Main Points of the OS module

You can find below the topics we have covered during the OS course. Reading the indicated chapters of the Tanenbaum book is essential for a good exam result. Read the Introduction (Ch.1) and the relevant parts of Chapters 10 (UNIX) and 11 (Windows 2000). Revising the exercises that I gave you from the book should be part of the preparation for the exam — see the web page

`http://www.dcs.bbk.ac.uk/~szabolcs/os.html`

for the list.

Processes and Threads

Processes

Section [2.1]

- Definition: program in execution
- Usage: multiprogramming; pseudo-parallelism
- Creation: principal events that cause process creation, system calls in UNIX and Windows
- Termination: conditions that terminate processes, system calls
- Hierarchy (in UNIX)
- States: running, blocked, ready; possible transitions between states
- Implementation: process table

Threads (aka Lightweight Processes)

Sections [2.2.1–2.2.5]

- Refined process model: resource grouping + thread of execution

- Multithreading: multiple execution in the same process (memory) environment for co-operation by sharing resources
- Modelling threads: multiple threads inside one process; per process data (address space, open files, global variables, etc.) and per thread data (program counter, registers, stack, state, etc.)
- Advantages of multithreading: pseudo-parallelism with shared address space and data, less overhead with context switch than with processes, better performance for I/O-bound applications
- Examples: 1) word processor with interactive and background threads, 2) multi-threaded web server with dispatcher and worker threads
- Implementations (and their relative advantages):
  - in user space (+: fast context switches, process-specific scheduling algorithms; -: problems with blocking system calls)
  - in kernel (+: easier semantics for blocking system calls; -: costly context switches)

Scheduling

Section [2.5]

- When to schedule: process creation and ternination, blocking system calls, interrupts
- Goals:
  - general (fairness, policy enforcement, balance)
  - batch systems (throughput, turnaround time)
  - interactive systems (response time, proportionality)
  - real-time systems (meeting deadlines, predictability)
- Preemtive and non-preemtive algorithms
- Scheduling in batch systems:
  - first-come-first-served (FCFS),
  - shortest-job-first (SJF),
  - shortest-remaining-time-next (SRTN)
- Scheduling in interactive systems:
  - round-robin scheduling,

  - priority scheduling,
  - multiple queues,
  - shortest-process-next (aging),
  - guaranteed scheduling,
  - lottery scheduling,
  - fair-share scheduling

- Comparison of the different scheduling algorithms with respect to the scheduling goals; policy versus mechanism
- Thread scheduling: user space and kernel threads
- Scheduling in UNIX and in Windows

Interprocess Communication

Sections [2.3, 2.4.1, 2.4.2]

- Race condition between concurrent cooperating processes or threads that share data, examples (spooling directory), consequences
- Critical-section problem; solution should satisfy
  1. mutual exclusion,
  2. no assumption on speed of processes,
  3. progress (no deadlock),
  4. bounded waiting (no starvation)
- Solutions for two processes using busy waiting
  - software: Peterson's solution
  - help from hardware: test-and-set lock (TSL)
- Semaphores:
  - definition: integer variable with *up* and *down* operations, sleep and wake up, queues,
  - usage: eliminates busy waiting, but possible starvation and deadlock,
- Monitors
- Solutions for classical synchronization problems avoiding busy waiting

  - bounded-buffer (producer–consumer)
  - readers–writers
  - dining philosophers
  - sleeping barber

- Interprocess communication (IPC) via message passing as an alternative to shared memory

Deadlocks

Chapter [3]

- Definition: a set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause;
- Examples; usually deadlock may occur when processes are granted exclusive access to resources (hardware or software);
- Characterization; the four necessay conditions for deadlock to occur:
  1. mutual exclusion,
  2. hold and wait,
  3. no preemption,
  4. circular wait.
- Modeling deadlocks
  - One resource of each type: resource allocation graph using process and resource nodes and arrows for indicating requests and allocations;
  - Multiple resources of each type: allocation and request matrices.
- Methods for handling deadlocks
  - Ostrich algorithm: ignore the problem;
  - Prevention: deny one of the four conditions above
    mutual exclusion — spooling,
    hold and wait — request all required resources initially,
    no preemption — take resources away,
    circular wait — linearly order resources;
  - Detection and recovery: detect deadlocks and take recovery action (rollback or kill)

    - single instances of resource types: check for cycles in the resource allocation graph,
    - multiple instances of resource types: try to satisfy the needs of at least one process;
  - Avoidance: make sure that the machine reaches only safe states, i.e., that it is guaranteed that there is some scheduling order in which every process can run to completion even if they request their maximum number of resources immediately
    - single instance of resource type: detecting cycles in the resource allocation graph,
    - multiple instances of resource type: Banker's algorithm.

Memory Management

Sections [4.1–4.4, 4.5.1, 4.6.1, 4.6.3, 4.8.1, 4.8.2]
Main goal: to allocate (main) memory to processes in an efficient and safe way

- MM without virtual memory: multiprogramming with fixed partitions, swapping; implementation by bitmaps or linked lists; memory allocation algorithms
- Virtual memory:
  - provides large logical (virtual) address space for programmers
  - only those parts of the program are in main memory that are currently used by a process
  - mapping between logical and physical addresses
  - paging and segmentation
- Paging:
  - page tables (single- and multilevel), translation lookaside buffers (TLB), inverted page tables
  - page fault handling
  - page replacement algorithms: optimal, not recently used (NRU), FIFO, second chance and clock, least recently used (LRU) and aging, WSClock
  - working set and locality of reference, thrashing, Belady's anomaly
  - design issues: local vs. global allocation, load control, page size, cleaning policy
  - paging in UNIX and Windows
- Segmentation: two dimensional address space — virtual memory is divided into logical units (segments); supports protection and sharing
- Segmentation with paging: MULTICS

I/O System

Sections [5.1–5.4]

- Principles of I/O hardware
  - Devices: block and character
  - Communication between device and machine: port, bus, controller
  - Memory mapped I/O
  - Direct memory access (DMA)
  - Interrupts
- Principles of I/O software
  - Goals: device independence, uniform naming, error handling, buffering
  - Programmed I/O: polling, busy waiting
  - Interrupt-driven I/O: context switches, interrupts
  - DMA (programmed I/O via the DMA, interrupt on completion)
- I/O software layers
  - Interrupt handlers
  - Device drivers: their tasks, uniform interface
  - Device-independent I/O software: buffering
  - User-space I/O software: spooling
  - Life cycle of an I/O request
- Disks
  - Hardware: cylinders, tracks, sectors
  - Efficiency and reliability using RAID schemas: strips, mirroring, block interleaved parity
  - Disk formatting, cylinder skew, interleaving
  - Disk scheduling: FCFS, SSF, Elevator algorithm; their performance with respect to access time (seek time + rotational latency) and fairness
  - Error handling: spare sectors

File Systems

Sections [6.1–6.3, 6.4.3, 6.4.5, 11.7]

- File: contiguous logical address space, attributes (name, size, type, etc.), operations (open, delete, read, write, etc.)
- Access methods: sequential or random (direct)
- Directory: hierarchial system for organizing files; tree structure or general directed graph structure, relative and absolute path names, operations (create, delete, open, close, etc.), symbolic and hard links
- Implementation — simplicity, fast and flexible access, efficient use of space
  - contiguous allocation
  - linked list and file allocation table (FAT)
  - i-nodes

  Directory entry
- Disk space management: block size, free list on linked list or bitmap, quotas
- Reliability and recovery, Performance
- Example file systems
  - MS-DOS using FAT
  - UNIX using i-nodes
  - Windows 2000 using master file table (MFT)

Multiple Processor Systems

Chapters [8.1, 8.2]

- Advantages: resource sharing, computation speedup, reliability, communication
- Types: multiprocessors (shared memory), multicomputers (private memories and message passing), distributed systems (complete systems connected by network)
- Multiprocessors:
  - hardware: bus-based, crossbar switch and omega switching networks; master-slave and symmetric multiprocessors
  - synchronization: test-and-set lock, semaphores; spinning vs. switching

  - scheduling: timesharing among unrelated processes, space sharing and gang scheduling for related processes

- Multicomputers:
  - hardware: interconnection toplogies, switching schemes (store-and-forward packet switching and circuit switching), network interface boards
  - communication software: blocking vs. non-blocking calls, remote procedure calls, distributed shared memory
  - scheduling: load balancing using graph-theoretic deterministic or heuristic algorithms

Protection and Security

Sections [9.1, 9.3.1, 9.3.4, 9.4, 9.5.1–9.5.4, 9.6, 9.7.1–9.7.3]
Controlling access to resources both internally and externally

- User authentication using passwords: encryption (using salt), countermeasures against illegal logins
- Inside attacks: trojan horses, login spoofing, logic bombs, trap doors, generic security attacks, design principles for security
- Outside attacks:
  - viruses: damage scenario, how they work, how they spread, parasitic and memory resident viruses
  - anti-virus techniques: checking file length, integrity, behavior, avoidance
- Protection mechanisms: protection domains, access control list, capability lists, trusted systems, multilevel security (Bell–La Padula, Biba)