ECE 471 – Embedded Systems
Lecture 7

Vince Weaver

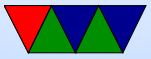
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

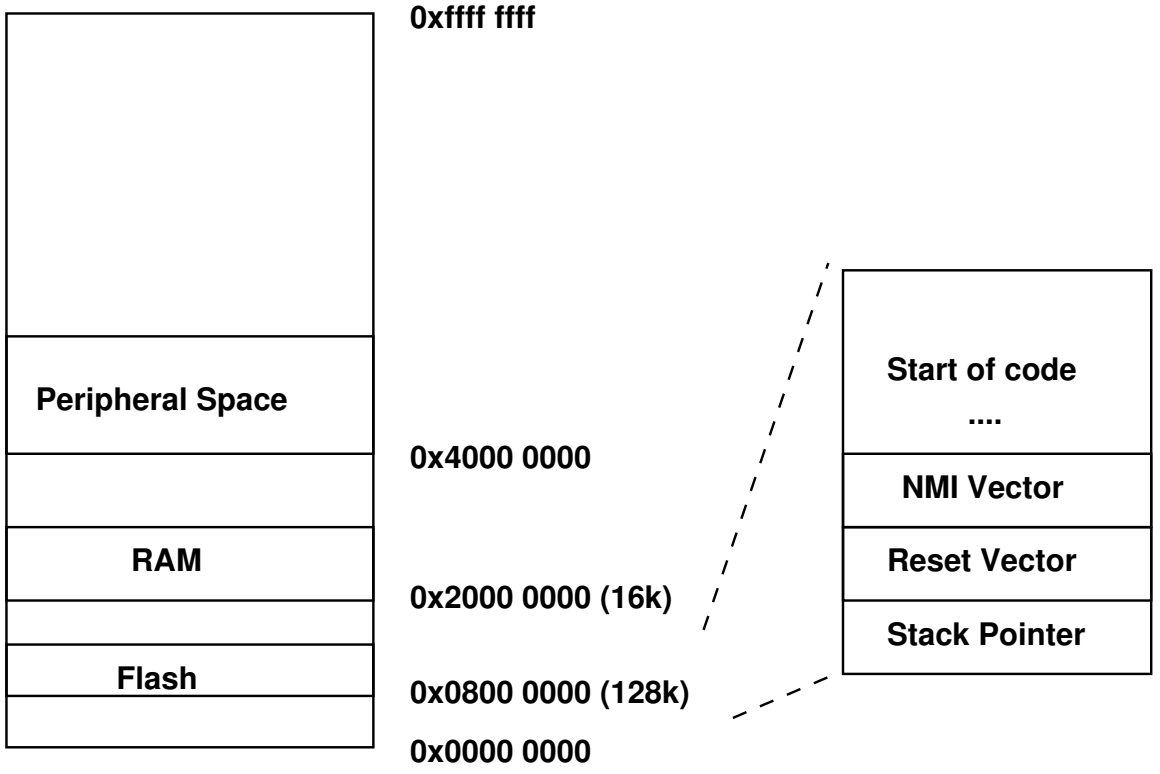
13 September 2017

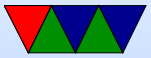
Announcements

- ???

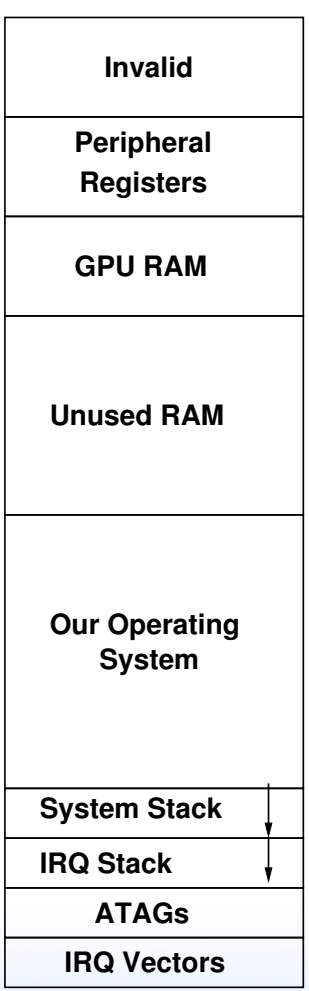


STM32L-Discovery Physical Memory Layout





Raspberry Pi Layout



0xffff ffff (4GB)

0x2100 0000 (528MB)

0x2000 0000 (512MB)

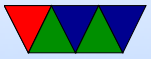
0x1c00 0000 (448MB)

0x0000 8000 (32k)

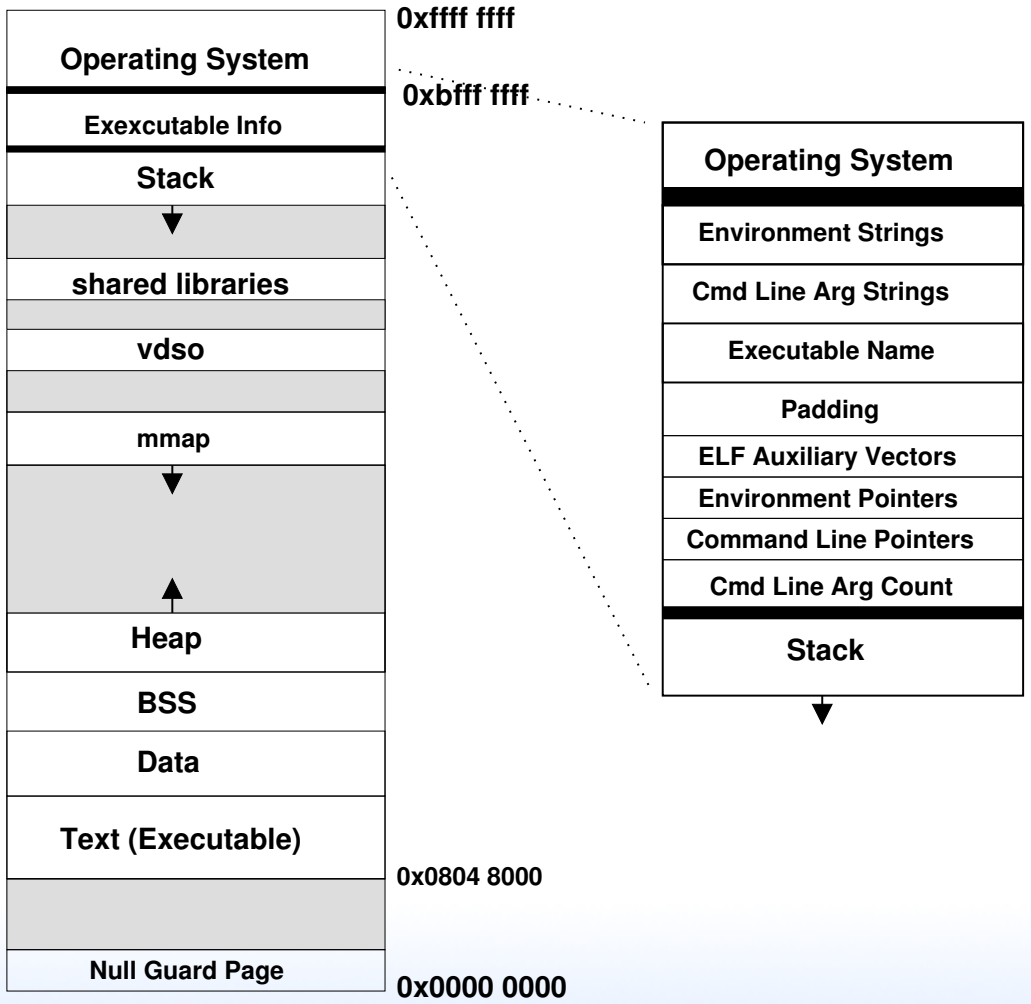
0x0000 4000 (16k)

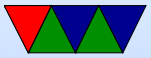
0x0000 0100 (256)

0x0000 0000



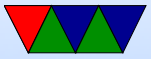
Linux Virtual Memory Map





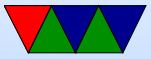
Program Memory Layout on Linux

- Text: the program's raw machine code
- Data: Initialized data
- BSS: uninitialized data; on Linux this is all set to 0.
- Heap: dynamic memory. `malloc()` (`brk()` syscall) and C++ `new()`. Grows up.
- Stack: LIFO memory structure. Grows down.



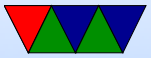
Program Layout

- Kernel: is mapped into top of address space, for performance reasons
- Command Line arguments, Environment, AUX vectors, etc., available above stack
- For security reasons "ASLR" (Address Space Layout Randomization) is often enabled. From run to run the exact addresses of all the sections is randomized, to make it harder for hackers to compromise your system.



Loader

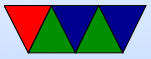
- /lib/ld-linux.so.2
- loads the executable (handles linking in libraries, etc)



Static vs Dynamic Libraries

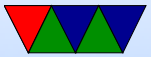
- Static: includes all code in one binary.
  Large binaries, need to recompile to update library code, self-contained, don't have to worry about incompatible updates

- Dynamic: library routines linked at load time.
  Smaller binaries, share code across system, automatically links against newer/bugfixes when system library updated

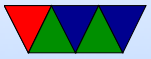


How a Program is Loaded

- Kernel Boots
- `init` started
- `init` calls `fork()` – makes an exact copy of itself
- child calls `exec()` – replaces itself with executable from disk
- Kernel checks if valid ELF. Passes to loader

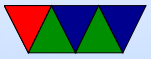


- Loader loads it. Clears out BSS. Sets up stack. Jumps to entry address (specified by executable)
- Program runs until complete.
- Parent process returned to if waiting. Otherwise, init.



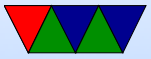
Assembly Language: What's it good for?

- Understanding your computer at a low-level
- Shown when using a debugger
- It's the eventual target of compilers
- Operating system writers (some things not expressible in C)
- Embedded systems (code density)
- Research. Computer Architecture. Emulators/Simulators.
- Video games (or other perf critical routines, glibc, kernel, etc.)



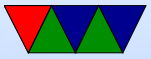
What the OS gives you at start

- Registers
- Instruction pointer at beginning
- Stack
- command line arguments, aux, environment variables
- Large contiguous VM space

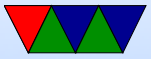


ARM Architecture

- 32-bit
- Load/Store
- Can be Big-Endian or Little-Endian (usually little)
- Fixed instruction width (32-bit, 16-bit THUMB) (Thumb2 is variable)
- arm32 opcodes typically take three arguments (Destination, Source, Source)
- Cannot access unaligned memory (optional newer chips)
- Status flag (many instructions can optionally set)

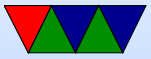


- Conditional execution
- Complicated addressing modes
- Many features optional (FPU [except in newer], PMU, Vector instructions, Java instructions, etc.)

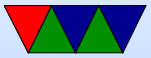


Registers

- Has 16 GP registers (more available in supervisor mode)
- r0 - r12 are general purpose
- r11 is sometimes the frame pointer (fp) [iOS uses r7]
- r13 is stack pointer (sp)
- r14 is link register (lr)
- r15 is program counter (pc)
  reading r15 usually gives PC+8
- 1 status register (more in system mode).
  NZCVQ (Negative, Zero, Carry, oVerflow, Saturate)

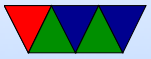


Low-Level ARM Linux Assembly



Linux C (ABI)

- Application Binary Interface
- The rules an executable needs to follow in order to talk to other code/libraries on the system
- A software agreement, this is not enforced at all by hardware
- r0-r3 are first 4 arguments/scratch (extra go on stack) (caller saved)



- r0-r1 are return value
- r4-r11 are general purpose, callee saved
- r12-r15 are special
- Things are more complex than this. Passing arrays and structs? 64-bit values? Floating point values? etc.