CS350 Midterm Review

W18

A Modern OS

1. Creates the execution environment for a program; loads programs
2. Provides common utilities and libraries such as an I/O library
3. Abstracts hardware resources
4. Responds to interrupts, exceptions and system calls

Which of these objects has/runs an OS?

1. A car’s ECU (Engine Control Unit)
2. A cable set-top box
3. A router
4. A laptop or desktop
5. A cell phone
6. A smart watch
7. A TI graphing calculator

Which of these objects has/runs an OS?

1. A car’s ECU (Engine Control Unit)
2. A cable set-top box
3. A router
4. A laptop or desktop
5. A cell phone
6. A smart watch
7. A TI graphing calculator

ALL of them.

threads

1. Have their own stack, but share an address space
2. Execute concurrently
    a. Multi-threading (i.e., multiple CPUs)
    b. Timesharing
    c. Combination of (a) and (b)
3. Created with thread_fork
4. Yield with thread_yield
5. Exit with thread_exit

What is printed when Blah(NULL, 0) is called?

```
void Blah( void * n, unsigned long i ) {
    kprintf( “%d\n”, i );
    If ( i < 3 )  {
        thread_fork( “foo”, NULL, Blah, NULL, i + 1 );
        thread_fork( “foo2”, NULL, Blah, NULL, i + 1 );
        }
}
```

What is printed when Blah(NULL, 0) is called?

- **One 0**
- **Two 1s**
- **Four 2s**
- **Eight 3s**

0 will always come first

A 1 must preceed two 2s, a 2 must preceed two 3s.

synchronization

- Race conditions caused by multiple threads reading/writing same variable **OR** compiler re-arranging/optimizing code **OR** CPU re-arranging loads/stores
- Protect regions of shared access (called a critical section)
- Restrict the number of threads in a region of code
- Synchronize access to resources

What is the primary difference between

- Locks and semaphores
- Locks and condition variables

What is the primary difference between

- **Locks and semaphores**
  - **A lock lets a single thread into a critical section, and that thread is the only one that can release the lock**
  - **A semaphore is a resource counter. While there are resources available, threads may enter/take them. Any thread can produce new resources.**
- **Locks and condition variables**
  - **Condition variables let a thread block (because a condition it requires is not yet true) while in a critical section so that another thread by acquire the critical section and make the condition true**

Deadlock prevention

What are the advantages/disadvantages of “No-hold-and-wait”?

Deadlock prevention

Advantage:

- **Threads never own locks while they are blocked waiting to acquire another thread; i.e., no deadlock**

Disadvantage:

- **Threads spin while trying to repeatedly acquire a set of locks**

Which scenarios need synchronization?

1. N threads will read a global variable “MAXSIZE” but will never write it.
2. A car manufacturer produces cars at a rate of 100 cars per day. Car dealerships desire to purchase cars at a rate of 10 cars per month, but there are N of them.
3. An global array of values is used in several functions: element_wise_add, scalar_multiply and array_sum. These functions could be called by any number of concurrently running threads.

Which scenarios need synchronization?

1. N threads will read a global variable “MAXSIZE” but will never write it.
    a. **NO!**
2. A car manufacturer produces cars at a rate of 100 cars per day. Car dealerships desire to purchase cars at a rate of 10 cars per month, but there are N of them.
    a. **YES. Semaphores and a lock!**
3. An global array of values is used in several functions: element_wise_add, scalar_multiply and array_sum. These functions could be called by any number of concurrently running threads.
    a. **YES. A lock!**

Processes

- Execution environment created by OS for program to run
  - Address space
  - Threads
  - Resources (hardware, file system, etc).
- User processes isolated from kernel
  - Security, abstract design
- A user process can ONLY interact with the kernel via SYSTEM CALLS
  - V0 = system call code; a0-a3 = parameters, then SYSCALL
  - System call exception handled by KERNEL
- Threads now have TWO stacks
  - User stack, located in the address space of the process
  - Kernel stack, for executing/storing kernel data, located in KERNEL virtual memory
- PROCESSES SHARE NOTHING

Draw the kernel stack for a running thread that is pre-empted.

Draw the kernel stack for a running thread that is pre-empted.

1. **TRAPFRAME**
2. **Mips_trap**
3. **Mainbus_interrupt**
4. **Timer_handler**
5. **Thread_yield**
6. **Thread_switch**
7. **switchframe**

Draw the user stack for a process that calls waitpid.

Draw the user stack for a process that calls waitpid.

1. **User-code**
2. **waitpid**

Draw the kernel stack for a process that calls waitpid before determining what kind of exception has been raised.

Draw the kernel stack for a process that calls waitpid before determining what kind of exception has been raised.

1. **trapframe**
2. **mips_trap**

Draw the kernel stack for a process that calls waitpid.

Draw the kernel stack for a process that calls waitpid.

1. **trapframe**
2. **mips_trap**
3. **syscall**
4. **sys_waitpid`**

Draw the kernel stack for a process that calls waitpid and is preempted during sys_waitpid function.

Draw the kernel stack for a process that calls waitpid and is preempted during sys_waitpid function.

1. **trapframe**
2. **mips_trap**
3. **syscall**
4. **sys_waitpid**
5. **trapframe**
6. **mips_trap**
7. **mainbus_interrupt**
8. **timer_interrupt_handler**
9. **thread_yield**
10. **thread_switch**
11. **switchframe**

What is the difference between execv and fork?

What is the difference between execv and fork?

1. **Fork creates a NEW process. The new process is an identical clone of the parent, except for the return value from fork.**

2. **Execv changes the program that the process is running but does not change any other details of that process (i.e, PID, parent, children). The original address space is destroyed, a new one is created, the new program is loaded and begins execution.**

Suppose we modify our definition of waitpid such that it can wait on ANY process to exit, not just its children. What modifications to sys_waitpid are required?

Suppose we modify our definition of waitpid such that it can wait on ANY process to exit, not just its children. What modifications to sys_waitpid are required?

Only need to check if proc exists instead of checking if it is a child.

Can only fully delete proc after waitpid is called on it.

Virtual Memory

- Isolate programs from real memory
  - Security, abstraction, etc.
- Efficiency is key!
  - Space
  - Time
- Many techniques:
  - Dynamic relocation
  - Single-level paging
  - Segmentation
  - two/multi-level paging
- Don’t forget the TLB!

Dynamic Relocation

A system has 4GB of memory and uses 32 bit physical addresses. 16 bit virtual addresses are used.

What is the maximum number of processes that could fit into RAM at any one time assuming each process uses the maximum amount of virtual memory?

Dynamic Relocation

A system has 4GB of memory and uses 32 bit physical addresses. 16 bit virtual addresses are used.

What is the maximum number of processes that could fit into RAM at any one time assuming each process uses the maximum amount of virtual memory?

2^32/2^16 = 2^16

The maximum size for a processes address space is 2^16 bytes. So, divide the physical memory by this size.

Dynamic Relocation

If the virtual address is equal to the value in the MMU’s limit register, what happens?

Dynamic Relocation

If the virtual address is equal to the value in the MMU’s limit register, what happens?

Exception

Single Level Paging

A system has 4GB of memory and uses 32 bit physical addresses. Virtual memory uses 16 bit virtual addresses. Page size is 4KB (2^12 bytes).

1. How many bits are required for the page offset?
2. How many pages fit into virtual memory?
3. Assuming page size = frame size, how many frames fit into physical memory?
4. How many bits are needed for the page number?
5. How many bits are needed for the frame number?
6. If a page table entries valid bit is 0, what does the MMU do?

Single Level Paging

A system has 4GB of memory and uses 32 bit physical addresses. Virtual memory uses 16 bit virtual addresses. Page size is 4KB (2^12 bytes).

1. How many bits are required for the page offset? **12**
2. How many pages fit into virtual memory? **2^16/2^12 = 2^4**
3. Assuming page size = frame size, how many frames fit into physical memory? **2^32/2^12 = 2^20**
4. How many bits are needed for the page number? **4**
5. How many bits are needed for the frame number? **20**
6. If a page table entries valid bit is 0, what does the MMU do? **exception**

Single Level Paging

A system has 4GB of memory and uses 32 bit physical addresses. Virtual memory uses 16 bit virtual addresses. Page size is 4KB (2^12 bytes).

1. If a process uses a 64KB contiguous region of virtual memory for its address space, starting at vaddr = 0x0, how many pages does it use?
2. If a process uses a 129KB contiguous region of virtual memory for its address space, starting at vaddr = 0x0, how many pages does it use?
    a. What if we now use 16KB pages?

Single Level Paging

A system has 4GB of memory and uses 32 bit physical addresses. Virtual memory uses 16 bit virtual addresses. Page size is 4KB (2^12 bytes).

1. If a process uses a 64KB contiguous region of virtual memory for its address space, starting at vaddr = 0x0, how many pages does it use?
    a. 64/4 = 16
2. If a process uses a 129KB contiguous region of virtual memory for its address space, starting at vaddr = 0x0, how many pages does it use?
    a. ceil(129/4) = 33
    b. What if we now use 16KB pages?
        i. 9

How does a software-managed TLB differ from a hardware-managed TLB?

How does a software-managed TLB differ from a hardware-managed TLB?

In a software-managed TLB, TLB misses are handled by the KERNEL OF THE OS. The kernel will find the page-to-frame mapping and ADD IT TO THE TLB, so the instruction can be re-issued (and will be a TLB hit).

In a hardware-managed TLB, TLB misses are handled by the HARDWARE.

Segmentation

A system uses 16 bit physical and virtual addresses. Address spaces have 9 segments.

1. How many bits are used for the segment number?
2. How many bits are used for the segment offset?
3. What is the segment and offset for:
    a. 0x0001
    b. 0x10AB
    c. 0x82FF
    d. 0x207F

Segmentation

A system uses 16 bit physical and virtual addresses. Address spaces have 9 segments.

1. How many bits are used for the segment number? **4**
2. How many bits are used for the segment offset? **12**
3. What is the segment and offset for:
    a. 0x0001 **SEG = 0, OFFSET = 0x0001**
    b. 0x10AB **SEG = 1, OFFSET = 0x00AB**
    c. 0x82FF **SEG = 8, OFFSET = 0x02FF**
    d. 0x207F **SEG = 2, OFFSET = 0x007F**