Simple DriverKit Setup

In XCode

- Create new project
- Select macOS > DriverKit Driver (search if needed)
- Supposed to inslude entitlements by default
- **.iig file**; class definition for driver

```
// MyUserUSBInterfaceDriver.iig

public:
    virtual bool init () override;
    virtual kern_return_t Start (IOService *provider) override;
    virtual kern_return_t Stop (IOService *provider) override;
    virtual void free () override;

protected:
    virtual void ReadComplete (OSAction *action,
                               IOReturn  status,
                               uint32_t  actualByteCount,
                               uint64_t  completionTimestamp)
                               TYPE(IOUSBHostPipe::CompleteAsyncIO);
```

kern_return_t Start & Stop; IOKit Lifecycle methods

IOUSBHostPipe;

- Protocol that DriverKit callbacks MUST conform to.
- All instance variables must be allocated by the driver at **time of initialization**

1. **Declare struct to hold all instance vars;**

```
// MyUserUSBInterfaceDriver.cpp Instance Variable Definition

struct MyUserUSBInterfaceDriver_IVars
{
    IOUSBHostInterface        *interface;
    IOUSBHostPipe             *inPipe;
    OSAction                  *ioCompleteCallback;
    IOBufferMemoryDescriptor  *inData;
    uint16_t                   maxPacketSize;
};
```

- Include all instance vars you would've included in the kernel class b4
- **Same USB kernel types** as **Kext**
  - **IOUSBHostInterface** provider
  - **IOUSBHostPipe** object
    - Performs IO
  - **OSAction** object
    - "Encapsulated" Async callbacks (IO)
- **Allocate** at **init**

init "routine";

```
// MyUserUSBInterfaceDriver.cpp init () method

bool
MyUserUSBInterfaceDriver::init ()
{
    bool result = false;

    result = super::init();
    __Require(true == result, Exit);

    ivars = IONewZero(MyUserUSBInterfaceDriver_IVars, 1);
    __Require_Action(NULL != ivars, Exit, result = false);

Exit:
    return result;
}
```

- Calls **init** on **superclass** same as **Kext**
- **Allocates struct**

Start implementation;

```
// MyUserUSBInterfaceDriver.cpp Start () method

kern_return_t
IMPL (MyUserUSBInterfaceDriver,
      Start)
{
    kern_return_t                   ret;
    IOUSBStandardEndpointDescriptors descriptors;

    ret = Start(provider, SUPERDISPATCH);
    __Require(kIOReturnSuccess == ret, Exit);

    ivars->interface = OSDynamicCast(IOUSBHostInterface, provider);
    __Require_Action(NULL != ivars->interface, Exit, ret = kIOReturnNoDevice);
…
```

- Calls into superclass and validates provider
- “Slightly different” then kernel implementation…
- **IMPL;** Required to support **user process** > **kernel proxy object** IPC
- **Start; kext** called **superstart**

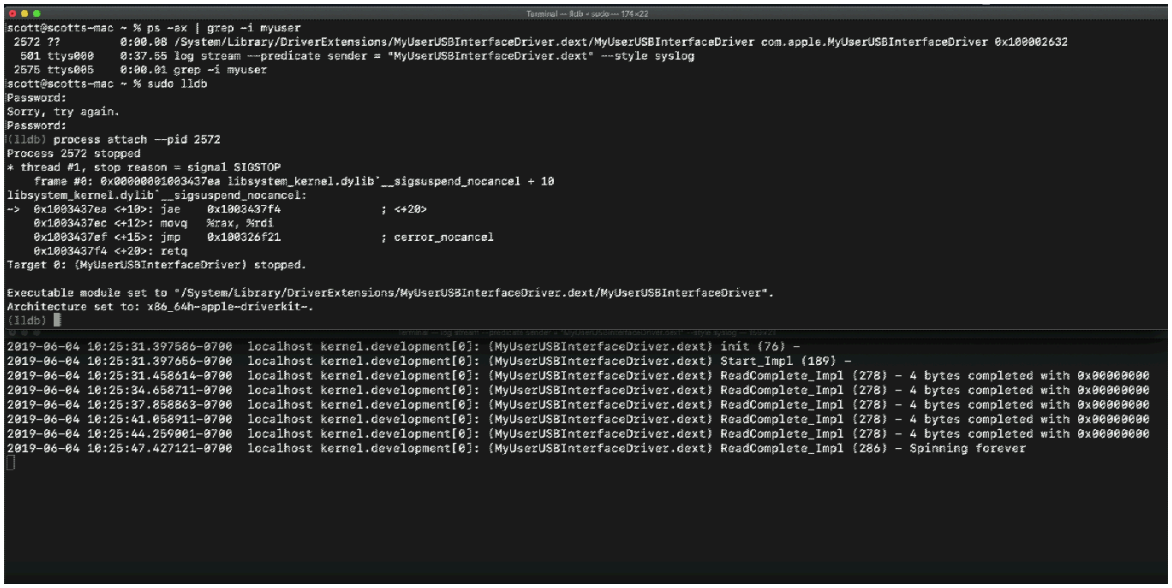
USB DriverKit API;

```
// MyUserUSBInterfaceDriver.cpp Start () continued…


ret = ivars->interface->Open(this, 0, NULL);
__Require(kIOReturnSuccess == ret, Exit);

ret = ivars->interface->CopyPipe(kMyEndpointAddress, &ivars->inPipe);
__Require(kIOReturnSuccess == ret, Exit);

ret = ivars->interface->CreateIOBuffer(kIOMemoryDirectionIn,
                                       ivars->maxPacketSize,
                                       &ivars->inData);
__Require(kIOReturnSuccess == ret, Exit);
```

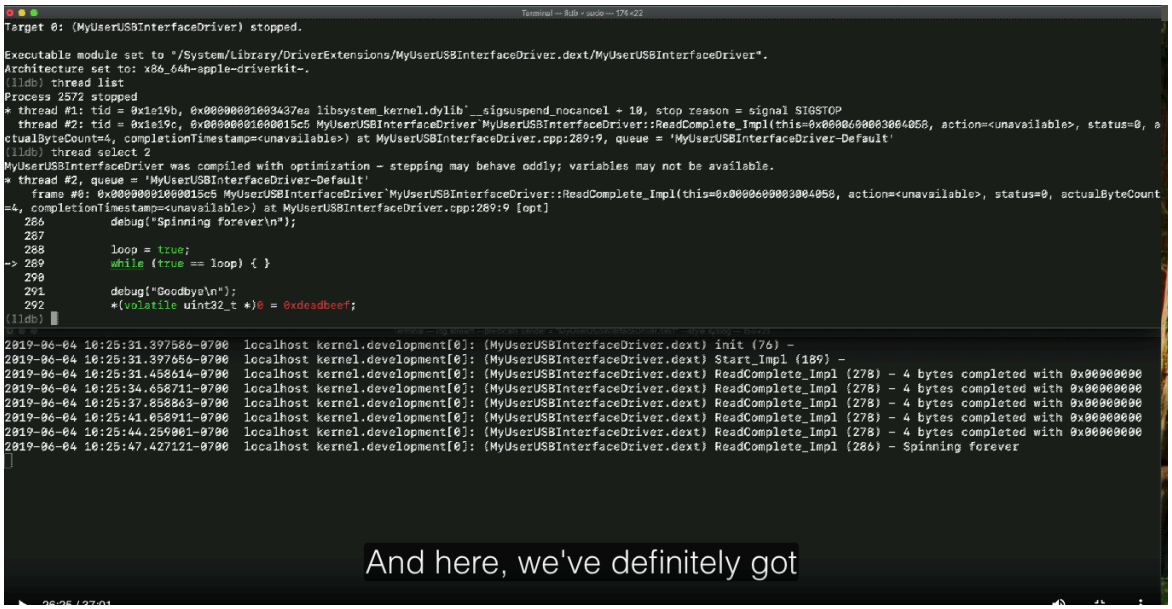
Allocate OS attribute object to encapsulate the callback;

```
// MyUserUSBInterfaceDriver.cpp Start () continued…

ret = OSAction::Create(this,
                       MyUserUSBInterfaceDriver_ReadComplete_ID,
                       IOUSBHostPipe_CompleteAsyncIO_ID,
                       0,
                       &ivars->ioCompleteCallback);
__Require(kIOReturnSuccess == ret, Exit);

ret = ivars->inPipe->AsyncIO(ivars->inData,
                             ivars->maxPacketSize,
                             ivars->ioCompleteCallback,
                             0);
__Require(kIOReturnSuccess == ret, Exit);
```

Witchcraft and wizardry;

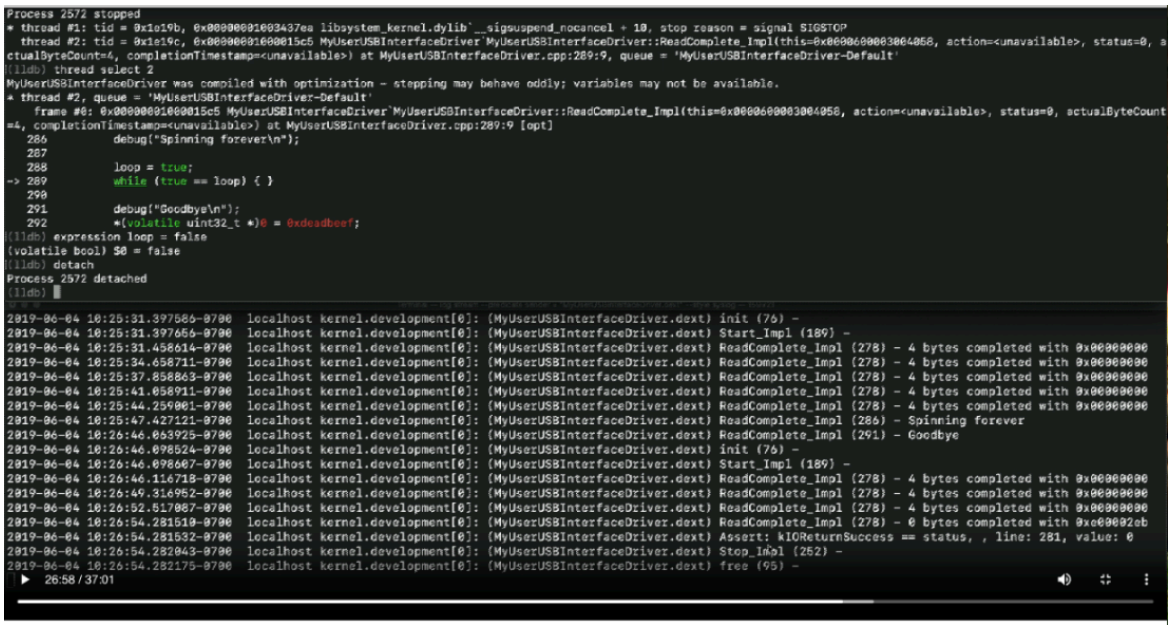


- **ps -ax | grep -i myuser**
  - Shows the driver is running
  - PID is 2572
- **sudo lldb**
  - lldb will show what's happening in the driver
- **process attach —pid 2572**
  - Attach to the target process



- **thread list**
  - Find the thread running the **readComplete** method
  - We can see that it is thread #2
- **thread select 2**
  - Select thread #2
  - Shows an infinite loop is occuring
  - Shows a de-referenced null pointer that would crash the driver
  - Shows the driver crashed but restarted w/out effecting the rest of the system
- **expression loop = false**
  - Modify loop variable
    - Only able to do this in user space not kext
- **detach**
  - detaches lldb from the process

On unplug you can see that your **stop** and **free methods** run as normal



That's all for now!