Porting Complex Unix Applications to a deeply embedded Forth Environment

Jeremy Moller, Desmond Ward, George Redpath

Controlled Electronic Management Systems Ltd

Introduction

CEM Systems is one of the leading designers of complex Access Control Systems world wide. For many years large sections of the system had been written on PCs using Unix. With continuous efforts at reducing the cost it was decided to port the major control element from Unix to an embedded forth environment.

This process started approximately 4 years ago and the first products are now being shipped. The internal code for the product family to be developed is the 90X0 and covers a range of devices. The device described in this paper is the first commercially available device the 9040

Application Firmware

This section gives a brief overview of the S9040 application software.

System Hardware Architecture

Please refer to Figure 1

The S9040 connects to a central database computer (**CDC**) via a 10 Base-T Ethernet connection. All card data and personnel data entry is performed at the CDC. A copy of the database is held on the S9040, which is continuously updated in near real time.

A maximum of 64 remote devices can be connected to the S9040 by four RS485 serial ports. The S9040 must be able to generate and maintain offline card databases for these devices, respond to card swipe queries from these devices, and forward any card swipe transactions and alarm data from the remote devices onto the CDC.

Application Firmware

The S9040 Application can be divided into 5 main task types.

1. TCP/IP, Ethernet communications
2. Remote device communications
3. Generation and maintenance of remote device offline database
4. Forwarding of transaction and alarm data to the CDC
5. General housekeeping

Remote Device Communication

The S9040 communicates with a maximum of 64 remote devices over a serial link using our specific CEM Multidrop protocol. The remote devices operate as master devices contacting the S9040 when necessary. Each device has it's own unique address. All remote devices use a defined set of poll characters to indicate their current state and the preferred action from the S9040. The S9040 acts on the various poll characters to download device configuration, device database, device database updates, time update or receive offline transaction/alarm data.

In addition to this periodic communication with the remote devices, the S9040 must be able to respond to card swipe queries, buffer both transaction and alarm details for further processing. All card swipe queries include processing of Access Level, Time zone and Holiday permissions decisions together with verifying anti pass back permissions. Typical card swipe query response time is around 35mS, but must not be greater than 400mS. In addition, the S9040 must forward

broadcast commands issued from the CDC onto the remote devices.

Information about the communications between the S9040 and the remote devices are logged to disk in real time for diagnostic purposes.

Remote database operation

The S9040 must be able to generate a device configuration and database for each remote device connected to the S9040. This data is generated from the local database held on the S9040 which supports a maximum of 1000 Access Levels and Timezones in addition to holiday definitions.

A remote device or user command are used to initiate the database build mechanism and the S9040 can service four separate devices, one on each serial port, at any one time. The time taken to build a device database depends on the number of records for that particular remote device but as an estimate, a 70,000 card database can be built in approximately 120 seconds.

In addition to performing a database build for any remote devices, the S9040 also continuously maintains the local database. Database changes received from the CDC are actioned immediately. Any changes effecting a remote device database are buffered and transmitted to the remote device when it is ready to receive the updates.

Transaction Processing

All card swipe outcome transactions and alarm transactions are buffered for prompt transmission to the CDC. If the link to the central database computer is down then the transactions are stored on disk in the S9040 and transmitted to the CDC whenever the link has been restored. All transactions are time stamped and tagged with the point of origin.

Housekeeping

A number of background tasks are performed at periodic intervals. These include checking the integrity of the program (held in RAM), checking the size of the diagnostic log files, testing the state of the TCP/IP sockets link, and synchronising the disk cache.

Forth environment

Multi tasking scheduler

Our experience over the years with the UNIX based system, which this is to replace, was that some polled events could be missed, as the respective process was not able to run at the required time. This was usually solved by throwing large quantities of memory and a faster processor at the problem.

The design of the new controller had to avoid processes running excessively late, soft real time design as against hard real time design would suffice for this particular application.

The MPE FORTH development system comes with a multi task scheduler built in. This is a co-operative scheduler, like UNIX each task will run in turn until it relinquishes control back to the scheduler. The scheduler then passes on control to the next process to be run. There is a danger with this system, in that a task may need a large amount of processing for a period. This then has a knock on affect with other processes by making them late. This situation could be avoided by carefully designing software loops etc.

It was decided to take this a step further by adding dynamic time slicing to the scheduler. All processes are initially given a fixed time window to run in. If a process for some reason used up more than this time window, the scheduler would automatically switch context to the next process to be run. The process that had

timed out is now given a shorter time window to run the next time the context passes on to it. This will continue until the process finally voluntarily relinquishes control back to the scheduler.

The up side of implement time slicing is that the system runs much smoother even when heavily loaded and polled events aren't missed.

There is a down side however, greater use has to be made of the semaphore mechanisms in the scheduler. There is a real chance of two or more processes requiring access to the same resource at the same time and causing possible corruption of data.

Forth TCP/IP

Several components are needed to build up a system and the Ethernet TCP/IP stack is one of the vital components that enable us to implement our system. It gives us 10Mbit access to an Intranet or even Internet for an embedded controller. The protocol stack is capable of working across routers and gateways should that be necessary.

The protocol stack supplied by MPE Ltd provides what is known as the BSD socket layer which provided an API similar to that available with UNIX and Windows.

The socket layer provides a good platform for implementing several of the standard protocols necessary to interface to the rest of the world. These include an the following network applications;

- CEM-FTP server/client, for transferring data files between the controlling host PC and the embedded controller similar to the known FTP.
- HTTP server, for remote graphical monitoring and administration of the embedded controller by the use of dynamic HTML web pages.
- SMTP client, for posting alert email messages.
- BOOTP client, for remotely configuring the unit's IP address.
- TELNET server, for remote diagnostics.

CEM-FTP

It is important for the project that is made possible to be able to transfer file entities between the controller and the host. This application protocol provides most of the functions that are expected of FTP. We had to implement our own version of FTP, as the version of the TCP/IP stack we had at the time, did not support TCP client sockets. The application allows a client to put a file, get a file, obtain a directory listing and change the current working directory. This is closely associated with the capabilities of the embedded controller, (refer to the section on the file system).

HTTP server

The HTTP web server is an important enhancement to system. By using so called server side includes it is possible for the server to dynamically change the contents of a web page, e.g. the voltage and temperature display of a charging battery pack. By using a combination of client side Java and Javascript we are also able to dynamically update the contents of the page without the user intervention. This is sometimes known as "Push technology".

The biggest problem with web pages is how to integrate these with a ROM image. We do have one advantage in the we have a file system, but in fact decided to integrate most of the web pages into the ROM image. A pre-compilation process is used to convert the various web files to a Forth compatible file with an index table to help the server find the location of each web file in the ROM image.

SMTP client

If the controller has access to an email server, why not use email to notify some

one of a problem? With this simple email client it is possible to connect to the email server and email an alert message, e.g. to a GSM mobile phone via a SMS gateway.

BOOTP client

With our experience of networking embedded devices over many years, we know that installers often have difficulty sitting four jumper links correctly, so how would they manage setting a four field number, the IP address, correctly.

To alleviate this problem it was decided to implement the BOOTP protocol. With this the installer has to note the unit's Ethernet address, which is unique for every Ethernet based device, enter that into the central server and then the server automatically allocates the device an IP address appropriate for it.

File system

In order to implement an embedded copy of the UNIX based application system, a file system is vital constituent part. The controller needs to be able to handle over 70,000 card personnel records and other associated data. In a reasonably stripped down format, this will require storage space of around 6Mb, split across many data files of varying sizes. These data files are dynamic, records can be changed, added or deleted. Different users will have different data sizes so using a fixed format file system could be restrictive and most likely wasteful of space. Also to be considered is that fact that it would be useful if the files could be grouped in appropriate directories.

Being an embedded unit, it is considered advantageous that there are no moving parts, so a standard IDE hard disk was out of the question. The SANDISK compact flash modules can use a IDE type interface and are available in sizes from 8Mb to over 96Mb. Other manufacture's compact flash modules were tried and rejected as their performance was pitifully slow, SANDISK two blocks written in 3mS, other module 2 blocks write in 48mS. This is apparently caused by the internal software

The maximum number of write cycles is over 100,000 and the modules have intelligence to automatically handle failing flash cells. These therefore provide an ideal solution for the embedded file system requirements.

After using the system for a while it became apparent that even with a block write time of 1.5 ms data caching was necessary.

Disk format

The disk format was another consideration. The options include:

- a MS-DOS compatible file system. It is considered wasteful of space and relatively slow.
- a UNIX like file system. This was chosen, as it is quite efficient in space usage and fast.

There may be a requirement to allow the disk to be removed from the embedded platform and used in a DOS PC in order to allow the transfer or recover data. It was felt that the efficient space usage and speed was probably more important.

This format of the disk is divided into 5 sections, the boot area, the root area, the disk usage area, the file descriptor area and the data area.

The boot area can be used to allow a boot loader in ROM to find and load the application off the file system.

The root area contains the directory listing for the files and directories in the root directory.

The disk usage area contains a bitmap of every sector on the disk. It performs the vital function of tracking the usage of each block on the disk. One bit relates to one disk block, usually 512 bytes. It is used

when a new file is created, an old file is deleted or a file is extended.

The file descriptor area contains the information about every file and directory located on the disk. It contains information about the size of a file, the locations of the file, the type of file or directory and the date it was last modified, amounts other things.

Finally there is the data area, which contains all the user files and directories.

File system summary

It was decided that a useful file system would have the following characteristics:

- Use a Compact Flash module
- Allow creation and deletion of files.
- Allow files to expand.
- Use a hierarchical directory structure
- Have a ‘standard’ API.
- Present a UNIX like disk format.

Database

The system we were intending to implement as an embedded system depends on a quick look up database system. This posed a problem, how to implement a suitable database engine to allow record look ups, adds, deletes and indexing of a supplied by the host controller, with more than 250,000 records. The data table was the “.dat” part of an INFORMIX database.

Various look up possibilities were considered,

- B-tree index, could be quite disk intensive, using little RAM but difficult to code.
- Hash index, uses large block of RAM, around twelve times the number of records that are to be indexed in bytes, single disk access for each look up and quite easy to code.
- Binary lookup, completely out of the question as it would not be reasonable to shuffle records about on a block addressed storage device.

There was no off the shelf solution to this, not even to reverse engineer some C source suitable for embedded applications. After evaluating several solutions, it was decided to use a hash table with a large block of RAM, to provide a hash lookup for each record. A suitable hash function was found that gave a relatively even spread of hash values for a 32bit number.

The index that is created has then to be stored on the disk so that it doesn’t need to be rebuilt each time the controller is rebooted.

The structure of the hash record is shown below. The *key_data* happens to be a 32bit number. The *disk_offset* points to the location of the record on the disk from the start of the data file. The **next* points to the next record in the link list for the associated hash value.

```
Struct hash_record
      4     field key_data
      4     field disk_offset
      4     field *next
end-struct
```

There are other database tables in the system but fortunately, they are not as extensive as main table, can be indexed in a much simpler and non-memory intensive way.

Boot loader

What is an embedded system?

Is this a system were the operating system and application are blown into a ROM and then executed as a single entity?

Does it include a system that when booting up runs from ROM and then the software in the ROM loads the runtime software to RAM and executes the application and OS from the RAM after that?

Our system uses the second option. The board contains a boot ROM that in conjunction with the file system and the embedded Ethernet loads the operating system, and application into RAM and starts the application from there. The ROM is then only accessed for the necessary vector table for interrupts etc.

The sequence for the boot ROM from reset is as follows:

1. Test checksum of the program area and run application if all OK
2. If step 1 fails, Read the BOOT sector of disk for the location of the primary application, loads this application to RAM, and runs it.
3. If step 2 fails, Read the BOOT sector of disk for the location of backup application, loads this application to RAM, and runs it.
4. If step 3 fails, initialise the Ethernet, ask the operator for a suitable IP address and start the TFTP (Trivial File Transfer Protocol) server. The application can then be loaded from a PC with a TFTP client application.

The advantage of the boot loader and file system is that once the simple boot loader is in ROM the controller can easily be upgraded later by CEMFTP and stored on the disk. The system will of course need to be restarted for the new application to be used, but this can be done automatically.

Network security

We are implementing a controller that is to be used for security purposes and it is capable to being connected to the Internet, so the cry goes up, “what about network security?”.

This is a problem especially as customers require interoperability with standard applications, but want privacy.

What level of security do we need? Is login authentication enough? Do we need data encryption?

Connection authentication is the first step at least to ensure that unwanted persons cannot access the controller and change data records or other parameters, especially with the Web based and Telnet interfaces.

Data encryption is usually carried out by either the symmetric-key encryption method, were both the sender and receiver must have the same key to encrypt and decrypt the data, but how do we distribute key safely?

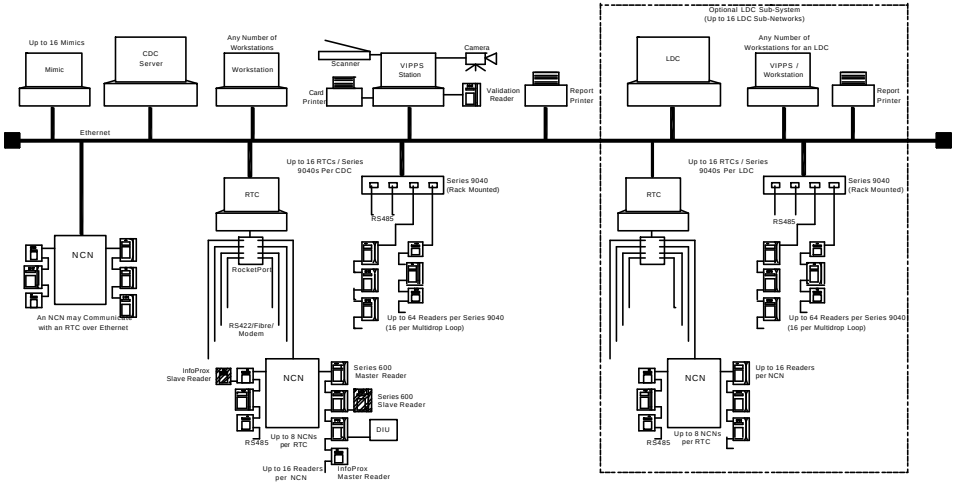
After checking various algorithms, we decided to use the Blowfish algorithm, with various source codes available, including Forth, from www.counterpane.com/blowfish.html and combine it with the XOR method for speed. Only part of the key is to be passed between the two parties, with the rest of the key being made from common information know to both parties, like the IP addresses, etc.

Data encryption can also be carried out by the Public/Private-key encryption method. This gets round the key distribution problem by having a different key to encrypt the data from the one that decrypts the data. This is beyond our capabilities currently. However, the algorithm has just come out patent by RSA Security Inc., so we may consider it in the future.

Conclusions

The software described is now being shipped in products, and has meet all its design aims. It has also given a very compact core that can be scaled for a range of products. The most interesting of which is a card with and integral web server inside a UK form factor light switch.

The next phase of work will involve using the ARM port of Forth and investigating what can be done with the programmable cores that are now becoming available.


Optional LDC Sub-System
(Up to 16 LDC Sub-Networks)
Up to 16 Mimics
Mimic
CDC
Server
Any Number of
Workstations
Workstation
Scanner
Camera
VIPPS
Station
Card
Printer
Validation
Reader
Report
Printer
LDC
Any Number of
Workstations for an LDC
VIPPS /
Workstation
Report
Printer
Ethernet
Up to 16 RTCs / Series
9040s Per CDC
RTC
Series 9040
(Rack Mounted)
RS485
NCN
RocketPort
An NCN may Communicate
with an RTC over Ethernet
RS422/Fibre/
Modem
Up to 64 Readers per Series 9040
(16 per Multidrop Loop)
InfoProx
Slave Reader
NCN
Series 600
Master Reader
Series 600
Slave Reader
DIU
RS485
Up to 8 NCNs
per RTC
Up to 16 Readers
per NCN
InfoProx
Master Reader
Up to 16 RTCs / Series
9040s Per LDC
RTC
Series 9040
(Rack Mounted)
RS485
Up to 64 Readers per Series 9040
(16 per Multidrop Loop)
NCN
Up to 16 Readers
per NCN
RS485
Up to 8 NCNs
per RTC