ECE 531 – Advanced Operating Systems
Lecture 35

Vince Weaver

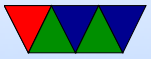
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

3 December 2025

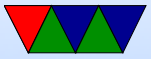
Announcements

- Don't forget Projects
- Working on grading HW6 and HW7
- Reminder: second midterm in class Friday

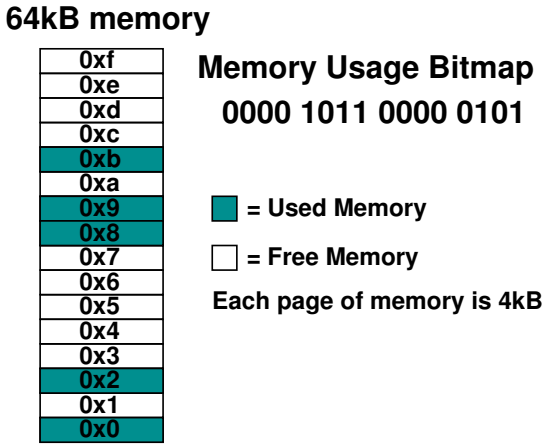


HW#6 Review – Code Investigation

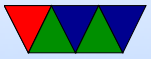
- Scheduling algorithm – round robin (it walks list, wraps around on end, stops if we get back to original)
- Memory Algorithm – first fit
- Changing to next-fit, mostly involves static variable to hold last found address and wrapping around when hit end



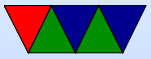
HW#6 Review – Questions



- Memory free, 11 bits 0, times 4k=44k
- First fit, at 0x3 as it's the first block with 4 consecutive blocks free
- Best fit would go at 0xc as that's exactly 16k



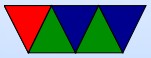
- Best fit might be better, reduce fragmentation
- Not possible to allocate 32k, even though 44k free
  Could you de-frag the RAM? Not if C pointers involved
- Note that using bitmaps like this can also be used to track free disk blocks. In that case you actually can de-fragment disk if needed.



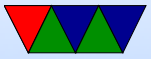
HW#7 Review – Virtual Memory

- Features of virtual memory?
  - Giving illusion of more RAM than we have
  - Demand paging
  - Needed for caching? not always, only on machines with VIPT (virtually-indexed) caches like on ARM/Pi
- What hides page lookup overhead? TLB
- What is it called when a page is not found in the page tables?

  Page fault. It's the OS's job to handle this

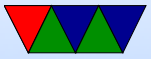


- If physical memory is full, room can be made by swapping out a page to disk. Can also kick out pages that aren't dirty (like executable data). Also things like disk cache.



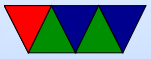
HW#7 Review – Filesystems

- What is the purpose of a filesystem?
  To find locations of files on a block device and lookup by name

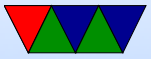


HW#7 Review – Device Types

- The SD-card driver is a block device
- char device: bytes streaming in, can't seek
- block devices: data comes in chunks, can usually more or less have random access to all the data



Multicore Concerns

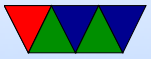


Race Conditions

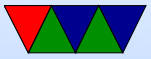
- Shared variable access (increment memory_free after allocate?)
- Read-Modify-Write on ARM, value starts at 0

Core A	Core B
Read value from memory	
Increment value in reg	Read value from memory
Write back to memory	Increment value in reg
	Write back to memory

- What is the final result?

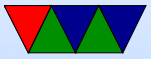


- What should the result be?



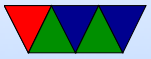
Critical Sections

- Want mutual exclusion, only one can access structure at once
  1. no two processes can be inside critical section at once
  2. no assumption can be made about speed of CPU
  3. no process not in critical section may block other processes
  4. no process should wait forever



How to avoid

- Single core: just disable interrupts
  - Bad for performance
  - OS might not let you do this as user (why?)
- Multicore: Locks/mutex/semaphore

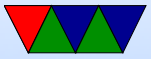


Mutex

- mutex_lock(&lock);
  - if unlocked (0), then it set lock=1 and return
  - if locked, return failure
  - what do we do if failed?

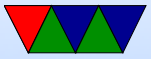
    Busy wait? (spinlock)

    re-schedule (yield)?
- mutex_unlock(&lock): sets lock to zero
- NOTE: we need special instructions for this (see later)



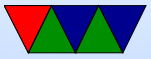
Semaphore

- Up/Down
- Wait in queue
- Blocking
- As lock frees, the job waiting is woken up



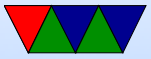
Locking Primitives

- Depend on Atomicity (what's that?)
- fetch and add (bus lock for multiple cores), xadd (x86)
- test and set (atomically test value and set to 1)
- test and test and set
- compare-and-swap
  - Atomic swap instruction SWP (ARM before v6, deprecated)
  - x86 CMPXCHG
  - Does both load and store in one instruction!



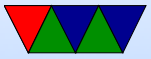
- Why bad? Longer interrupt latency (can't interrupt atomic op)
- Especially bad in multi-core

- load-link/store conditional
  - Load a value from memory
  - Later a store to same memory address.
  - Only succeeds if no other stores to that memory location in interim
  - ldrex/strex (ARMv6 and later) (requires VM active)
- Transactional Memory



Locking Primitives

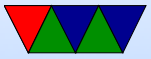
- can be shown to be equivalent
- how swap works:
  - lock is 0 (free). r1=1; swap r1,lock
  - now r1=0 (was free), lock=1 (in use)
  - lock is 1 (not-free). r1=1, swap r1,lock
  - now r1=1 (not-free), lock still==1 (in use)



ARMv6/ARMv7 Mutexes

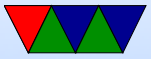
- On ARMv6 could use swap, but deprecated
- Now ldrex/strex. Note uses cache infrastructure so VM/caches must be initialized

```
.equ    MUTEX_UNLOCKED, 0
.equ    MUTEX_LOCKED,   1
.global mutex_lock
mutex_lock:
        ldr     r1, =MUTEX_LOCKED
lock_retry:
        ldrex   r2, [r0]
        cmp     r2, r1          @ are we already locked?
        wfeeq                   @ if so, go to sleep (wait for event)
        beq     lock_retry
        strex   r2, r1, [r0]    @ conditionally store value of r1 into r0
                                @ r2 lets you know if it worked or not
        cmp     r2, #1          @ if this failed
```



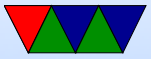
```
        beq     lock_retry      @ then keep retrying
        @ lock was acquired
        dmb                     @ Memory barrier
        bx      lr              @ return

.global mutex_unlock
mutex_unlock:
        ldr     r1, =MUTEX_UNLOCKED
        dmb                     @ memory barrier
        str     r1, [r0]        @ clear the lock
        dmb
        sev                     @ wake other processors waiting in wfe
                                @ by sending wakeup event
        bx      lr
```



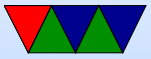
Memory Barriers

- Not a lock, but might be needed when doing locking
- Modern out-of-order processors can execute loads or stores out-of-order
- What happens a load or store bypasses a lock instruction?
- Processor Memory Ordering Models, not fun
- Technically on BCM2835 we need a memory barrier any time we switch between I/O blocks (i.e. from serial to GPIO, etc.) according to documentation, otherwise loads could return out of order



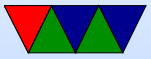
Deadlock

- Two processes both waiting for the other to finish, get stuck
- One possibility is a bad combination of locks, program gets stuck
- P1 takes Lock A. P2 takes Lock B. P1 then tries to take lock B and P2 tries to take Lock A.



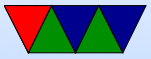
Livelock

- Processes change state, but still no forward progress.
- Two people trying to avoid each other in a hall.
- Can be harder to detect



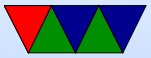
Starvation

- Not really a deadlock, but if there's a minor amount of unfairness in the locking mechanism one process might get "starved" (i.e. never get a chance to run) even though the other processes are properly taking and freeing the locks.



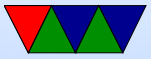
How to avoid Deadlock

- Don't write buggy code
- Reboot the system
- Kill off stuck processes
- Pre-emption (let one of the stuck processes run anyway)
- Rollback (checkpoint occasionally)



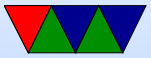
Priority Inversion

- Low-importance task interrupts a high-priority one
- Say you have a camera. Low-priority job takes lock to take picture.
- High-priority task wants to use the camera, spins waiting for it to be free. But since it is high-priority, the low priority task can never run to free the lock.



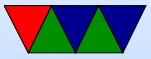
Locking in your OS

- When?
- Interrupts
- Multi-processor
- Pre-emptive kernel (used for lower latencies)
- Big-kernel lock? Fine-grained locking? Transactional memory?
- Semaphores? Mutexes
- Linux futexes?



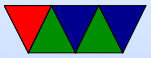
Where does our OS need locks?

- Does a single-core operating system need locks?
  Yes, interrupts can cause similar troubles
- Any shared resources
- What if multiple processes try to write the console at the same time?
- What if try to update the memory allocation/free list at same time?



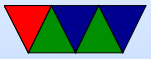
531-OS Locking Example

- Need to protect any place where multiple processes can be in the kernel at the same time accessing shared state
- For example, memory allocation. Multiple CPUs could be running code to access at same time
- If both do a `find_free()` at same time (before the other marks as reserved) and then return it, two processes could end up using the same memory (bad)
- Where to lock?
- You could lock the whole thing, but ideally want to lock



the smallest amount of code possible.

- Be sure when a lock is taken that all possible ways to exit the code release the lock, or could end up with deadlock
- Note that functions can be run by multiple cores as local variables end up on stack and each thread of execution has its own stack



531-OS Locking Example – Code

```
void *memory_allocate(uint32_t size) {

    int first_chunk,num_chunks,i;
// Lock here?  (No, why?)
    if (size==0) size=1;
    num_chunks = ((size-1)/CHUNK_SIZE)+1;
// Lock here? (yes, why?)
    first_chunk=find_free(num_chunks);
    if (first_chunk<0) {
        printk("Error!\n");
// Unlock here? (yes, why?)
        return NULL;
    }
    for(i=0;i<num_chunks;i++) memory_mark_used(first_chunk+i);
// Unlock here? (yes, why?)
    memset((void *)(first_chunk*CHUNK_SIZE),0,num_chunks*CHUNK_SIZE);
// Unlock here? (no, why?)
    return (void *)(first_chunk*CHUNK_SIZE);

}
```