What Have We Lost

Lloyd Moore, President

Lloyd@CyberData-Robotics.com

www.CyberData-Robotics.com

Agenda:

As technology and software development techniques have moved forward we have left behind some simpler techniques that are still useful. Let’s review some of them….

Link Time Overlays

Description:

- A linker option allowing individual functions to be replaced / mocked in an executable. The technique is somewhat related to function overlays of the distant past when memory was VERY scarce.

Modern Day Approach:

- Dynamic Linked Libraries (to some degree)

How To:

- Set the linker command line option allowing multiple definitions
  --allow-multiple-definition for GCC
- Identify individual functions to be replaced
- Create duplicate function definitions in different compilation units/object files/libraries
- Based on the build type include or don't include the object the object with the alternate definition

Link Time Overlays

Advantages:

- Allows for a known common code based to be modified for testing, while GUARANTEEING the integrity of the rest of the program.
- Particularly useful when software needs to pass a formal qualification and the exact code used in production must be modified.
- Linkage is static and minimal modifications are needed to the program.
- Program configuration is controlled and guaranteed by the build system.

Disadvantages:

- Technique is obscure and not obvious to many folks these days.
- Duplicate function names are not expected and will mislead folks.
- Generally frowned upon if used where not absolutely required.

CSV Files, Simple Formats

Description:

- Using only the simplest file format needed to represent the information.

Modern Day Approach:

- Markup languages such as JSON and XML

How To:

- For a CSV file write out an optional header line with the field titles comma separated:
  Field1,Field2,Field3
- Write out each set of values, comma separated as a single line:
  1,2,3

- If size and performance are really critical can also use a binary format such as:

STX	0x01	0x02	0x03	ETX

CSV Files, Simple Formats

Advantages:

- For simple, regular data the format is easy to read and write in code. (fprintf, fscanf, strtok or similar)
- Removes the need to pull in a serialization/de-serialization library.
- Files are typically MUCH smaller in size as redundant field names and delimiters are removed.
- Can easily interop with a spreadsheet.
- Binary files can be even smaller and faster.

Disadvantages:

- Not as human readable.
- Generally not usable where the data structure varies significantly, and if it does much of the simplicity is lost.
- Depending on the application may not be as resistant to error detection.
- Needs custom code to parse into internal data structures (JSON).

Simple Serial Protocols

<u>Description:</u>

- Basically the same ideas we just discussed but instead of storing to disk the information is sent over a serial link (cabled or wireless).

<u>Modern Day Approach:</u>

- More complex formats such as JSON or XML embedded inside a transmission protocol such as TCP/IP.

<u>How To:</u>

- Most beneficial here will be the binary protocols in cases where efficiency is critically important.
- Generally will want to have a packet format with the following properties:
  - A clear start of packet/message indicator for framing
  - A clear end of packet/message indicator for framing
  - Some type of error detection and/or correction (1+2 = 3)

STX	0x01	0x02	0x03	ETX

Simple Serial Protocols

Advantages:

- Can be optimized and tuned for the specific application:
  - Send x,y coords from [0,99]
  - Errors handled by repeating packet
  - Can do this with 2 bytes by flagging one coordinate having high bit set
  - A typical JSON packet would be about 20 bytes (depending on exact white space):

```
{
  "x": 10,
  "y": 20
}
```

- Allows for the use of simpler lower speed physical layer, sending more data, and/or faster update rate.

Disadvantages:

- Not easily human readable.
- Not parsed by protocol analyzers.
- Have to write custom code on both sides to match to data contents.
- Complex or variable data becomes much harder to work with.

Simple Serial Ports

Description:

- RS-232/422/485 are simple 2 to 9 wire serial ports that have been around for many decades. They used to be standard everywhere but now are only seen in special use cases.

Modern Day Approach:

- Ethernet most common for long distances, USB most common for short distances.

How To:

- PCI cards are still available that support RS-232/422/485
- Can also use USB to RS-232/422/485
- FTDI chipset is the most common and widely supported on all major operating systems
- Ethernet to RS-232/422/485 adapters are available

- In code simply open the serial port as a file and start reading and writing to it.
- Will need to configure the communication parameters (baud rate, start bits, stop bits and parity bit) in the OS.

Simple Serial Ports

Advantages:

- Simple to use and can have fine grained control
- Can be extremely robust to environmental influences
  - Chipsets can be “rad hard”
- Can easily connect to very simple microcontrollers and hardware devices
- Latency is well defined

Disadvantages:

- Will have to code any “networking stack” that you need yourself
- Low baud rate by today’s standards
- Limited range by today’s standards
- Some cabling can be heavy by today's standards

“Stream” File Processing

Description:

- Treat the file like a set of information packets, reading one “packet” at a time and processing it.

Modern Day Approach:

- Read the whole file into a large array in memory and process the array.

How To:

- Open the file just as you would normally
- Instead of reading the whole file into memory at once read just as much as you need for a single “work unit” of processing
- Keep reading the file and processing “work units” until you reach the end of the file

“Stream” File Processing

Advantages:

- In most cases code ends up being no more complex than working from memory – but it is different.
- The resulting code can typically be used on any type of “stream” including a network socket.
- Get some parallelism for free:
  - The OS will buffer and cache access to the file for you automatically
  - Can process the “work unit” while the OS works in the background on the file
- Can handle files of any size, including live streams that run forever!

Disadvantages:

- Random access to the data structure is harder in some cases.
- Some parsing operations will benefit from a state machine implementation which may be harder for some.

Blinking LED & Debug GPIO

Description:

- Configure a simple GPIO line as an output and toggle it from within your code. This is most effective during “board bring up”.

Modern Day Approach:

- Using the JTAG connected debugger and/or debug serial port of the device.

How To:

- Start by constructing an absolute minimum program needed, or use an existing example program.
- Configure a single line (or port whichever is simpler) to be an output.
- Set the output high and then low, possibly repeating at some interval.
- Monitor the state of the output line with a scope, logic analyzer or simply connect a low power LED to the line (exact circuit will depend on the hardware being used).

Blinking LED & Debug GPIO

Advantages:

- Extremely simple to get working compared to other approaches.
- Extremely low timing latency which can be critical for debugging some issues.
- Multiple lines can be used to convey system state in real time.
- Can be used to debug issues where the processor will not stay running (think watch dog).
- Used to be the “standard” first test running when bringing up a new board.
- Can work when the JTAG (or other debugging link) is unstable.

Disadvantages:

- Requires external hardware of some type to monitor.
- Requires some basic electronics knowledge to set up.
- Limited information can be sent.

“Bare Metal” Coding

Description:

- Creating your application code directly “against” the hardware without using a framework or operating system.

Modern Day Approach:

- Use a “small” ARM processor and put Linux on it.

How To:

- Create a description of what you are trying to do (requirements).
- If the application is simple, or has “hard real time” requirements consider working directly “against” the hardware.
- Exact details will vary greatly depending on what you are trying to do and could be several additional talks.

“Bare Metal” Coding

Advantages:

- Can implement more “hard real time” applications.
- Can be simpler and faster development in some cases.
- Can use lower cost hardware.
- Can improve power consumption.
- Can improve boot time.
- Fine grained access and control of all available hardware.
- Can use interrupts and DMA engines to simulate a multi-threaded application to improve determinism and latency.

Disadvantages:

- May require more coding to implement “device drivers”.
- Development and debugging tools more limited.
- You will be doing “embedded systems” development by it’s vary nature.

Hand Optimized Functions

Description:
- Creating a dedicated function that does only what you need it to do in place of using a library function.

Modern Day Approach:
- I need to do something – where is the library that does this for me?

How To:
- First consider what is REALLY special about what you need to do!
- In most cases just go get the off the shelf library and use it!
- Profile and evaluate the available libraries and be sure they cannot be made to work for you.
- Profile and evaluate the available libraries again and be sure they cannot be made to work for you.
- Ask a friend/co-worker to get a second, third and forth opinion.
- Make sure you know WHY the existing solutions won't work in your case and have some idea of how to improve the situation.
- Ok if you made it to here and you still think there is something special about what you need to do – code it up and make sure it works better for your use case!

Hand Optimized Functions

Advantages:

- Will be completely tailored to exactly what you need to do, and nothing else saving overhead.
- You might come up with a way better way of doing something and get some great “kudos” for it!

Disadvantages:

- Will take considerably more time to develop, debug and maintain.
- You might spend a lot of time attempting to develop the solution only to find out your solution is no better than the available solution.
- Coworkers are going to make you justify why you did it yourself!

Simply Installed Programs

Description:

- Creating a statically linked program that consists of ONE file only containing everything the application needs to run.

Modern Day Approach:

- Create a large set of files that need to be installed and then either create an installer application or wrap everything into a Docker container.

How To:

- Evaluate if a single file is really appropriate for your application.
- Put development processes in place that control the addition of dynamic linking, individual configuration files, registry settings and such. (Should have these anyway!)
- There are many way to embed binary data into an executable file if needed, and some frameworks support this directly.
  - “The Power of Compile Time Resources”, Jason Turner https://www.youtube.com/watch?v=3aRZZxpJ_fc
- On the “first run” the single executable can setup registry settings and other needed configurations in place of having an installer.

Simply Installed Programs

Advantages:

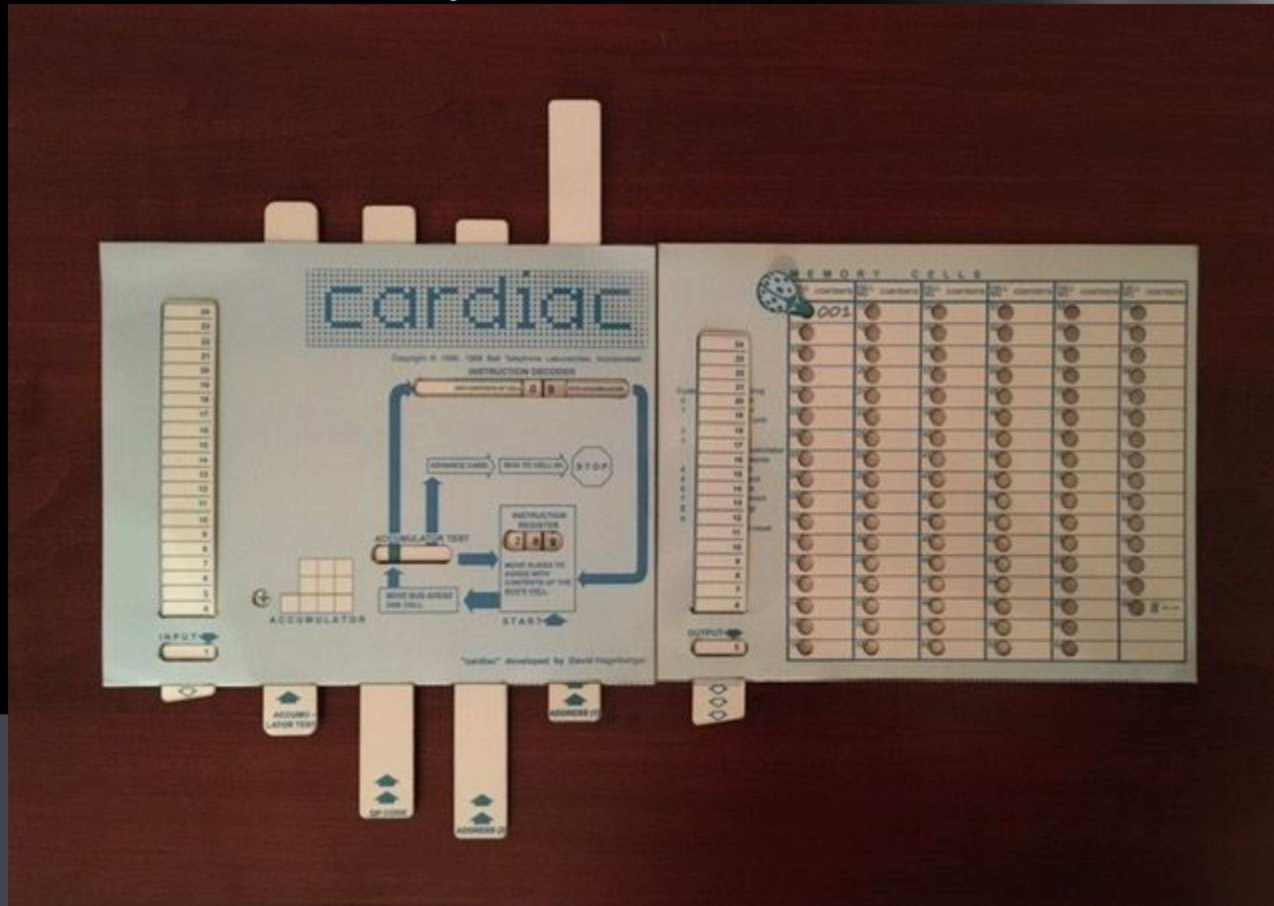
- Application is easily moved around.
- No installer needed.
- No dependency issues, or DLL hell.
- Application development can be simpler in some cases.
- Application performance can be faster in some cases.

Disadvantages:

- Architecture of the application will be limited.
- When there is an update the whole application will need to be replaced.
- Not all dependencies that you may need to use can be packaged into an executable.

Cardiac Cardboard Computer

Contributed by James Newton



A computer made from cardboard that was used for training. Simulated all the basic operations of a very simple CPU by sliding cardboard members.

Cardiac Cardboard Computer

Developed by James Newton

Memory Data Bus
Instruction Register
000
ALU
Decode
Accumulator
0000
Program Counter
00
Memory Address Bus
Memory

Deck
Reader
CPU
Output
PC: 00
Instruction Decoder
Instruction Register:
Opcode:
Operand:
Accumulator: 0000
Load
Read
Reset
Clear Mem
Step
Slow
Run
Halt

This is "forked" from
https://www.cs.drexel.edu/~bls96/museum/cardsim.html
(description) and edited by JamesNewton@MassMind.org via
https://stackblitz.com/edit/cardiac-computer-sim?file=script.js
to add an SVG block diagram which was made via:
https://svgedit.netlify.app/editor/index.html
and is now animated in the simulator to assist in the understanding of the user. The source code is at
https://github.com/JamesNewton/JamesNewton.github.io/edit/master/cardiac

Has been "reborn" as a website developed by James Newton:
https://jamesnewton.github.io/cardiac/

Open Discussion & Q & A

I’m sure there are other techniques that I have not covered, any stories from the audience….