ECE 531 – Advanced Operating Systems
Lecture 17

Vince Weaver

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

vincent.weaver@maine.edu

10 October 2025

Announcements

- Midterm the 20th
- Don't forget fall break

HW#4 Review – Code

- Be sure your code compiles
- Note: timer running at 1MHz which is 10e6 Hz, not 2e20 Hz.

HW#4 Review – Question 1

- FIQ vs IRQ difference?
- FIQ banks some registers, so is faster (no saving),
- higher priority
- only one so don't have to search for source

HW#4 Review – Question 2

- BASIC_PENDING bit 19 is interrupt 57 which is UART.
- This info is in the manual but you have to consult two tables (confusing)
- First table (which) says it's GPU interrupt 57
- Second table says UART is interrupt 57

HW#4 Review – Question 3

- How to change modes?
- Write to the mode field of CPSR register.

HW#4 Review – Question 3

- Subtract 4 because it offsets by four when saving the PC
- Why? Historical reasons.
- Most likely original pipelined processor design it's just what the PC happened to be when an IRQ happened and it was easier to handle in software than hardware
- Then they were stuck with this forever

Memory Management

- Until now we have used static memory location (hard-coded at compile time)
- If we want to start running programs with fork/exec we need to allocate memory for the new processes
- How does memory allocation work?

Various Types of Memory Management

- Application / Userspace
- Operating System

Application Memory Allocation on C/Linux

```
#include <stdio.h>

int global_x=5; /* data */
int global_y=0; /* bss */

int main(int argc, char **argv) {

    int local_x=5; /* stack */
    static int static_y=5; /* data */
    static int static_x=0; /* bss */

    char *heap_x=malloc(1024); /* heap or mmap() */

    printf("Hello␣world\n");

    return 0;
}
```

Compiler Optimization Note

- The compiler can (and will) optimize away memory accesses whenever possible
- At -O2 optimization if you don't use a pointer to a variable it might only ever live in a register
- The old `register` keyword used to enforce this

Static Allocation – Data Segment

- Global and static variables that are initialized go in the data segment
- Loaded directly from the executable

Static Allocation – BSS Segment

- Global and static variables initialized to zero go in the bss segment.
- Uninitialized global/static variables also go in BSS
  On Linux at least these will be initialized to zero even if you don't request it
  You wouldn't want actually uninitialized data on process start up, huge security risk.
- These aren't in executable, it just holds the total BSS size request, and the OS allocates and zeros it at start

Dynamic Allocation – Variables on the Stack

- Local variables go on the stack
- Stack auto-grows down
- Note: stack often has strict alignment rules 4? 16?

```
/* arm32 */
int q[1000];

sub     sp, sp, #4000   @ 0xfa0
...g
add     sp, sp, #4000   @ 0xfa0
bx      lr

/* arm64 */
int q[1000];
```

```
adrp    x0, 0 <__abi_tag-0x278>
sub     sp, sp, #0xfa0
movi    v2.4s, #0x4
add     x1, sp, #0xfa0
ldr     q1, [x0, #2048]
mov     x0, sp
mov     v0.16b, v1.16b
add     v1.4s, v1.4s, v2.4s
mul     v0.4s, v0.4s, v0.4s
str     q0, [x0], #16    // q0 = 128 bit fp
cmp     x1, x0
b.ne    7b8 <foo+0x18>  // b.any
ldr     w0, [sp, #60]
add     sp, sp, #0xfa0
ret
```

Variables on the Stack – More

- Can you dynamically allocate on stack? `alloca()`
- Also variable defined arrays (gcc extension?)

```
int array[y];
```

- Downsides/Issues
  - stack overflow attacks (show example)
  - What happens if you return a pointer to a local variable
  - Contents of uninitialized variables might have old data, be not zero

Dynamic Memory Allocation – Heap

- `malloc()` is not a syscall, but a library call
- Generally the C library will request chunks of memory from the OS, then hand it out in smaller pieces as requested

The Heap

- "program break" is the address just above the data segment.
- can allocate/deallocate memory by moving this boundary
- Kernel interface is the `brk()` system call which moves the end of the data segment (essentially making the heap bigger)
- `brk(address)` moves new end of data segment to address if possible
- `sbrk(size)` moves break area by size

Dynamic Memory – mmap()

- Widely used modern method of getting memory to use
- `mmap()` initially mapped file into memory so can be accessed with load/store memory accesses rather than disk read/write
- You can specify ANONYMOUS access and it will back with zeroed out memory instead, essentially letting you allocate arbitrary sizes of memory
- In addition you can set extra constraints like READ / WRITE / EXEC to have it read only, read write,

executable (shared libraries are loaded this way, map memory, copy in as executable)

- Can mark as SHARED to share pages between processes for inter-process communication (IPC)
- MAP_FIXED can be used to request it be loaded at a specific address if possible
- MAP_LOCKED can request not be swapped out

How malloc() Works

- Many ways to write malloc(), each C library has own
- Basically a big chunk of RAM is grabbed from the OS, and then split into parts in a custom way.
- Do you just grab a chunk of mem and return a pointer? Or is there extra info you need to track?
- The biggest problem is fragmentation, which happens when memory is freed in non-contiguous areas.

dlmalloc – Doug Lea

- glibc uses ptmalloc based on dlmalloc
- Memory allocated in chunks, with 8 or 16-byte header
- bins of same sized objects, doubly linked list
- Small allocations (256kB?) closest power of two used
- Larger, mmap used, multiple of page size.

Manual vs Automatic

- With C you can manually allocate and free memory. Prone to errors:
  - Use-after-free errors
  - Buffer overflows
  - Memory Leaks
  - ALL OF THE ABOVE CAN LEAD TO ROOT EXPLOITS
- High-level languages such as Java will automatically allocate memory for objects.

- The user never sees memory pointers.
- Unused memory areas are periodically freed via “garbage-collection”.
- At the same time the memory can be compacted, avoiding fragmentation.
- Problem? Slow, not real-time, can be complex detangling complex memory dependency chains.

Pre-Fall-Break-Bonus

- “Pi-on-Fire” demo
- Runs on a Pi-1B on top of an OS based on the one from class
- Won second place in the “modern demo” compo at Demosplash 2019
- Chiptune music by DYA
- `http://www.deater.net/weave/vmwprod/pionfire/`