CS350 Midterm Review

W19

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

Give two reasons why an OS might abstract how data is stored on a disk.

Give two reasons why an OS might abstract how data is stored on a disk.

1. **Security. If a use program can directly access data as it is stored on disk, the user may be able to bypass permissions, etc.**
2. **Good abstract design. The OS can change the implementation without affecting the user program.**

Introduction 8.1

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

Threads and Concurrency 4, 5, 10, 11

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

A thread needs to wait for a variable, cost, to be less than 100 before proceeding. Use only a lock to show how that thread might synchronize this.

A thread needs to wait for a variable, cost, to be less than 100 before proceeding. Use only a lock to show how that thread might synchronize this.

```
lock_acquire( lk )
        while( cost >= 100 )
                lock_release( lk )
                lock_acquire( lk )
        use cost
lock_release( lk )
```

With cooperative concurrency, a context switch only occurs if a thread exits, blocks, or yields. Can there still be race conditions with this model?

1.

With cooperative concurrency, a context switch only occurs if a thread exits, blocks, or yields. Can there still be race conditions with this model?

Yes, if the CPU supports multithreaded applications.

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**

Synchronization 54-56

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.
4. A program where each the i-th thread accesses only the i-th element of an array.

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!**
4. A program where each the i-th thread accesses only the i-th element of an array.
   a. **NO!**

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**

Process and the kernel 33-42

Draw the user stack for a process that calls fork.

Draw the user stack for a process that calls fork.

1. **User-code**
2. **fork**

Process and the kernel 27-28

Draw the kernel stack for a process that calls fork while the kernel is determining what kind of exception has been raised.

Draw the kernel stack for a process that calls fork while the kernel is determining what kind of exception has been raised.

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

Process and the kernel 31-32

Draw the kernel stack for a process that calls fork.

Draw the kernel stack for a process that calls fork.

1. **trapframe**
2. **mips_trap**
3. **syscall**
4. **sys_fork**

Process and the kernel 29-33

Draw the kernel stack for a process that calls fork and is preempted during sys_fork function.

Draw the kernel stack for a process that calls fork and is preempted during sys_fork function.

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

Process and the kernel 29-42

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.**

Process and the kernel 4, 6

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
  - Segmentation

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

Virtual Memory 8

List some disadvantages of dynamic relocation.

List some disadvantages of dynamic relocation.

1. **external fragmentation**
2. **large amounts of wasted space for sparse-layout address spaces**
    a. **results in lower-degree of multiprocessing due to large size address spaces**

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**

Segmentation T/F for a K-bit virtual address

1. Each segment has a maximum possible size 2^k bits.
2. Each segment uses the same number of bits for the segment number.
3. There is a maximum number of segments that can exist.
4. There is no fragmentation with segmentation.
5. The MMU has 3 registers for every segment.

Segmentation T/F for a K-bit virtual address

1. Each segment has a maximum possible size 2^k bits.
   a. **FALSE**
2. Each segment uses the same number of bits for the segment number.
   a. **TRUE**
3. There is a maximum number of segments that can exist.
   a. **TRUE**
4. There is no fragmentation with segmentation.
   a. **FALSE**
5. The MMU has 3 registers for every segment.
   a. **FALSE**

Virtual Memory 16