Record/Device/Driver Support

Shanghai EPICS Seminar

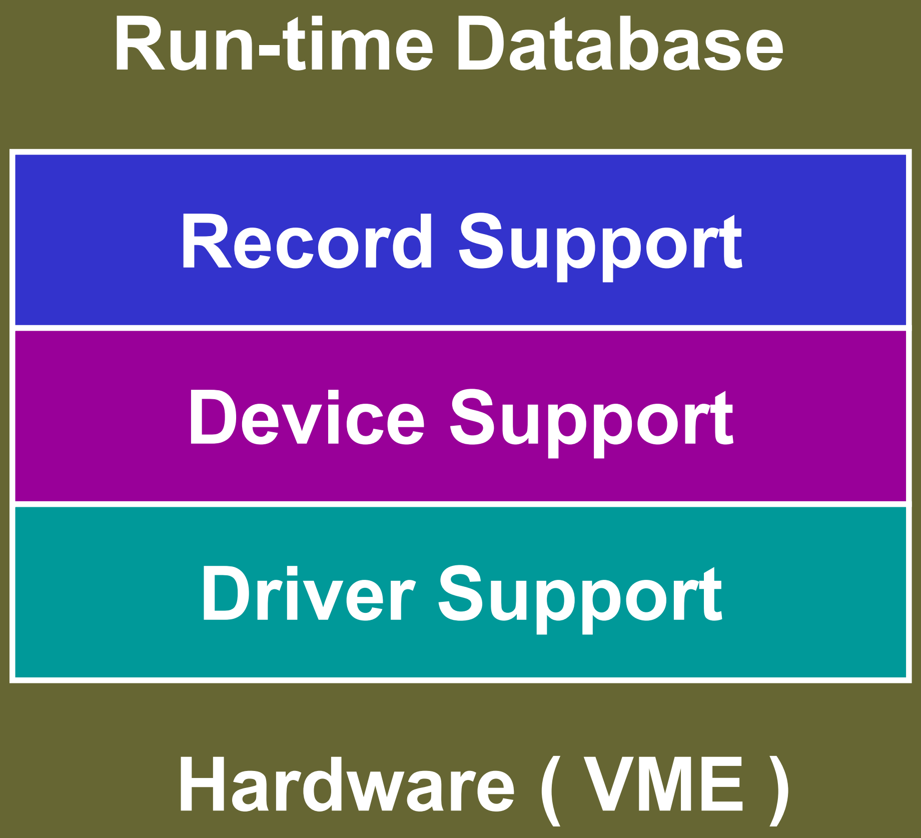
Thursday，8/31

J.Odagiri

Before Getting Started…

- We will not work on any devices…
  - Lots of things to know even without hardware
- Instead, we will work on an example, checking and modifying some source codes.
- Who can remember all details at once?
  - Let me get to focus on essential points
  - Please consult the manual for more details

Overview



Run-time Database

Record Support

Device Support

Driver Support

Hardware (VME)

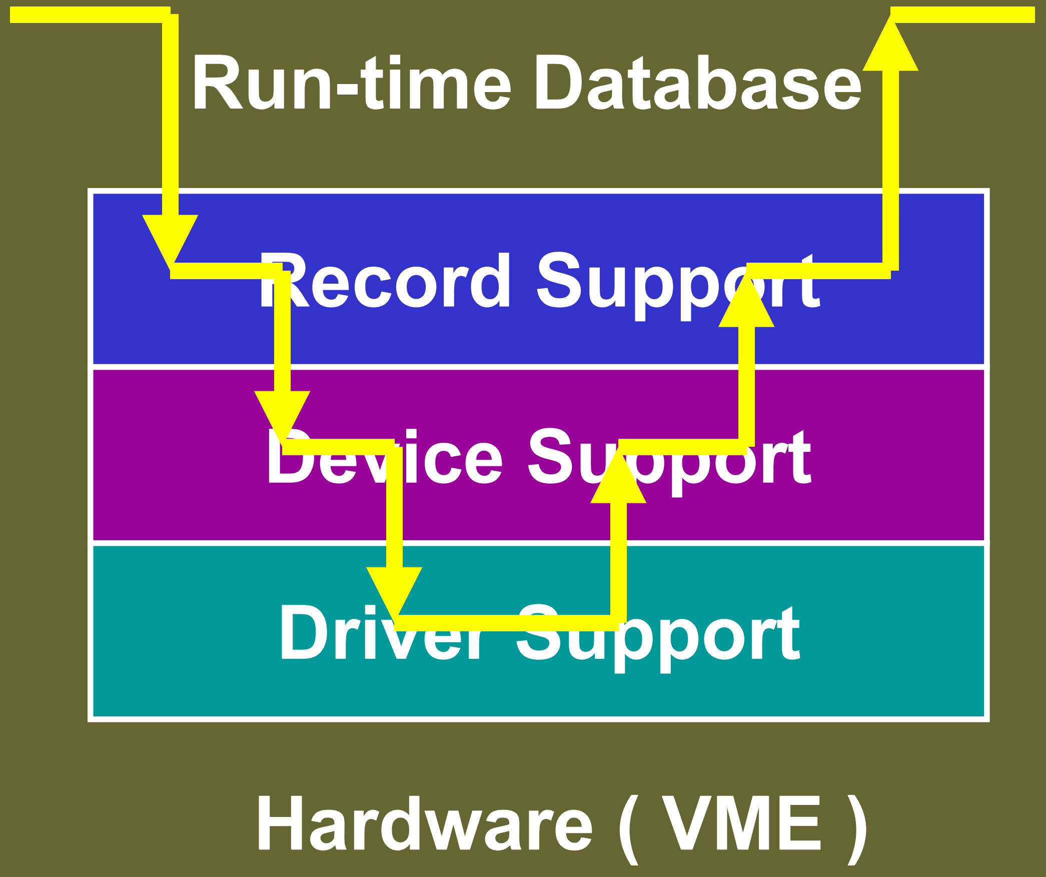
Comments on Record Support

- Record Support consists of a set of routines.
- They can be called from several different tasks:
  - CA_CLIENT task
  - SCAN task
  - CALLBACK task
  - Sequencer task
  - VxWorks shell task …

Comments on Device Support

- Interfaces database records to device drivers or the hardware itself
- Can be divided into two basic classes:
  - Synchronous – for register based devices without delays for I/O (CAMAC)
  - Asynchronous – for devices which can be accessed via I/O requests that may take large amount of time to complete (GPIB)

How Synchronous I/O Works



How Asynchronous I/O Works

- The whole process can be divide into two phases.
- Phase-I
  - Request message to be sent from IOC to the remote device is created and sent
- Phase-II
  - Response message from the remote device is returned to the IOC
  - IOC reads the data in the response message and put it into the database record

More on Asynchronous I/O

- Each of phase-I and phase-II can be completed in no time.
- After a task completed phase-I, it can go ahead to process next record.
- The question is… who takes care of phase-II.
    - Another task in the driver support module should take care of it.
    - The task can invoke phase-II by itself, or get the EPICS callback task to manage phase-II.

More on Asynchronous I/O (continued)

- The delay time between phase-I and phase-II is determined by :
  - Performance of the remote device
  - Transfer rate of the field-bus
  - Not by IOC nor EPICS
- Phase-I is just an initiation of the I/O.
- Phase-II is to execute the steps that a synchronous I/O executes.

Comments on Driver Support

- Why do we need to have two layers of modules, Device and Driver?
- Logically, it is not necessary. The manual says the device support layer was created later by a historical reason.
- But still, better to have two layers when …
  - It is complicated
  - There is an existing driver outside EPICS
  - …

Goals

- Part-I Record/Device support
  - Role and structure of record/device support
  - How they work together to get/put values
  - How to write new record/device support
- Part-II Driver support
  - How to access/probe VME space
  - How to connect interrupts to a handler
  - Basic framework of asynchronous drivers

Part-I
Record/Device Support

- To make the story more concrete, a new record type， rompinRecord was created for this lecture.
- rompinRecord is basically same with longinRecord, except for…
  - Removed many miscellaneous fields and routines
  - Instead, many debug prints inserted

The Sources Are…

- Record support
  - rompinRecord.c
  - rompinRecord.dbd
- Device support
  - devRiSoft.c
  - devRiSoftAsyn.c

rompinRecord.dbd

```
recordtype(rompin) {
        include “dbCommon.dbd”
        field(VAL,DBF_LONG) {
                prompt(“Current value”)
                asl(ASL0)
                pp(TRUE)
        }
                ……
```

dbCommon.dbd

```
field(NAME,DBF_STRING) {
            prompt(“Record Name”)
            special(SPC_NOMOD)
            size(29)
      }
```

Some of Special Values

- SPC_NOMOD
  - The field can not be modified at run-time except by the record/device support modules.
- SPC_DBADDR
  - cvt_dbaddr() should be called when code outside record/device support want to access the field.
- SPC_MOD
  - special() should be called when the field is modified by database access.

rompinRecord.c

- Consists of
  - Record Support Entry Table(RSET)
  - Device Support Entry Table(DSET)
  - Implementations of record support routines defined in the RSET
  - And their forward declarations
  - Internal support routines

Record Support Entry Table

```
struct rset rompinRSET = {
        long              number,
        RECSUPFUN         report,
        RECSUPFUN         init,
        RECSUPFUN         init_record,
        RECSUPFUN         process,
                          ……
    RECSUPFUN  get_alarm_double };
```

Declarations

```
/* Create RSET – Record Support Entry
   Table */
#define       report        NULL
#define       initialize    NULL
static long   init_record();
static long   process();
              . . .
#define       get_alarm_double      NULL
```

Device Support Entry Table (in Record Support)

```
struct  rompindset  {
        long                number;
        DEVSUPFUN     dev_report;
        DEVSUPFUN     init;
        DEVSUPFUN     init_record;
        DEVSUPFUN     get_ioint_info;
        DEVSUPFUN     read_rompin;
};
```

devRiSoft.c

- Software device support to get a value from another record through:
  - Channel Access link
  - Database link
  - Constant link
- If you get the value from hardware, you replace this with, say, devRiMyDevice.c, which is specific to the device.

Device Support Entry Table (in Device Support)

```
struct {
        long                    number;
                ……
        DEVSUPFUN     read_rompin;
} devRiSoft = {
        5,
                ……
        read_rompin,
```

devRiSoftAsyn.c

- Basically, this does the same as devRiSoft does.
- But this emulates asynchronous device support modules for slow message based devices, like GPIB.
- To make the difference clear, the delay time has been set to 3 seconds.

Getting Back to Record Support …

```
/* Create RSET – Record Support Entry
   Table */
#define      report        NULL
#define      initialize    NULL
static long init_record();
static long process();
             . . .
#define      get_alarm_double      NULL
```

process()
Most Important Routine

- Defines and implements the details of “record processing”
- Called by dbProcess(), the database access routine, to process the record
- Calls a device support I/O routine, in many cases

process() Is Responsible For…

- Set record active while it is being processed
- Perform I/O (with aid of device support)
- Check for record specific alarm conditions
- Raise database monitors
- Request processing of forward links

How process() Performs I/O

```
static long process( prompin )
   rompinRecord     *prompin;
{
                ……
   status=readValue(prompin);
                ……
}
```

readValue(): Internal Routine of record Support

```
static long readValue(prompin)
   rompinRecord        *prompin;
{
              ......
   status =
   (*pdset->read_rompin)(prompin);
              ......
}
```

read_rompin() in Device Support

```
static long read_rompin(prompin)
   struct rompinRecord *prompin;
{
                ……
   status =
   dbGetLink(&prompin->inp, … );
                ……
   }
```

process() Is Responsible For…

- Set record active while it is being processed
- Perform I/O (with aid of device support)
- Check for record specific alarm conditions
- Raise database monitors
- Request processing of forward links

How process() Raises Monitors

```
static long process(prompin)
   rompinRecord    *prompin;
{
                ……
   monitor( prompin );
                ……
}
```

monitor(): Internal Routine of record Support

```
static void monitor( prompin )
   rompinRecord        *prompin;
{
   unsigned short  monitor_mask;
                 ……
   if ( monitor_mask ) {
        db_post_events ( prompin, … );
   }
```

db_post_events()
Part of IOC Core

- Create a message to inform the client of the change, and put it on a queue
- Get CA_EVENT task to send the message to the client
- Arguments:
  - The address of the record/field
  - Monitor mask
    - DBE_ALARM - change of alarm state
    - DBE_LOG - change of archive state
    - DBE_VAL - change of value state

CA_CLIENT and CA_EVENT

- CA_CLIENT task invokes dbProcess()
    - dbProcess() calls process()
        - process() calls monitor()
            - monitor() calls db_post_event()
                - db_post_event() puts a message on a queue to inform the client of the change, and notify CA_EVENT that something is in the queue.
- CA_EVENT task picks the message out of the queue and send it back to the client

process() Is Responsible For…

- Set record active while it is being processed
- Perform I/O (with aid of device support)
- Check for record specific alarm conditions
- Raise database monitors
- Request processing of forward links

How process() processes Flink

```
static long process (void *precprd)
{
   rompinRecord *prompin = …
                ......
   recGblFwdLink ( prompin );
                ......
}
```

Global Record Support Routines (base/src/db)

- recGblSetSevr()
- recGblGetGraphicDouble()
- recGblGetAlarmDouble()
- recGblGetControlDouble()
- recGblInitConstantLink()
- recGblResetAlarms()
- recGblFwdLink()
- recGblGetTimeStamp() …

Things to do First

- "Uncomment out" the relevant lines in Makefile.Vx
  - RECTYPES += ../rompinRecord.c
  - SRC.c += ../rompinRecord.c
  - SRC.c += ../devRiSoft.c
  - SRC.c += ../devRiSoftAsyn.c
  - LIBOBJS += rompinRecord.o
  - LIBOBJS += devRiSoft.o
  - LIBOBJS += devRiSoftAsyn.o

Things to do Next

- “Uncomment out” the relevant lines in shanghaiInclude.dbd
  - device(rompin,CONSTANT, devRiSoft,”Soft Channel”)
  - device(rompin,CONSTANT, devRiSoftAsn,”Soft Asyn”)

Making Modules

- Typing “gmake” at src will do it for you.
- The header file, rompinRecord.h, will be also created based on the definitions given in the rompinRecord.dbd.
- After making, please check what you’ve got(the instructors will help you do it).

Testing with IOC

- Modify the startup script, st.cmd2, so as to load the test database (rompin.db)
- Start MEDM and open the display file for the test (rompin.adl)
- Boot the IOC with the modified startup script (st.cmd2)
- Have a fun for a while…

PACT

```
static long process( prompin )
{
              ......
   unsigned char        pact=prompin->pact;
              ......
   status = readValue( prompin );
   if ( !pact && prompin->pact )           retrun(
   0 );
```

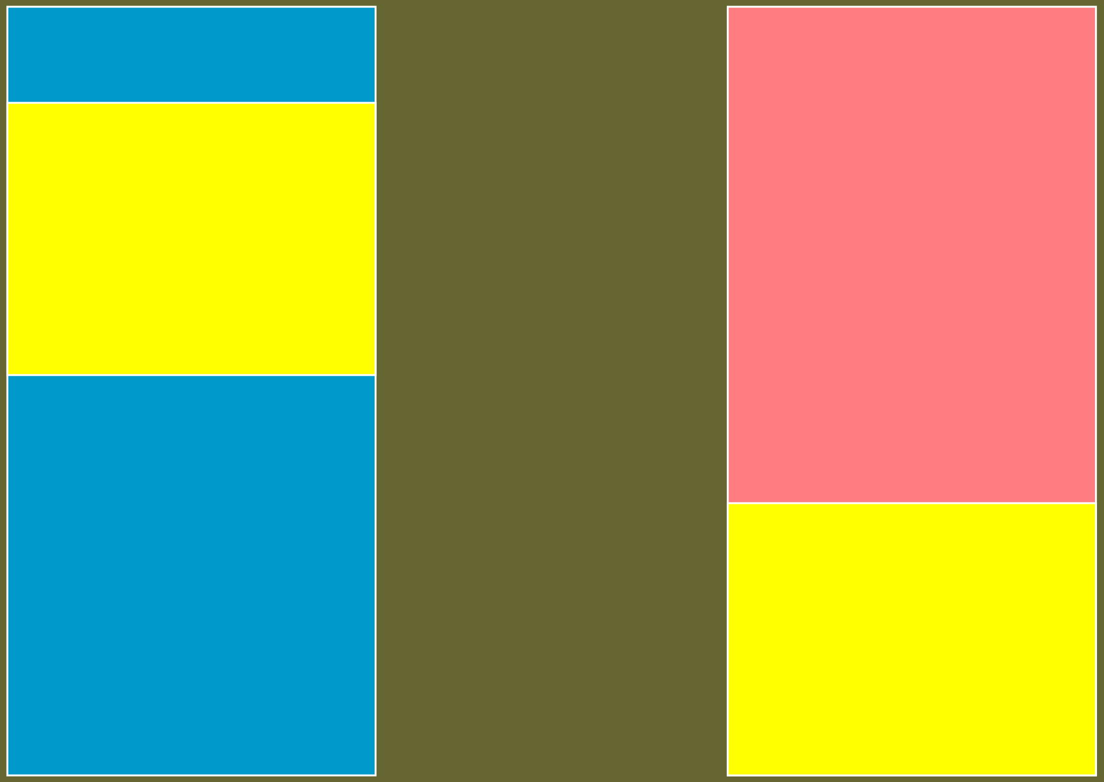
More on PACT

- PACT == TRUE means the record is active.
- Before dbProcess() calls process(), it checks if PACT is FALSE (and the record is not disabled).
- Asynchronous completion routines in record support modules call process() without checking PACT.

Part-I I
Driver Support

- How to access/probe VME space
- How to connect interrupts to a handler
- Other common techniques to implement device drivers

CPU local address space and VME spaces



CPU local

VME bus

sysBusToLocalAdrs()
A VxWorks(BSP) function

- convert a bus address to a local address

```
STATUS sysBusToLocalAdrs(
        int     adrsSpace;
        char *  busAdrs;
        char ** pLocalAdrs;
   )
```

vxMemProbe()
A VxWorks(BSP) function

- probe an address for a bus error

```
STATUS vxMemProbe(
        char * Adrs;
        int       mode;
        int    length;
        char * pVal;
   )
```

intConnect()
A VxWorks(BSP) function

- connect a C routine to a hardware interrupt

```
STATUS intConnect(
        VOIDFUNCPTR * vector;
        VOIDFUNCPTR   routine;
        int           paramerter;
   )
```

sysIntEnable()
A VxWorks(BSP) function

- enable a bus interrupt level

```
STATIS sysIntEnable(
        int  intLevel;
)
```

Binary Semaphores

- SemBCreate()
  - Crate and initialize a binary semaphore
- semTake()
  - If empty, the caller goes to sleep.
- semGive()
  - If another task calls this, the sleeping task wakes up.

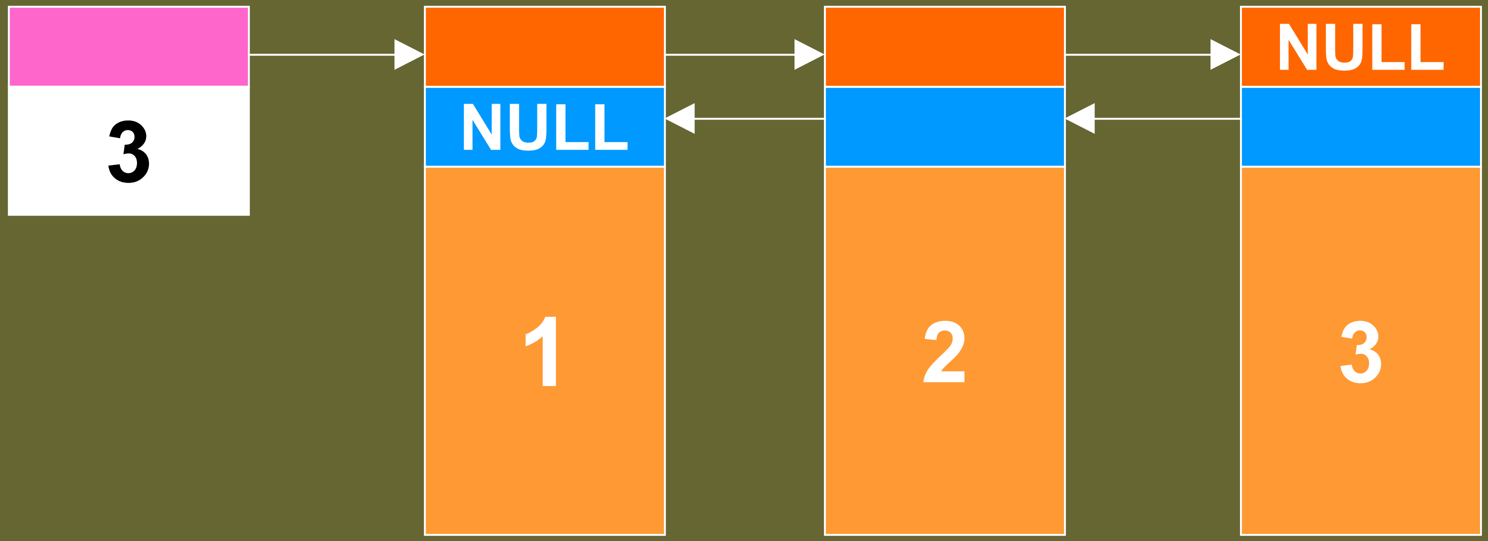
Notification of Events

```
void print_task()
{
    while( TRUE )
    {
        semTake( intSem, … );
        printf( “got the intterrupt   ” );
    }
}
VOIDFUNCPTR int_handler()
{
    semGive( intSem );
}
```

Mutual-exclusion (Mutex) Semaphores

- Binary semaphores can be used for mutual-exclusion.
- But, VxWorks offers another type of semaphores which specialize in mutex.
  - Priority inversion safe
  - Allows the owner to take recursively
  - Only the owner can give it.

Linked Lists



Linked List Library

- lstInit()   [ellInit()   ]
- lstAdd()   [ellAdd()   ]
- lstGet()   [ellGet()   ]
- lstCount() [ellCount()]
- lstFirst() [ellFirst()]
- lstNext()   [ellNext()   ]
- lstInsert() [ellInsert()]
- …

Mutex for Linked List

```
void some_task()
{
    while( TRUE )
    {
                    …
          semTake( mutexSem, … );
          ellGet( queue );
          semGive( mutexSem );
                    …
}
```

Watchdog Timers

- wdCreate()
  - Crate a watchdog timer
- wdStart()
  - Start a watchdog timer
- wdCancel()
  - Cancel a currently counting watchdog
- wdDelete()
  - Delete a watchdog timer

driverAsyn.c

- A sample code which shows you how to use semaphores and linked list libraries.
  - Create and initialize linked lists
  - Create and initialize semaphores
  - Spawn a task which manages requests
  - Has a simplest interrupt handler

Practices

- Check how PACT works… again.
- In process() of rompinRecord,
  - Comment out monitor() and see what happens.
  - Comment out recGblFwdLink() and make sure that forward link does not work.
- Modify rompinRecord so that MEDM can make the graphic display nicely.
- Modify rompinRecord so that it can raise alarms.

If you have time left…

- Compile driverAsyn.c and see how it works.
- When you test it, you are supposed to work on behalf of the iocCore and the hardware…