EPICS: PORTING iocCORE TO MULTIPLE OPERATING SYSTEMS

M. Kraimer, Advanced Photon Source, Argonne National Laboratory, Argonne, IL USA

Abstract

An important component of EPICS (Experimental Physics and Industrial Control System) is iocCore, which is the core software in the IOC (input/output controller) front-end processors. Currently iocCore requires the vxWorks operating system. This paper describes the porting of iocCore to other operating systems.

1 INTRODUCTION

Originally IOC (input/output controller) meant a VME/VXI-based system that interfaced to various hardware interfaces. Today it is also used on non-VME systems as well as software-only systems. It does, however, still require the vxWorks operating system. The goal is to remove the dependency on vxWorks.

The IOC software can be divided into the following categories:

- vxWorks - A proprietary real-time operating system
- iocCore - Core EPICS software, described below
- Hardware support - Support for specific hardware

An EPICS IOC allows extensible record/device/driver support, i.e., there is a clear separation between iocCore and hardware support. Thus iocCore can be used without hardware support and/or support for non-VME based hardware. Beginning with the 3.14 releases, iocCore will be implemented via Operating System Independent (OSI) libraries.

iocCore includes the following components:

- Database locking, scanning, and processing
- Channel access client and server
- Standard record types and soft device support
- Access security
- Other non-hardware specific components

The port is based on the following assumptions:

- All hardware support will be built separately. Thus it does not need to be ported.
- iocCore requires a multithreaded environment.

OSI components are defined such that:

- The vxWorks implementation has minimal overhead compared to vxWorks specific calls.
- The components can be implemented via a combination of POSIX.1, POSIX.4 (POSIX Real Time), and POSIX Threads (pthreads).

For components that require a different implementation for different environments, the implementation may be via header and/or source files as long as user code can code to the “prototype” header files.

2 OVERVIEW OF CHANGES

2.1 Replacements for Existing vxWorks and EPICS Components

The following OSI libraries replace vxWorks libraries.

- osiClock - tickLib, sysLib
- osiFindGlobalSymbol, registry - symFindByName
- osiInterrupt - intLib
- osiRing - rngLib
- osiSem - semLib
- osiThread - taskLib
- osiWatchdog - wdLib

Each osiXXX interface defines only the functions needed by iocCore rather than all the features of the vxWorks libraries.

2.2 Registry

vxWorks provides a function, symFindByName, that is used to dynamically locate global data and functions. This facility is unique to vxWorks and is not easily recreated in other environments. Instead a facility is provided to register and find pointers to functions and structures. This leaves the problem of registering everything currently located via calls to symFindByName. This is solved via a Perl script that generates a C function, which registers the record, device, and driver support. dbLoadDatabase calls this function after loading the database.

2.3 Build Environment

The build environment is different. The principal features are:

- Each source directory has a single Makefile. This builds for both the host and for all IOC targets.
- The new configuration files are located in base/configure.
- The existing base/config is still present so that existing applications still build without major changes.

2.4 task_params.h

This file, which defines vxWorks-specific options for iocCore tasks is no longer part of iocCore. Instead osiThread provides generic options.

2.5 vxWorks Shell

If the target is not for vxWorks, the vxWorks target shell is not available. iocInit, dbLoadRecords, etc. must be called directly by main or the equivalent. The vxWorks debugging environment is not present although a nicer one using xgdb may be available.

2.6 Interrupt Level

The vxWorks intLock/intUnlock routines are an essential part of base. For example any code, including interrupt routines, can call callbackRequest. Most operating systems do not allow such tight coupling between interrupt routines and user processes. osiInterrupt is provided to solve this problem. For operating systems like vxWorks, in which everything runs in a shared memory, multithreaded kernel environment, an implementation of osiThread must be provided. For other operating systems, e.g., winNT, Unix, Linux, a generic version is provided. The generic version uses a global lock, where global means global to the process.

3 STATUS OF PORT

3.1 Work Completed

- All code except Channel Access, Sequencer, and vxWorks-dependent device and driver support has been converted to use the new libraries.
- The registry has been implemented.
- The example generated by makeBaseApp has been successfully tested on vxWorks.
- A separate subdirectory base/src/vxWorks has been created and all vxWorks-specific code moved to this subdirectory. This makes it possible for existing vxWorks IOC applications to use the new system with only minor changes to the applications.

3.2 Work Remaining

- Implement osiSem and osiThread for other platforms. If the implementation is done via POSIX (including POSIX Real Time and POSIX Threads) then many platforms can be supported. William Lupton (KECK) has already developed an alpha version.
- Convert Channel Access (client and server) to use OSI calls. This will be done by Jeff Hill (LANL).
- Convert the sequencer to use OSI calls. William Lupton has already implemented an alpha version. Also the sequencer will be unbundled from base.
- Resolve problems about single thread vs multi threaded environments.
- TEST TEST TEST

4 PROTOTYPE DEFINITIONS

For a particular operating system, each function may be implemented as desired, but the final result must appear to user code like the definitions in this section. For example, functions can be implemented via macros defined in a header file that replace the generic header file.

For each OSI definition, a vxWorks-specific version is available that causes no or almost no performance loss vs direct vxWorks calls. For all except osiSem and osiThread, a generic version is supplied. osiSem and osiThread must be implemented for each operating environment. These can be implemented via POSIX (including POSIX Real Time and POSIX Threads). Since only osiSem and osiThread must be implemented for each operating system, they are the only libraries that will be discussed here.

4.1 osiSem

```
typedef void *semId;
typedef enum {
  semTakeOK,semTakeTimeout
} semTakeStatus;
typedef enum {
    semEmpty,semFull
} semInitialState;

semId semBinaryCreate(
    int initialState);
void semBinaryDestroy(semId id);
void semBinaryGive(semId id);
semTakeStatus semBinaryTake(semId id);
void semBinaryTakeAssert(semId id);
semTakeStatus semBinaryTakeTimeout(
    semId id, double timeOut);
semTakeStatus semBinaryTakeNoWait(semId id);
void semBinaryFlush(semId id);

semId semMutexCreate(void);
void semMutexDestroy(semId id);
void semMutexGive(semId id);
semTakeStatus semMutexTake(semId id);
void semMutexTakeAssert(semId id);
semTakeStatus semMutexTakeTimeout(
    semId id, double timeOut);
semTakeStatus semMutexTakeNoWait(
    semId id);
void semMutexFlush(semId id);
```

Mutual exclusion semaphores

- Must implement recursive locking.
- Should implement priority inheritance and be deletion safe.

For POSIX

- Binary can be implemented easily as a condition variable.
- Mutex can be implemented via various POSIX facilities. Takes careful thought. A pthread mutex is not sufficient.

For vxWorks:

- the entire implementation of Binary and Mutex is via macros in a vxWorks specific header file.

On a single-threaded environment

- Mutex is implemented as though the caller always has access to the resource.
- Binary issues an error message and terminates if an attempt is made to create an instance.

4.2 osiThread

```
#define threadPriorityMax 99
#define threadPriorityMin 0

/*some generic values */
#define threadPriorityLow 10
#define threadPriorityMedium 50
#define threadPriorityHigh 90

/*some iocCore specific values */
#define threadPriorityChannelAccessClient 10
#define threadPriorityChannelAccessServer 20
#define threadPriorityScanLow 60
#define threadPriorityScanHigh 70
int threadGetOsiPriorityValue(
    int ossPriority);
int threadGetOssPriorityValue(
    int osiPriority);

typedef enum {
 threadStackSmall,
 threadStackMedium, threadStackBig
} threadStackSizeClass;

unsigned int threadGetStackSize(
    threadStackSizeClass size);

typedef void *threadId;
threadId threadCreate(
    const char *name,
    unsigned int priority,
    unsigned int stackSize,
    THREADFUNC funptr,void *parm);
void threadDestroy(threadId id);
void threadSuspend(threadId id);
void threadResume(threadId id);
int threadGetPriority(threadId id);
void threadSetPriority(
    threadId id,int priority);
void threadSetDestroySafe(
    threadId id);
void threadSetDestroyUnsafe(
    threadId id);
const char *threadGetName(
    threadId id);
int threadIsEqual(
    threadId id1, threadId id2);
int threadIsReady(threadId id);
int threadIsSuspended(threadId id);
void threadSleep(double seconds);
threadId threadGetIdSelf(void);
void threadLockContextSwitch(void);
void threadUnlockContextSwitch(void);
threadId threadNameToId(
    const char *name);
```

Thread priorities are assigned a value from 0 to 99. A higher value means higher priority.

threadGetStackSize can be called to get one of three default sizes. This should be done whenever possible. Code can specify any size it desires, but such code is not portable.

5 REGISTRY

iocCore currently uses symFindByName to dynamically bind the following:

- record/device/driver support
  The registration facility provides a type safe and easy to use alternative to symFindByName.
- subroutine record subroutines
  An easy to use solution must be developed.
- initHooks
  A new implementation of initHooks is now provided. It provides a routine initHookRegister. This MUST be called by any routine that wants to be called during initialization.
- drvTS.c
  This has been moved to base/src/vxWorks. Thus for now it is only supported on vxWorks
- Other hardware or vxWorks-dependent code.

Thus only the first two items need a solution.

The basic idea is to provide a registration facility. Any storage meant to be “globally” accessible must be registered before it can be accessed by other code.

A Perl script is provided that reads the xxxApp.dbd file and produces a C file containing a routine registerRecordDeviceDriver, which registers all record/device/driver support defined in the xxxApp.dbd file.

Functions are provided to register (registryADD) and find (registryFind) a void pointer. Using these functions, typesafe functions are provided to register:

- record types,
- device support, and
- driver support.

6 ACKNOWLEDGMENTS

The changes to the EPICS build system for the iocCore port have been made by Janet Anderson (APS) and Jeff Hill (LANL). Janet made the final major set of changes that were needed. Jeff had previously created operating system indepenent libraries for several EPICS components used on workstations.

This work is supported by the the U.S. Department of Energy, Office of Basic Energy Sciences, under Contract No. W-31-109-ENG-38.

7 REFERENCES

A list of EPICS documentation can be found at: http://csg.lbl.gov/EPICS/RecommendedDocs.html