asynDriver: An Interface Between EPICS Drivers and Device Support

Mark Rivers, Marty Kraimer, Eric Norum
University of Chicago
Advanced Photon Source
References

• This talk is short version of

• asynDriver is available at
What is asyn and why do we need it?

**Motivation**

- Standard EPICS interface between device support and drivers is only loosely defined
- Needed custom device support for each driver
- asyn provides standard interface between device support and device drivers
- And a lot more too!
History – why the name asynDriver

• The initial releases of asynDriver were limited to “asynchronous” devices (e.g. slow devices)
  – Serial
  – GPIB
  – TCP/IP
• asyn provided the thread per port and queuing that this support needs.
• Current version of asynDriver is more general, synchronous (non-blocking) drivers are also supported.
• Device support written as though asynchronous
asynDriver Architecture

Device support (or SNL code, another driver, or non-EPICS software)

asynCommon (connect, report, …) Interfaces (named; pure virtual functions) asynOctet (write, read, setInputEos, …)

Port (named object)

Port driver

addr=0

device

addr=1

device
Control flow – asynchronous driver
Control flow – synchronous driver
asynManager – Methods for drivers

- registerPort
  - Flags for multidevice (addr), canBlock, isAutoConnect
  - Creates thread for each asynchronous port (canBlock=1)
- registerInterface
  - asynCommon, asynOctet, asynInt32, etc.
- registerInterruptSource, interruptStart, interruptEnd
- interposeInterface
- Example code:

```c
status = pasynManager->registerPort(portName,
    ASYN_MULTIDEVICE, /*is multiDevice*/
    1, /* autoconnect */
    0, /* medium priority */
    0); /* default stack size */
status = pasynManager->registerInterface(portName,&pPvt->common);
pasyManager->registerInterruptSource(portName, &pPvt->int32,
    &pPvt->int32InterruptPvt);
```
asynManager – Methods for Device Support

• Create asynUser
• Connect to device, i.e. to port driver
• Queue request for I/O to port
  – asynManager calls callback when port is free
    • Will be separate thread for asynchronous port
  – I/O calls done directly to interface methods in driver
    • e.g. pasynOctet->write()

• Example code:
  /* Create asynUser */
  pasynUser = pasynManager->createAsynUser(processCallback, 0);
  status = pasynManager->connectDevice(pasynUser, pPvt->portName, pPvt->addr);
  pasynInterface = pasynManager->findInterface(pasynUser, asynInt32Type, 1);
  ...
  status = pasynManager->queueRequest(pPvt->pasynUser, 0, 0);
  ...
  status = pPvt->pint32->read(pPvt->int32Pvt, pPvt->pasynUser, &pPvt->value);
asynManager – asynUser

• asynUser data structure. This is the fundamental “handle” used by asyn.

```c
asynUser = pasynManager->createAsynUser(userCallback process, userCallback timeout);
asynUser = pasynManager->duplicateAsynUser(pasynUser, userCallback queue, userCallback timeout);
typedef struct asynUser {
    char *errorMessage;
    int errorMessageSize;
    /* The following must be set by the user */
    double timeout;  /*Timeout for I/O operations*/
    void *userPvt;
    void *userData;
    /*The following is for user to/from driver communication*/
    void *drvUser;
    /*The following is normally set by driver*/
    int reason;
    /* The following are for additional information from method calls */
    int auxStatus;  /*For auxillary status*/
}asynUser;
```
Standard Interfaces

Common interface, all drivers must implement
- `asynCommon`: `report()`, `connect()`, `disconnect()`

I/O Interfaces, most drivers implement one or more
- All have `write()`, `read()`, `registerInteruptUser()` and `cancelInterruptUser()` methods
- `asynOctet`: `writeRaw()`, `readRaw()`, `flush()`, `setInputEos()`, `setOutputEos()`, `getInputEos()`, `getOutputEos()`
- `asynInt32`: `getBounds()`
- `asynInt32Array`
- `asynUInt32Digital`
- `asynFloat64`
- `asynFloat64Array`

Miscellaneous interfaces
- `asynOption`: `setOption()` `getOption()`
- `asynGpib`: `addressCommand()`, `universalCommand()`, `ifc()`, `ren()`, etc.
- `asynDrvUser`: `create()`, `free()`
asynRecord

• New EPICS record that provides access to most features of asyn, including standard I/O interfaces

• Applications:
  – Control tracing (debugging)
  – Connection management
  – Perform interactive I/O

• Very useful for testing, debugging, and actual I/O in many cases
Tracing and Debugging

• Standard mechanism for printing diagnostic messages in device support and drivers
• Messages written using EPICS logging facility, can be sent to stdout, stderr, or to a file.
• Device support and drivers call:
  – asynPrint(pasynUser, reason, format, ...)
  – asynPrintIO(pasynUser, reason, buffer, len, format, ...)
  – Reason:
    • ASYN_TRACE_ERROR
    • ASYN_TRACEIO_DEVICE
    • ASYN_TRACEIO_FILTER
    • ASYN_TRACEIO_DRIVER
    • ASYN_TRACE_FLOW
• Tracing is enabled/disabled for (port/addr)
Current port Drivers

- Unix/Linux/vxWorks/cygwin serial ports
- TCP/IP sockets
- GPIB via National Instruments VME, Ethernet/GPIB devices, Ip488 Industry Pack modules
- VXI-11
- dac128V digital-to-analog (Industry Pack)
- Canberra AIM multi-channel analyzer and ICB modules (Ethernet)
- XIA DXP DSP spectroscopy system (CAMAC, EPP, PXI soon)
- APS quad electrometer (VME). Supports interrupts.
- epid record fast feedback (float 64 with callbacks for input, float64 for output)
- Mca fast-sweep (Int32Array with callbacks)
Fast feedback device support (epid record)

- Supports fast PID control
- Input: any driver that supports asynFloat64 with callbacks (e.g. callback on interrupt)
- Output: any driver that supports asynFloat64.
- In real use at APS for monochromator feedback with IP ADC/DAC, and APS VME beam position monitor and DAC
- >1kHz feedback rate
Summary- Advantages of asynDriver

- Drivers implement standard interfaces that can be accessed from:
  - Multiple record types
  - SNL programs
  - Other drivers
- Generic device support eliminates the need for separate device support in 90% (?) of cases
  - synApps package 10-20% fewer lines of code, 50% fewer files with asyn
- Consistent trace/debugging at (port, addr) level
- asynRecord can be used for testing, debugging, and actual I/O applications
- Easy to add asyn interfaces to existing drivers:
  - Register port, implement interface write(), read() and change debugging output
  - Preserve 90% of driver code
- asyn drivers are actually EPICS-independent. Can be used in any other control system.