EPICS Simple device support

Simion startet with it on

https://gitlab.fhi.mpg.de/beinlich/epics-devsupportfromexample.git

This guide explains how to implement interrupt-driven device support for longin (Long Integer Input) records in EPICS, including background tasks and proper cleanup with exit hooks.

Longin record device support with I/O Intr scanning allows records to be processed asynchronously when data arrives, rather than periodic scanning. This is ideal for hardware interrupts, network events, or any asynchronous data source.

Key Components

1. **Device Support Structure** - Standard EPICS device support entry table
2. **Interrupt Callback** - Function called when data arrives
3. **Background Task** - Thread that monitors hardware/network
4. **Exit Hook** - Cleanup function for shutdown
5. **IOSCANPVT** - EPICS interrupt scan mechanism

Prerequisites

Load Simion's pre-work from FHI gitlab

```
cd ~/EPICS/FHI
git clone https://gitlab.fhi.mpg.de/beinlich/epics-devsupportfromexample.git
cd epics-devsupportfromexample
git checkout -b add_intr_heinz
```

Now add I/O Intr Support. See git diff.

```
git commit -a -m'add I/O Intr support'
git push --set-upstream origin add_intr_heinz
```

Device Support Interrupt/Callbacks/Background task (example longin)

This guide explains how to implement interrupt-driven device support for longin (Long Integer Input)

records in EPICS, including background tasks and proper cleanup with exit hooks.

Overview

Longin record device support with I/O Intr scanning allows records to be processed asynchronously when data arrives, rather than periodic scanning. This is ideal for hardware interrupts, network events, or any asynchronous data source.

Key Components

1. **Device Support Structure** - Standard EPICS device support entry table
2. **Interrupt Callback** - Function called when data arrives
3. **Background Task** - Thread that monitors hardware/network
4. **Exit Hook** - Cleanup function for shutdown
5. **IOSCANPVT** - EPICS interrupt scan mechanism

Basic Device Support Structure

```
#include <epicsExport.h>
#include <devSup.h>
#include <longinRecord.h>
#include <callback.h>
#include <dbScan.h>
#include <epicsMutex.h>
#include <epicsThread.h>
#include <epicsExit.h>
#include <alarm.h>
#include <recGbl.h>

typedef struct {
    IOSCANPVT ioscanpvt;
    epicsMutexId lock;
    epicsInt32 value;
    int dataReady;
    int connected;
} devLonginPvt;

/* Forward declarations */
static long init_record(longinRecord *prec);
static long get_ioint_info(int cmd, longinRecord *prec, IOSCANPVT *ppvt);
static long read_longin(longinRecord *prec);

/* Device Support Entry Table */
struct {
    long number;
    DEVSUPFUN report;
    DEVSUPFUN init;
    DEVSUPFUN init_record;
    DEVSUPFUN get_ioint_info;
    DEVSUPFUN read_longin;
} devLonginMyDevice = {
    5,
    NULL,
    NULL,
    init_record,
    get_ioint_info,
    read_longin
};

epicsExportAddress(dset, devLonginMyDevice);
```

Initialize Record

```
static long init_record(longinRecord *prec)
{
    devLonginPvt *pPvt;

    /* Allocate private structure */
    pPvt = (devLonginPvt *)calloc(1, sizeof(devLonginPvt));
    if (!pPvt) {
        recGblRecordError(S_dev_noMemory, (void *)prec,
                          "devLonginMyDevice: init_record() malloc failed");
        return S_dev_noMemory;
    }

    /* Create mutex for thread-safe access */
    pPvt->lock = epicsMutexCreate();
    if (!pPvt->lock) {
        free(pPvt);
        recGblRecordError(S_dev_noMemory, (void *)prec,
                          "devLonginMyDevice: epicsMutexCreate failed");
        return S_dev_noMemory;
    }

    /* Create IOSCANPVT for interrupt scanning */
    scanIoInit(&pPvt->ioscanpvt);

    /* Store private data in record's dpvt field */
    prec->dpvt = pPvt;

    pPvt->value = 0;
    pPvt->dataReady = 0;
    pPvt->connected = 0;

    /* Register this device instance */
    registerDevice(pPvt);

    return 0;
}
```

Get I/O Interrupt Info

This function is called by EPICS when the record is set to I/O Intr scan mode:

```
static long get_ioint_info(int cmd, longinRecord *prec, IOSCANPVT *ppvt)
{
    devLonginPvt *pPvt = (devLonginPvt *)prec->dpvt;

    if (!pPvt) {
        return -1;
    }

    *ppvt = pPvt->ioscanpvt;
    return 0;
}
```

Read Function

This function is called when the record processes (triggered by interrupt):

```
static long read_longin(longinRecord *prec)
{
    devLonginPvt *pPvt = (devLonginPvt *)prec->dpvt;

    if (!pPvt) {
        recGblSetSevr(prec, READ_ALARM, INVALID_ALARM);
        return -1;
    }

    /* Lock mutex for thread-safe access */
    epicsMutexLock(pPvt->lock);

    if (!pPvt->connected) {
        epicsMutexUnlock(pPvt->lock);
        recGblSetSevr(prec, COMM_ALARM, INVALID_ALARM);
        return -1;
    }

    if (pPvt->dataReady) {
        /* Copy data to record */
        prec->val = pPvt->value;

        pPvt->dataReady = 0;
        epicsMutexUnlock(pPvt->lock);
        return 0;
    } else {
        epicsMutexUnlock(pPvt->lock);
        recGblSetSevr(prec, READ_ALARM, INVALID_ALARM);
        return -1;
    }
}
```

Background Task Implementation

Global Variables for Background Task

```
static epicsThreadId backgroundTaskId = NULL;
static int backgroundTaskShouldExit = 0;
static epicsMutexId deviceListLock = NULL;

/* List to track all device instances */
typedef struct deviceNode {
    devLonginPvt *pPvt;
    struct deviceNode *next;
} deviceNode;

static deviceNode *deviceList = NULL;
```

Register Device Function

```
static void registerDevice(devLonginPvt *pPvt)
{
    deviceNode *newNode;
    static int firstTime = 1;

    /* Create device list lock on first call */
    if (deviceListLock == NULL) {
        deviceListLock = epicsMutexCreate();
    }

    /* Add this device to the list */
    newNode = (deviceNode *)malloc(sizeof(deviceNode));
    if (newNode) {
        newNode->pPvt = pPvt;

        epicsMutexLock(deviceListLock);
        newNode->next = deviceList;
        deviceList = newNode;
        epicsMutexUnlock(deviceListLock);
    }

    /* Start background task if not already running */
    if (backgroundTaskId == NULL) {
        backgroundTaskShouldExit = 0;
        backgroundTaskId = epicsThreadCreate("devLonginBackground",
                                             epicsThreadPriorityMedium,
                                             epicsThreadGetStackSize(epicsThreadSta
                                             backgroundTask,
                                             NULL);

        if (backgroundTaskId == NULL) {
            printf("Failed to create background task\n");
        }
    }

    /* Register exit hook on first call */
    if (firstTime) {
        epicsAtExit(exitHook, NULL);
        firstTime = 0;
    }
}
```

Background Task Function

```
static void backgroundTask(void *arg)
{
    deviceNode *node;
    int counter = 0;

    printf("Longin background task started\n");

    while (!backgroundTaskShouldExit) {
```

```
        /* Simulate reading data from hardware/network */
        /* In real implementation, this might be:
         * - Reading from ADC
         * - Polling hardware register
         * - Waiting on interrupt
         * - Reading from socket
         * - Monitoring GPIO
         */

        epicsThreadSleep(1.0); /* 1 second interval */
        counter++;

        /* Lock device list */
        epicsMutexLock(deviceListLock);

        /* Iterate through all registered devices */
        for (node = deviceList; node != NULL; node = node->next) {
            devLonginPvt *pPvt = node->pPvt;
            epicsInt32 newValue;

            /* Simulate hardware read - replace with actual hardware access */
            if (readHardwareValue(&newValue) == 0) {
                /* Lock this device's data */
                epicsMutexLock(pPvt->lock);

                /* Store new value */
                pPvt->value = newValue;
                pPvt->dataReady = 1;
                pPvt->connected = 1;

                epicsMutexUnlock(pPvt->lock);

                /* Trigger record processing via I/O Intr */
                scanIoRequest(pPvt->ioscanpvt);
            } else {
                /* Mark as disconnected on read error */
                epicsMutexLock(pPvt->lock);
                pPvt->connected = 0;
                epicsMutexUnlock(pPvt->lock);

                /* Still trigger processing to update alarm state */
                scanIoRequest(pPvt->ioscanpvt);
            }
        }

        epicsMutexUnlock(deviceListLock);
    }

    printf("Longin background task exiting\n");
}

/* Placeholder hardware read function */
static int readHardwareValue(epicsInt32 *value)
```

```
{
    static int counter = 0;

    /* Replace this with actual hardware read:
     * - Read from ADC via SPI/I2C
     * - Read from network socket
     * - Read from shared memory
     * - Read hardware register
     * - etc.
     */

    *value = counter++;
    return 0; /* 0 = success, -1 = error */
}
```

Alternative: Event-Driven Background Task

For hardware that generates actual interrupts or has blocking I/O:

```
#include <sys/select.h>

static void backgroundTaskEventDriven(void *arg)
{
    fd_set readfds;
    struct timeval timeout;
    int maxfd;
    deviceNode *node;

    printf("Event-driven background task started\n");

    while (!backgroundTaskShouldExit) {
        FD_ZERO(&readfds);
        maxfd = -1;

        /* Add file descriptors to monitor */
        /* This could be sockets, device files, pipes, etc. */
        int hardwareFd = getHardwareFileDescriptor();
        if (hardwareFd >= 0) {
            FD_SET(hardwareFd, &readfds);
            if (hardwareFd > maxfd) maxfd = hardwareFd;
        }

        /* Timeout allows periodic check of exit flag */
        timeout.tv_sec = 1;
        timeout.tv_usec = 0;

        int ret = select(maxfd + 1, &readfds, NULL, NULL, &timeout);

        if (ret > 0) {
            /* Data available */
            if (FD_ISSET(hardwareFd, &readfds)) {
```

```
                epicsInt32 newValue;

                if (readFromHardware(hardwareFd, &newValue) == 0) {
                    /* Update all devices */
                    epicsMutexLock(deviceListLock);

                    for (node = deviceList; node != NULL; node = node->next) {
                        devLonginPvt *pPvt = node->pPvt;

                        epicsMutexLock(pPvt->lock);
                        pPvt->value = newValue;
                        pPvt->dataReady = 1;
                        pPvt->connected = 1;
                        epicsMutexUnlock(pPvt->lock);

                        scanIoRequest(pPvt->ioscanpvt);
                    }

                    epicsMutexUnlock(deviceListLock);
                }
            }
        } else if (ret < 0) {
            /* Error - mark all devices as disconnected */
            epicsMutexLock(deviceListLock);
            for (node = deviceList; node != NULL; node = node->next) {
                epicsMutexLock(node->pPvt->lock);
                node->pPvt->connected = 0;
                epicsMutexUnlock(node->pPvt->lock);
                scanIoRequest(node->pPvt->ioscanpvt);
            }
            epicsMutexUnlock(deviceListLock);

            epicsThreadSleep(1.0);
        }
    }

    printf("Event-driven background task exiting\n");
}
```

Multi-Channel Hardware Example

For hardware with multiple channels:

```
typedef struct {
    IOSCANPVT ioscanpvt;
    epicsMutexId lock;
    epicsInt32 value;
    int dataReady;
    int connected;
    int channel;  /* Hardware channel number */
} devLonginPvt;
```

```
/* Parse INP field for channel number */
static long init_record(longinRecord *prec)
{
    devLonginPvt *pPvt;
    int channel;

    /* Parse INP field like "@CHAN0" or "@CHAN1" */
    if (sscanf(prec->inp.value.instio.string, "@CHAN%d", &channel) != 1) {
        recGblRecordError(S_dev_badInpType, (void *)prec,
                          "devLonginMyDevice: Invalid INP field");
        return S_dev_badInpType;
    }

    if (channel < 0 || channel >= MAX_CHANNELS) {
        recGblRecordError(S_dev_badInpType, (void *)prec,
                          "devLonginMyDevice: Channel out of range");
        return S_dev_badInpType;
    }

    pPvt = (devLonginPvt *)calloc(1, sizeof(devLonginPvt));
    if (!pPvt) {
        return S_dev_noMemory;
    }

    pPvt->lock = epicsMutexCreate();
    if (!pPvt->lock) {
        free(pPvt);
        return S_dev_noMemory;
    }

    scanIoInit(&pPvt->ioscanpvt);
    pPvt->channel = channel;
    prec->dpvt = pPvt;

    registerDevice(pPvt);

    return 0;
}

/* Background task reads all channels */
static void backgroundTaskMultiChannel(void *arg)
{
    deviceNode *node;
    epicsInt32 channelValues[MAX_CHANNELS];

    while (!backgroundTaskShouldExit) {
        /* Read all channels at once */
        if (readAllChannels(channelValues, MAX_CHANNELS) == 0) {
            epicsMutexLock(deviceListLock);

            /* Update each device with its channel's value */
            for (node = deviceList; node != NULL; node = node->next) {
```

```
                devLonginPvt *pPvt = node->pPvt;

                epicsMutexLock(pPvt->lock);
                pPvt->value = channelValues[pPvt->channel];
                pPvt->dataReady = 1;
                pPvt->connected = 1;
                epicsMutexUnlock(pPvt->lock);

                scanIoRequest(pPvt->ioscanpvt);
            }

            epicsMutexUnlock(deviceListLock);
        }

        epicsThreadSleep(0.1);
    }
}
```

Exit Hook for Cleanup

```
static void exitHook(void *arg)
{
    deviceNode *node, *nextNode;

    printf("Exit hook called - cleaning up longin devices\n");

    /* Signal background task to exit */
    backgroundTaskShouldExit = 1;

    /* Wait for background task to finish */
    if (backgroundTaskId != NULL) {
        epicsThreadSleep(0.5); /* Give thread time to exit */
        backgroundTaskId = NULL;
    }

    /* Clean up hardware resources */
    shutdownHardware();

    /* Clean up device list */
    if (deviceListLock) {
        epicsMutexLock(deviceListLock);
    }

    node = deviceList;
    while (node != NULL) {
        nextNode = node->next;

        /* Clean up device private data */
        if (node->pPvt) {
            if (node->pPvt->lock) {
                epicsMutexDestroy(node->pPvt->lock);
            }
            free(node->pPvt);
        }

        free(node);
        node = nextNode;
    }

    deviceList = NULL;

    if (deviceListLock) {
        epicsMutexUnlock(deviceListLock);
        epicsMutexDestroy(deviceListLock);
        deviceListLock = NULL;
    }

    printf("Exit hook cleanup complete\n");
}
```

Complete Example: ADC Reader

Here's a complete example for reading from an ADC:

```
#include <stdint.h>

/* Simulated ADC hardware interface */
#define ADC_BASE_ADDR 0x43C00000
#define ADC_DATA_REG  0x00
#define ADC_CTRL_REG  0x04
#define ADC_STAT_REG  0x08

static volatile uint32_t *adcBase = NULL;

static int initHardware(void)
{
    /* Map hardware registers (platform-specific) */
    /* This is just an example - actual implementation depends on hardware */

    printf("Initializing ADC hardware\n");

    /* On embedded Linux, might use /dev/mem and mmap() */
    /* On RTEMS, direct memory access */
    /* On simulation, use malloc() */

    adcBase = (volatile uint32_t *)ADC_BASE_ADDR;

    /* Configure ADC */
    adcBase[ADC_CTRL_REG/4] = 0x01; /* Enable ADC */

    return 0;
}

static void shutdownHardware(void)
{
    if (adcBase) {
        adcBase[ADC_CTRL_REG/4] = 0x00; /* Disable ADC */
        printf("ADC hardware shutdown\n");
    }
}

static int readHardwareValue(epicsInt32 *value)
{
    if (!adcBase) {
        return -1;
    }

    /* Check if data ready */
    uint32_t status = adcBase[ADC_STAT_REG/4];
    if (!(status & 0x01)) {
        return -1; /* No data available */
    }

    /* Read ADC value */
    *value = (epicsInt32)adcBase[ADC_DATA_REG/4];
```

```
    return 0;
}

static void backgroundTask(void *arg)
{
    deviceNode *node;

    /* Initialize hardware */
    if (initHardware() != 0) {
        printf("Failed to initialize hardware\n");
        return;
    }

    printf("ADC background task started\n");

    while (!backgroundTaskShouldExit) {
        epicsInt32 adcValue;

        /* Try to read from ADC */
        if (readHardwareValue(&adcValue) == 0) {
            /* Update all devices */
            epicsMutexLock(deviceListLock);

            for (node = deviceList; node != NULL; node = node->next) {
                devLonginPvt *pPvt = node->pPvt;

                epicsMutexLock(pPvt->lock);
                pPvt->value = adcValue;
                pPvt->dataReady = 1;
                pPvt->connected = 1;
                epicsMutexUnlock(pPvt->lock);

                scanIoRequest(pPvt->ioscanpvt);
            }

            epicsMutexUnlock(deviceListLock);
        }

        epicsThreadSleep(0.01); /* 10ms polling */
    }

    shutdownHardware();
    printf("ADC background task exiting\n");
}
```

Database Configuration

```
record(longin, "$(P)$(R)Value") {
    field(DTYP, "My Device")
    field(INP,  "@CHAN0")
    field(SCAN, "I/O Intr")
    field(EGU,  "counts")
    field(HOPR, "4095")
    field(LOPR, "0")
}

record(longin, "$(P)$(R)Count") {
    field(DTYP, "My Device")
    field(INP,  "@COUNTER")
    field(SCAN, "I/O Intr")
}
```

Advanced: Rate Limiting

To prevent flooding the system with too many interrupts:

```
typedef struct {
    IOSCANPVT ioscanpvt;
    epicsMutexId lock;
    epicsInt32 value;
    int dataReady;
    int connected;
    epicsTimeStamp lastUpdate;
    double minUpdateInterval; /* Minimum seconds between updates */
} devLonginPvt;

static void updateDevice(devLonginPvt *pPvt, epicsInt32 newValue)
{
    epicsTimeStamp now;
    double elapsed;

    epicsTimeGetCurrent(&now);

    epicsMutexLock(pPvt->lock);

    /* Calculate time since last update */
    elapsed = epicsTimeDiffInSeconds(&now, &pPvt->lastUpdate);

    /* Only update if enough time has passed */
    if (elapsed >= pPvt->minUpdateInterval) {
        pPvt->value = newValue;
        pPvt->dataReady = 1;
        pPvt->lastUpdate = now;

        epicsMutexUnlock(pPvt->lock);

        /* Trigger processing */
        scanIoRequest(pPvt->ioscanpvt);
    } else {
        /* Update value but don't trigger processing yet */
        pPvt->value = newValue;
        epicsMutexUnlock(pPvt->lock);
    }
}
```

Testing

```
# Start IOC
./st.cmd

# Monitor values
camonitor PV:Value

# Check device support
dbior

# Test with high-rate updates
# The I/O Intr mechanism should handle this efficiently
```

Performance Monitoring

Add diagnostic counters:

```
typedef struct {
    IOSCANPVT ioscanpvt;
    epicsMutexId lock;
    epicsInt32 value;
    int dataReady;
    int connected;

    /* Diagnostics */
    unsigned long updateCount;
    unsigned long errorCount;
    epicsTimeStamp firstUpdate;
    epicsTimeStamp lastUpdate;
} devLonginPvt;

/* Add diagnostic records */
record(longin, "$(P)$(R)UpdateCount") {
    field(DTYP, "My Device Stats")
    field(INP,  "@UPDATE_COUNT")
    field(SCAN, "1 second")
}
```

Key Points

1. **Thread Safety**: Always protect shared data with `epicsMutex`
2. **scanIoRequest()**: Call to trigger I/O Intr record processing
3. **One IOSCANPVT per device**: Each record instance needs its own
4. **Connection Status**: Track and report hardware connectivity
5. **Exit Hooks**: Essential for clean shutdown
6. **Error Handling**: Set alarms appropriately on communication errors
7. **Efficiency**: Single background task for all instances

Common Pitfalls

- Not calling `scanIoRequest()` - records never process
- Race conditions from missing mutex locks
- Memory leaks from not freeing resources in exit hook
- Creating too many threads - use one shared background task
- Not handling hardware disconnection properly
- Forgetting to initialize IOSCANPVT with `scanIoInit()`

Resources

- EPICS Application Developer's Guide - Device Support chapter
- `longinRecord.h` - Record definition
- `dbScan.h` - I/O Interrupt scanning API
- Example device supports in EPICS base