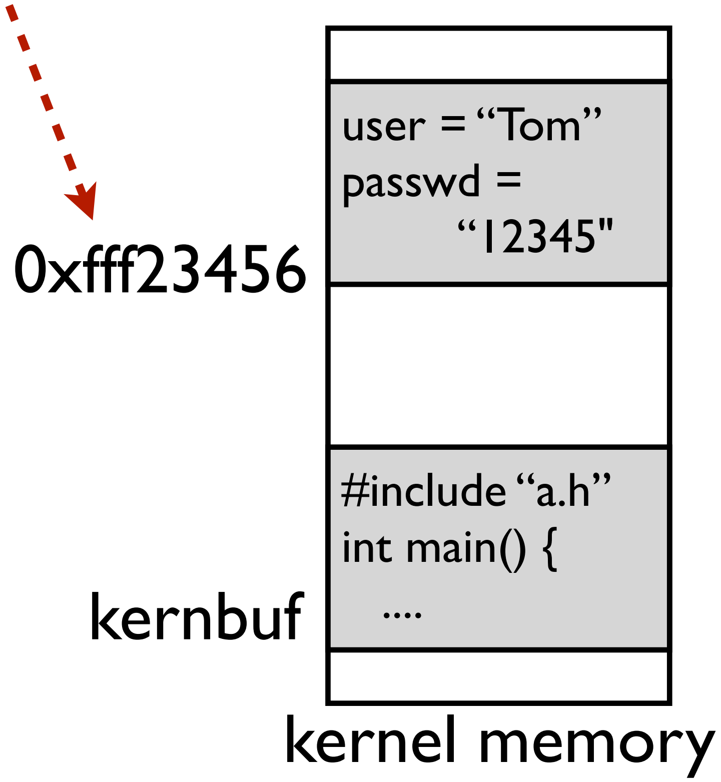
```
ReadFile((char*)0xfff23456, ...);
   // manufacture a buffer ptr
   // hope we get lucky and it points at
   // a kernel data structure!
```

User Space

Kernel Space

```
ReadFile(char* userbuf,
             int userlen) {
   ...
   memcpy(userbuf, kernbuf, sz);
   return;
}
```

Kernel must validate user buffers!



How do we pass data to/from a system call?

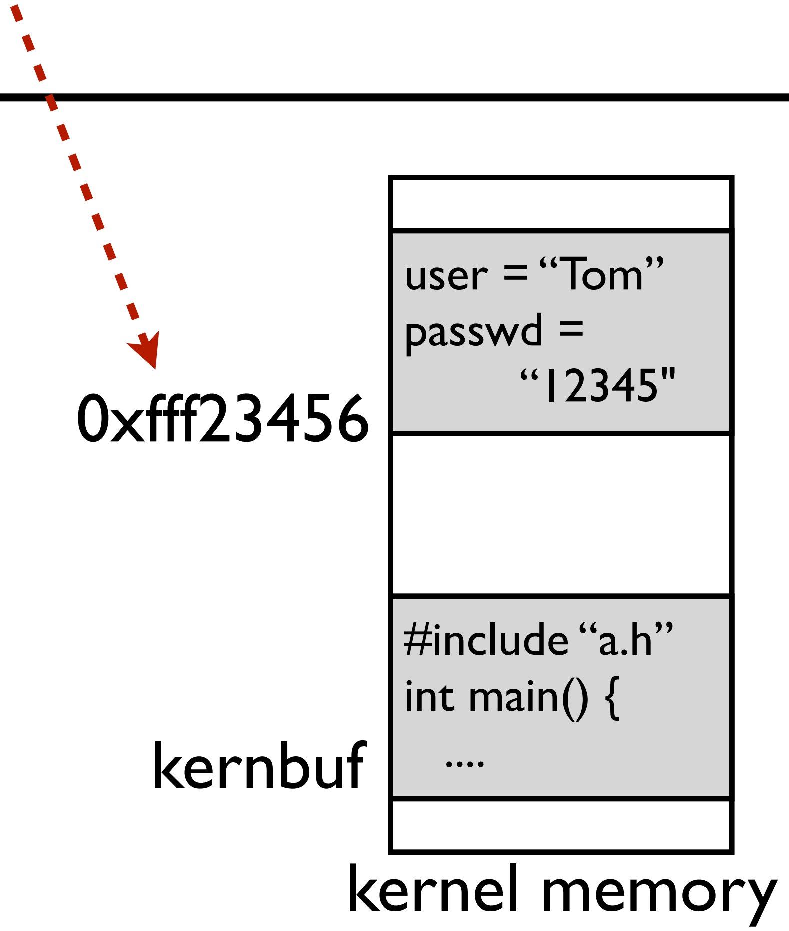
Evil user program:

User Space

```
ReadFile((char*)0xfff23456, ...);
    // manufacture a buffer ptr
    // hope we get lucky and it points at
    // a kernel data structure!
```

Kernel Space

```
ReadFile(char* userbuf,
              int userlen) {
  ...
  ProbeForWrite(userbuf, userlen, ..);
    // fails if userbuf is not valid
    // memory in user space
  ...
  memcpy(userbuf, kernbuf, sz);
  ...
```


0xfff23456
user = "Tom"
passwd =
"12345"
kernbuf
#include "a.h"
int main() {
....
kernel memory