VE482 Review

C1

Computers and Operating Systems

Job of an Operating System (OS)

- Manage and assign the hardware resources
- Hide complicated details to the end user
- Provide abstractions to ease interactions with the hardware

Hardware

Composition

- CPU
  - computer's brain
  - can only execute a specific set of instructions
  - fetches instructions from the memory and executes them
  - Two types:
    - Pipeline:
      - Execute instruction n, decode $n+1$ and fetch $n+2$
      - Any fetched instruction must be executed
      - Issue: conditional statements
    - Superscalar:
      - Multiple execution units, e.g. one for float, int, boolean
      - Multiple instructions fetched and decoded at a time
      - Instructions held in buffer to be executed
      - Issue: no specific order to execute buffered instructions
  - multi-threading
    - Real multi-threading:
      - Several physical CPU cores are available
      - The OS sees several CPUs and can use them all simultaneously
    - Fake multi-threading (Hyper-threading):
      - A physical CPU core is seen as two logical cores by the OS
      - Some resources are duplicated in each physical core
      - Hyper-threading allows a better utilisation of the CPU
  - Socket: the physical computing component
  - Core: number of independent CPUs on a socket
  - Threads: maximum number of instructions that can be passed through or processed simultaneously by a single core
  - Number of logical cores: number of cores times number of threads
- Memory
  - Random Access Memory (RAM): volatile

  - Read Only Memory (ROM)
  - Electrically Erasable PROM (EEPROM) and flash memory: slower than RAM, non volatile.
  - CMOS: save time and date , BIOS parameters
  - HDD: divided into cylinder, track and sector
- I/O devices

They are connected by bus.

Basic concepts

Major components of an OS

- System calls: allows to interface with user-space
  - Call for kernel service from user-space
  - Processes:
    - `pid=fork(); pid=waitpid(pid, &statloc, options); s=execve(name, argv, environp); exit(status);`
  - Files:
    - `fd=open(file,how,…); s=close(fd); s=stat(name,*buf); n=read(fd,buffer,nbytes); n=write(fd,buffer,nbytes); position=lseek(fd,offset,whence);`
  - Directory and file system:
    - `s=mkdir(name,mode); s=rmdir(name); mount(special,name,flags,types,args); umount(name); s=unlink(name); s=link(name1,name2);`
  - Misc:
    - `s=chdir(dirname); s=chmod(name,mode); sec=time(*); s=kill(pid,signal);`
- Processes: holds all the necessary information to run a program
  - Address space, which contains:
    - Executable program
    - Program's data
    - Program's stack
  - Set of resources:
    - Registers
    - List of open files
    - Alarms
    - List of related processes
    - Any other information required by the program
- File system: store persistent data
  - The OS hides peculiarities of the disk and other IO devices
    - The top directory is called root directory
    - Removable devices can be mounted onto the main tree
    - Block files: for storage devices such as disks
    - Character files: for devices accepting or outputting character streams
    - Pipe: pseudo file used to connect two processes
- Input-Output (IO): allows to interface with hardware
- Protection and Security: keep the system safe
  - CPU: divided into kernel mode and user mode
    - Kernel Mode:

            - Set using a bit in the PSW
            - Any CPU instruction and hardware feature are available
        - User mode:
            - Only a subset of instructions/features is available
            - Setting PSW kernel mode bit forbidden
    - Memory:
        - Base and limit registers: holds the smallest legal physical memory address and the size of the range, respectively
        - Memory outside the address space is protected
    - Input and Output:
        - They are all privilege instructions
        - The OS processes them to ensure their correctness and legality

Communication strategies between components

- Interrupt: Device controller interact with CPU.
    - Operating systems are almost always interrupt driven
    - Hardware interrupt:
        - Send instructions to the controller
        - The controller signals the end
        - Assert a pin to interrupt the CPU
        - Send extra information
    - Software interrupt:
        - A call coming from userspace
        - A software interrupt handler is invoked
        - System call: switch to kernel mode to run privileged instruction
- CPU busy waiting: continuously poll device.
    - Simplest method:
        - Call the driver
        - Start the input-output
        - Wait in a tight loop
        - Continuously poll the device to know its state
    - Waste resources
- Direct Memory Access (DMA): fast communication between memory and device.
    - Can transmit information close to memory speeds
    - Directly transfer blocks of data from the controller to the RAM
    - Only little needed from the CPU
    - Issue a single interrupt per block, instead of one per byte

Kernel Space vs. User Space

- Security: Kernel space is secured
- Speed (is the same, actually)
- Available resources: Something can only be used in kernel space
- Available actions: Something can only be used in kernel space

Common operating system structures: Monolithic vs. Micro kernel

Micro kernel will have more things in user space, which brings more flexibility. However, it needs to cross the boarder of user/kernel mode many times, which is slow.

MINIX3 is micro kernel.

Hybrid: between them.

C2

Processes

A process is an abstraction of a running program:

- At the core of the OS
- Process is the unit for resource management
- Turn a single CPU into multiple virtual CPUs
- CPU quickly switches from process to process
- Processes hide the effect of interrupts

Program vs. Process

- Running twice a program generates two processes
- Program: sequence of operations to perform
- Process: program, input, output, state

Process creation and termination

- Create:
    - System initialization
    - Upon a user launching a new program
    - Initialization of a batch job
- End:
    - Voluntarily:
        - The work is completed, issue a system call to inform the OS
        - An error is noticed, the process exits nicely
    - Involuntary:
        - Fatal error, program crashes
        - Another process kills it

Process hierarchies

- UNIX-like systems:
    - A parent creates a child
    - A child can create its own child
    - The hierarchy is called process group
    - It is impossible to disinherit a child
- Windows system:
    - All processes are equal
    - A parent has a token to control its child
    - A token can be given to another process

Process states

- Blocked (waiting for input)
- Running (only 1 process can be running at a time)
- Ready (waiting for CPU)

Modeling processes

Interrupts and processes

Threads

A thread is the basic unit of CPU utilisation consisting of:

- A thread ID
- The program counter
- A register set
- A stack space

All the threads within a process share the same:

- Code section
- Data section
- Operating system resources

Process vs. Threads

- Similarity:
  - A thread has the same possible states as a process
  - Transitions are similar to the case of a process
- Difference:
  - No protection is required for threads.
  - Process starts with one thread and can create more.
  - Threads can voluntarily give up CPU while processes want as much CPU as they can.
  - Creating a thread is much faster than creating a process

Implementation

`pthread` library

User-space threads (N:1)

Corresponding to micro kernel

- Kernel thinks it manages single threaded processes
- Threads implemented in a library
- Thread table similar to process table, managed by runtime system
- Switching thread does not require to trap the kernel

Kernel space thread (1:1)

Corresponding to monolithic kernel

- Kernel manages the thread table
- Kernel calls are issued to request a new thread
- Calls that might block a thread are implemented as system call
- Kernel can run another thread in the meantime
- Most systems are coming back to this implementation

Hybrid threads (M:N)

- Compromise between user-level and kernel-level
- Threading library schedules user threads on available kernel threads

C3

Single thread

In single tasking all threads are independent

- They cannot affect or be affected by anything
- Their state is not shared with other threads
- The input state determines the output
- Everything is reproducible
- Stopping or resuming does not lead to any side effect

Multiple threads

- A thread runs on one core at a time
- A thread can run on different cores at different times
- Each core is shared among several threads
- Several cores run several threads in parallel
- The number of cores has no impact on the running of the threads
- Changes made by one thread can affect others

Thread and process cooperation

Setup

- Several threads share a common global variable
- The execution sequence impacts the global variable
- By default the behavior is random and irreproducible

Race conditions

Two or more processes do an operation (e.g. edit a same variable) at the same time.

Critical region

The part of program that access a shared memory/file.

- No two processes can be in a critical region at a same time
- No assumption on the speed or number of CPUs
- No process outside a critical region can block other processes
- No process waits forever to enter a critical region

Peterson's idea

When wanting to enter a critical region a process:

- Shows its interest for the critical region
- If it is accessible it exits the function and accesses it
- If it is not accessible it waits in a tight loop

When a process has completed its work in the critical region it signals its departure

Side effects:

- Two processes: L, low priority, and H, high priority
- L enters in a critical region
- H becomes ready
- H has higher priority so the scheduler switches to H
- L has lower priority so is not rescheduled as long as H is busy
- H loops forever

Mutual exclusion at hardware level

Prevent the process in the critical region from being stopped.

- Disable interrupts:
    - Can be done within the kernel for a few instructions
    - Cannot be done by user processes
    - Only works when there is a single CPU
    - An interrupt on another CPU can still mess up the shared variable
- Atomic operations: Hardware level, not creatable, completely independent operation.
    - Either happens in its entirety or not at all
    - Several operations can be performed at once, e.g. A = B
    - Requires the CPU to support the atomic update of a memory space
    - Can be used to prevent other CPUs to access a shared memory
    - `TSL`: A simple atomic operation
        - Test and Set Lock: TSL
        - Copies LOCK to a register and set it to 1
        - LOCK is used to coordinate the access to a shared memory
        - Ensures LOCK remains unchanged while checking its value

Semaphore

- A positive integer variable
- Only changed or tested through two actions:

```
down (sem) {
    while (sem==0) sleep();
    sem--;
}
up (sem) {
    sem++;
}
```

- Checking or changing the value and sleeping are done atomically:

  - Single CPU: disable interrupts
  - Multiple CPUs: use TSL to ensure only one CPU accesses the semaphore
- Using semaphores to hide interrupts:
  - Each IO device gets a semaphore initialised to 0
  - A process accessing the device applies a down
  - The process becomes blocked
  - An interrupt is issued when the device has completed the work
  - The interrupt handler processes the interrupt and applies an up
  - The process becomes ready

Mutexes

A mutex is a semaphore taking values 0 (unlocked) or 1 (locked).

On a mutex-lock request:

- If the mutex is unlocked:
  - Lock the mutex
  - Enter the critical region
- If mutex is locked:
  - put the calling thread asleep
- When the thread in the critical region exits:
  - Unlock the mutex
  - Allow a thread to acquire the lock and enter the critical region

Monitors

- Only one process can be active within a monitor at a time
- A monitor can be seen as a "special type of class"
- Processes can be blocked and awaken based on condition variables and wait and signal functions

Barriers

Useful for problems where several processes must complete before the next phase can start.

C4

Abbreviation List

PSW: Program Status Word

- used to transfer between user/kernel mode

RAM: Random Access Memory

ROM: Read Only Memory

EEPROM: Electrically Erasable PROM

DMA: Direct Memory Access