Besturingssystemen

Examenvragen

Dit document is mede opgebouwd op basis van dit Google Doc (klik).

1 Theorie

Beslissingen besturingssysteem

Op verschillende plaatsen moet het besturingssysteem beslissingen nemen (vb. nieuwe thread/proces als de vorige is afgelopen, context switch bij schedulers, ...). Waar moet het besturingssysteem nog beslissingen nemen? Hoe worden ze geïmplementeerd en welke algoritmes kunnen hiervoor gebruikt worden?

Antwoord

Process scheduling Whenever CPU sits idle, OS must select process in ready queue to be executed.
Algorithms used: FCFS, SJF, Priority scheduling, RR, ...

Load balancing Keep workload balanced between multiple CPUs. Push/pull migration.

CPU hardware scheduling Decide which hardware thread to run on each core.

Paging (**page replacement**) When increasing multiprogramming, over-allocated memory (no free frames when page fault occurs).
Swap out process, free all frames, reduce level of MP ≈ page replacement.
Need to select victim frame and update associated page and frame tables.
Frame-allocation *algorithm* and page-replacement.
Page replacement: FIFO (Belady's anomaly: page-fault rate may increase as the number of allocated frames increases). Optimal page replacement: LRU. LRU-approximation: additional-reference-bits, second-chance, enhanced second-chance. Additional methods: page-buffering (free-frame pool).
Frame allocation: equal allocation, proportional allocation. Global vs local replacement.

Disk scheduling Practical relevance: having fast access time and large disk bandwidth.
Improve access time and bandwidth by managing the order in which disk I/O requests are serviced. Disk scheduling *algorithms*: FCFS, SSTF, SCAN/C-SCAN, LOOK/C-LOOK.
SSTF is common, increases performance over FCFS. Avoid starvation, use LOOK/C-LOOK.
Factors: file allocation (contiguous, linked, indexed), priority of requests (demand paging vs application I/O, writes vs reads).

Bestand openen in programma

Wat gebeurt er als je op een bestand klikt dat in een programma opent? Bv een wordbestand dat dan in word opent. Leg gedetailleerd en volledig uit en geef voorbeelden uit windows 7 en of linux.

Antwoord

1. **Fork** new process/thread

2. `open()` system call with access-mode parameter passes file name to the logical file system. Checked against **access rights**. Using process domain, look up access rights in access matrix. If read-only or read / write, access allowed.

3. Check **system-wide open-file table** and get location IF present, add entry in per-process open-file table ELSE search in-memory mount table for volume, in-memory directory-structure cache for recently accessed directories, search directory for file (or look up directory location in volume control block, if not cached) and add new entry in system-wide open-file table (copy of FCB) + new entry in per-process open-file table.
   Per-process open-file table contains copy of FCB + file pointer, file-open count, disk location of the file, access rights of all open files.

4. `open()` call returns pointer to appropriate entry in per-process open-file table (P-PO-FT). All operations performed via this pointer (specified via an index in P-PO-FT, so no searching required).

5. **Reading** a file: system call specifying name of file and where (in memory) to put the next block of the file. Directory is searched for the associated entry. System holds current-file-position pointer (in system-wide open-file table, pointed to by entry in per-process open file table): the location in the file where the next read/write is to take place.

6. If needed to write to file, acquire **exclusive lock**.

7. Removed from per-process open-file table at close, system-wide open-file table entry's open count decremented.

Staten van een proces

Geef alle staten waarin een process zich kan bevinden, en zeg wat er gebeurt bij elke overgang.

Antwoord

New The process is being **created** (by parent process).

Ready The process is **waiting** to be **assigned** to a processor.
New to ready: long-term scheduler puts process into ready queue in memory.
Running to ready: preempted / swapped out, interrupt occurs.
Waiting to ready: completion of I/O.

Running Instructions are being **executed**.
Ready to running: short-term scheduler assigns CPU to process.

Waiting The process is **waiting** for some **event** to occur (such as an I/O completion or reception of a signal).
Running to waiting: process is interrupted (requests I/O, `wait()` for termination of child)

Terminated The process **finished** execution.
Running to terminated: process terminates, resources deallocated, return exit status when parent calls `wait()`.

Booten

Bespreek in detail wat er gebeurt als de computer opstart. Vanaf het moment dat je de power knop induwt tot het moment dat de computer volledig is opgestart.

Antwoord

- **Booting** = the procedure of starting a computer by loading the kernel.
- Bootstrap program / bootstrap loader (sequential series of blocks, loaded as an image from ROM into memory, starts executing at fixed location; root partition). Bootstrap program initializes all aspects of system from CPU to device controllers and the contents of main memory.
  Code in boot ROM instructs device controller to read the boot blocks into memory and then starts executing that code. It then locates the kernel, loads it into main memory, and starts its execution.
- Root partition / **boot block** (fixed location on disk) is mounted at boot time and contains operating-system kernel.
- Finally, OS notes in its **in-memory mount table** that a file system is mounted, along with type of system.
- PCs: 2-step process; simple bootstrap loader fetches a more complex boot program from disk partition, which in turn loads the kernel.
- CPU reset (reboot/power up): instruction register loaded with predefined memory location of initial bootstrap program, located on ROM.
- Tasks of bootstrap program: run diagnostics on state of the machine, if passed, continue booting; initialize all system aspects (CPU registers - device controllers - contents of main memory)
- Large OS: bootstrap loader stored in firmware (ROM), OS on disk.
  Bootstrap program runs diagnostics + code that reads and executes code from boot (control) block on disk.
  Boot block loads remainder of bootstrap loader.
- OS probes hardware buses to determine what devices are present and installs corresponding **interrupt handlers** into the interrupt vector.
- Full bootstrap program is loaded, find kernel in file system, load it into memory, start its execution. Now, system is **running**.

Threads vs. processen

Threads en processen zijn concepten die heel belangrijk zijn in een besturingssysteem. Leg deze concepten ui en leg het verschil tussen beide uit. Leg uit hoe een BS deze entiteiten implementeert en beheert. Bespreek daarna alle delen van een BS waar threads en/of processen een rol spelen. Waar mogelijk en zinnig geef je vbn uit Linux en/of Windows XP.

Antwoord

Process Program in execution. Represented in the OS by PCB (process control block).
A process consists out of program code, program counter, registers, stack, data (and heap). A program (passive) becomes a process (active) when its executable file is loaded into memory.
Linux: Process identity: unique id (PID), credentials (user ID, group ID determine access rights), personality, namespace.

Process Control Block State, program counter (next instruction to be executed), CPU registers, CPU-scheduling info (priority, pointers to scheduling queues), memory-management info (base and limit registers, page tables), accounting info, I/O status info (allocated I/O devices, pointer to open-file table).
Linux: Process context: scheduling context, accounting, file table, file-system context, signal-handler table, virtual memory context.

Thread Flow of control within a process.
Uses same address space (code, data, files) as process, but has its own registers and stack. Executes certain set of statements. A process consists out of its address space + at least 1 thread of control.

Process vs thread Much more beneficial to create threads instead of processes (if possible), because of less overhead (economy). Threads use the already existing address space, instead of having to create a new one when creating a new process instead.
Other benefits: responsiveness, resource sharing, scalability.

Multithreading Multiple threads of a process execute different sets of code statements.

Implementation Thread **library** (system calls).
Implicit threading: thread pool

Management: Process scheduling Selecting a waiting process from the ready queue and allocating the CPU to it.
Process enters, put in job queue (on disk). Ready and waiting to be executed: ready queue (in memory). When interrupted or waiting for I/O: waiting queue. List of processes waiting for a particular I/O device: device queue.
Algorithms: FCFS, SJF, RR, Priority scheduling, ...
Long-term scheduler: selects processes from job queue on disk and loads them into ready queue in memory.
Short-term scheduler / CPU scheduler: selects from ready queue and allocates CPU.
Medium-term scheduler: select from swapped out processes and put into ready queue.

Context switch When interrupt occurs, save state, switch to other process, state restore to resume.

Process operations

Creation (by parent process) - parent blocks until child terminates or parent continues concurrently
Linux: `fork()` creates new process without running new program code. Subprocess continues execution where parent left off. `exec()` loads new binary object into process's address space and new executable starts executing in context of existing process.

Termination (finishes executing, resources deallocated, parent terminates child - when exceeded resource usage or task no longer required or parent is exiting) - resources deallocated, return exit status when parent calls `wait()`.

Role of processes / threads IPC, multithreading, synchronization, (virtual) memory-management, I/O, protection.

Interrupts en exceptions

Interrupts en exceptions uitleggen, hoe ze werken, waarom ze belangrijk zijn en voorbeelden geven van hs 16 en 17.

Antwoord

Interrupts Occurrences that induce OS to execute urgent, self-contained routine. Used to handle asynchronous events and to trap to supervisor-mode routines in the kernel.

CPU senses hardware called **interrupt-request line** after each executed instruction.
When CPU detects that a controller has asserted a signal on the interrupt-request line, CPU performs state save and jumps to the **interrupt-handler routine**.
Interrupt handler determines cause of interrupt, performs necessary processing, performs sate restore and executes a return from interrupt instruction to return the CPU to the execution state prior to interrupt.
Device controller *raises* interrupt by asserting signal on interrupt-request line. CPU *catches* interrupt and *dispatches* it to interrupt handler.
Handler *clears* interrupt by servicing device.

Interrupt-controller hardware CPU has two interrupt-request lines.

- Nonmaskable interrupt line: events such as unrecoverable memory errors.
- Maskable interrupt line: can be turned off by CPU before execution of critical instruction sequences that must not be interrupted. (Used by device controllers to request service.)

Interrupt mechanism accepts an address (number that selects a specific interrupt-handling routine, offset in **interrupt vector**). Reduces need for single interrupt handler to search all possible sources of interrupts to determine which one needs service.
Interrupt chaining: each element in interrupt vector points to head of list of interrupt handlers. Interrupt raised, handlers on corresponding list called one by one, until one is found that can service request. Compromise between overhead of huge interrupt table and inefficiency of dispatching to a single interrupt handler.
Interrupt priority levels: enable CPU to defer handling low-priority interrupts without masking all interrupts and makes it possible for high-priority interrupt to preempt execution of low-priority interrupt.

At boot time, OS probes hardware buses to determine what devices are present and installs corresponding interrupt handlers into the interrupt vector. During I/O, various device controllers raise interrupts when they are ready for service. These signify that output has completed, or that input data are available, or that a failure has been detected.
Windows: kernel dispatcher provides trap handling for exceptions and interrupts generated by hardware or software.

Exceptions Software based interrupt caused by an error (e.g.: division by zero, invalid memory access, attempt to execute privileged instruction from user mode).
Interrupts are used to handle exceptions.
Windows: exception dispatcher creates exception record containing reason for exception and finds exception handler to deal with it.

I/O device

Wat gebeurt er wanneer je een scanner (voor de eerste keer) aansluit. Vertel ook wat er gebeurt wanneer een applicatie iets op die scanner wil uitvoeren.

Antwoord

- User process issues system call (with file descriptor and other parameters) through API.
- Interrupt handler in kernel associates with right interrupt routine.
- New device: dynamically load (memory-map) associated device driver from disk.
  Linux: Kernel maintains dynamic tables of all known drivers and provides set of routines to allow drivers to be added to or removed from these tables at any time.
  Kernel calls module's startup routine on load and calls module's cleanup routine before unload.
  These are responsible for registering module's functionality.
  Scanner is character device (kernel passes request to device and lets device deal with request).
  Windows: **I/O manager**: responsible for managing file systems, device drivers and network drivers.
  Keeps track of which device drivers loaded and manages I/O buffers.
  Controls Windows cache manager which handles I/O caching.
  Device drivers arranged in list for each device.
  Driver represented in system as driver object.
  I/O manager packs requests into I/O request packet (IRP).
  Forwards IRP to first driver in targeted I/O stack for processing.
- File descriptor is looked up in device table.
  If entry found, port address available. Else, add new entry for device name with port number it is connected to.
- Kernel I/O subsystem determines whether or not it can already satisfy request (input available in buffer cache).
  IF NOT:
  - sends request to device driver, block process if appropriate
  - Device driver processes request, issues commands to controller (interrupts), configures controller to block until interrupted

  - Device controller monitors device, interrupts when I/O completed
  - Interrupt handler receives interrupt, stores data in device-driver buffer. If input, signal to unblock device driver.
  - Device driver determines which I/O completed, indicates state change to I/O subsystem.

- I/O subsystem transfers data (if appropriate) to process, returns completion or error code.

- Return from system call. I/O completed, input data available to user process, or output completed.

  I/O direction: read-only.

  Kernel controls device: scheduling, buffering, caching, spooling, device reservation, error handling (device driver). Name translation: connection between hardware devices and symbolic file name.

Begrippen verklaren

Verklaar onderstaande begrippen.

Antwoord

Rendez-vous Between sender and receiver when both `send()` and `receive()` are blocking.

Kernel thread Supported and managed directly by the operating system (kernel level).
Vs. user thread: supported above the kernel and are managed without kernel support (user level).

Spooling Spool: buffer that holds output for a device that cannot accept interleaved data streams.
Each application's output is spooled to a separate disk file. Spooling system copies queued spool files to printer one by one. Control interface allows user to edit spooling queue (remove, suspend, ...)

Memory-mapped file Part of virtual address space logically associated with a file.
Mapping a disk block to a page in memory, after page fault, page-sized portion of file is read from file system into frame.
Practical relevance: Subsequent read/writes (not synchronous, has to be written back to disk) handled as routine memory access.

`compare_and_swap()` Takes 3 params: `value`, `expected`, `new_value`.
Always returns `value` of lock. If `value` == `expected`, `value` = `new_value`.
`value` is initialized to 0.
First process to execute `compare_and_swap()` sets `value` to `new_value` (1). This process called `compare_and_swap()` in a while loop, executing as long as `compare_and_swap()` != 0. Second iteration: first process enters critical section as `value` was 0. Other processes can't enter until first process sets `value` of lock back to 0.
Each `compare_and_swap()` is executed **atomically** (uninterrupted).
Satisfies mutual exclusion requirement, but not bounded waiting.

Scheduling queues Ready, waiting, device, ...

POSIX thread Thread library that provides programmer with API for creating and managing threads.

TLB hit A page is found in the TLB (translation look-aside buffer), which has stored the corresponding physical frame.

Practical relevance: Address translation is done much faster, as the page doesn't have to be looked up in the page table in main memory (which is slower than the TLB).

Mandatory file locking

Deadlock prevention Preventing deadlocks from happening by making sure one of the 4 conditions does not hold.
Mutual exclusion: cannot prevent deadlocks by denying mutual-exclusion condition, because some resources are non-sharable.
Hold and wait: whenever process requests resource, it does not hold any other resources.
No preemption: if process holding resources and requests another that cannot be allocated immediately, all resources preempted, process blocked until all requests can be serviced.
OR if process requests resources, check whether available, if they are: allocate, else check whether allocated to other waiting process. If resources neither available nor held, requesting process must wait. Its resources may be preempted by requests from other processes. Process restarts when all requests can be serviced.
Circular wait: impose total ordering on resource types, resource requests only in increasing order or enumeration.

Stealth virus Attempts to avoid detection by modifying parts of the system that could be used to detect it.

Hold and wait When process requests resource, it holds other resources while waiting on its requested resource.

Security vs protection
Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. The goal is to ensure that each program component active in a system uses system resources only in ways consistent with stated policies.
Security is a measure of confidence that the integrity of a system and its data will be preserved.

Logische records in een file

SSTF (**disk scheduling**) Shortest-seek-time-first algorithm selects the request with the least seek time from the current head position.
SSTF chooses pending request closest to current head position. ($\approx$ SJF)
-: starvation, not optimal

Naming bij IPC (**inter-process-communication**) With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.
Symmetry: both sender and receiver name the other to communicate.
Asymmetry: only sender names recipient.

Demilitarized zone *Firewall*: computer, appliance or router in between trused and untrusted systems.
Implementation: internet = untrusted - semitrusted (demilitarized zone) - company's computer. Connections allowed: internet->DMZ and company->internet. Not allowed: internet->DMZ or DMZ->company.
Optionally: controlled communication DMZ->company.
DMZ must be attack proof and secure.

Armored virus Coded to make it hard for antivirus researchers to unravel and understand.
Can also be compressed to avoid detection and disinfection. Files frequently hidden via file attributes or unviewable file names.

Inverted page table A physical address is found by using a process id and offset. The process id is looked up in the inverted page table, the offset at which it is stored is the frame number. The frame number and offset are then used to compute the actual physical address.

File handle File name given to entry in open-file table. (= *file descriptor*)

Symmetric multiprocessing Each processor performs all tasks within the operating system.
Each processor has its own registers, cache and process queue.

Two-level design bij threading Multiplexes many user-level threads to a smaller or equal number of kernel threads, but also allows a user-level thread to be bound to a kernel thread.

External fragmentation As processes are loaded and removed from memory, the free memory space is broken into little pieces.
External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

Mount table Contains information about each mounted volume. Associates prefixes of path names with specific device names.

SCAN-algoritme bij schijven Disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
At the other end, the direction of head movement is reversed, and servicing continues. ($\approx$ *elevator*)
C-SCAN: when arm reaches end, go back to beginning instead of reversing direction. Servicing requests that've been there longer.

MULTICS ringstructuur bij protection
The protection domains are organized hierarchically into a ring structure. Each ring (numbered 0-7) corresponds to a single domain, with ring 0 being the most privileged one.
Each process is associated with a `current-ring-number`.
A process can only access those segments with a `current-ring-number` $>=$ that of the process.
Domain switching occurs when calling a procedure in a different ring. This switching is controlled by the following, being included in the ring field of the segment descriptor:

Access bracket A pair of integers, b_1 and b_2, such that $b_1 <= b_2$.

Limit An integer b_3 such that $b_3 >= b_2$.

List of gates Identifies the entry points (gates) at which the segments may be called.

When is a call allowed?

- A call to a procedure (segment) is allowed if the `current-ring-number` of the calling process is within the access brackets. The calling process holds its initial ring number.
- If `current-ring-number` $< b_1$, call is allowed because we have a transfer to a ring with fewer privileges. If parameters are passed that refer to segments in a lower ring (segments not accessible to the called procedure), then these segments must be copied into an area that can be accessed by the called procedure.
- If `current-ring-number` $> b_2$, call is allowed only if $b_3 >=$ `current-ring-number` and the call has been directed to one of the designated entry points in the list of gates. This allows processes with limited access rights to call procedures in lower rings that have more access rights, but in a controlled manner.

Disadvantage: not enforcing need-to-know principle.

Polymorf virus Polymorphic virus changes each time it is installed to avoid detection by antivirus software.
Changes do not affect virus's functionality, but rather change the virus's signature (pattern that can be used to identify a virus, typically series of bytes that make up virus code).

System call Provide interface to services of OS.
Accessed by programmer through API (system call interface).
Types: process control, file manipulation, device manipulation, information maintenance, communications and protection.

User mode When OS is executing on behalf of a user application, the system is in user mode.
Mode bit: added to hardware, indicates current mode.

Kernel mode When user application requests service from OS (via system call), the system transitions from user to kernel mode to fulfill the request.
Whenever a trap or interrupt occurs, switches to kernel mode. Whenever the OS gains control of the computer, it is in kernel mode.
OS always switches to user mode before passing control to user application.

Round-Robin scheduling A time quantum is set.
FCFS is applied, but after each quantum has passed, the CPU is preempted. The active process becomes last in the ready-queue. The first process in the queue acquires the CPU.

FCFS scheduling The process that enters the ready-queue first, will be given access to the CPU first. Implemented with a *FIFO-queue*.

Priority scheduling The process with the highest priority becomes first in the ready-queue.
Can be *preemptive* or *nonpreemptive*.

File allocation Allocating space to files so that disk space is utilized effectively and files can be accessed quickly.
3 major methods:

- contiguous
  -: external fragmentation (solution: compaction)
  +: easy sequential and direct access
- linked
  -: direct-access inefficient + space required for pointers (solution: clusters) + reliability (solution: doubly linked list)

+: no external fragmentation

- indexed
  -: /
  +: efficient direct-access + no external fragmentation

Microkernel A minimum of tasks is performed in kernel mode (IPC, memory management, CPU scheduling).
Other (system) tasks are performed at the user-level (applications, file system, device drivers).

Dynamic linking Load system libraries only once into memory, multiple processes use same library on same memory location.
Linking postponed to execution time. **Stub** included in image for each lib routine reference. Small piece of code that indicates how to locate appropriate memory-resident lib routine or how to load lib if routine not already present. Stub replaces itself with address of routine and executes it.
Practical relevance: Without dynamic linking, each program must include copy of its language library in executable.

LOOK scheduling *SCAN/C-SCAN*: arm moves across full width of disk.
LOOK/C-LOOK: arm moves only as far as the final request in each direction.
(look, because they 'look' for a request before continuing)

Need-to-know principle
At any time, a process should be able to access *only* those resources that it currently *requires* to complete its task.
This limits the amount of damage a faulty process can cause in the system.
E.g.: a procedure is only allowed access to its own variables and the formal parameters passed to it, a compiler only has access to a well-defined subset of files (source file, listing file, ...)

Virtual machine Running an OS as a user level application.

Mailboxen in IPC With indirect communication, messages are sent to and received from mailboxes (or ports).
Object into which messages can be placed by processes and from which messages can be removed.
Mailbox has unique id.
Two processes can communicate only if they have a shared mailbox.

Translation look-aside buffer (TLB) A piece of high speed memory (faster than main memory) that stores a certain amount of page numbers with their according frame numbers.
Practical relevance: address translation encounters a significant speed-up.

Master file directory (MFD) Indexed by user name or account number, contains entries that point to each UFD (user file directory).
When a user job starts or a user logs in, MFD is searched for associated UFD.

Sector slipping

Lock-key scheme Compromise between access lists and capability lists.
Each object has list of unique bit patterns, called *locks*.
Each domain has list of unique bit patterns, called *keys*.
Process executing in domain can only access object if domain has key that matches lock.
Managed by OS.

Transaction A set of operations that performs a specific task.

Many-to-one model Many user level threads are mapped to one kernel level thread.

Priority interrupts When a process is executing and another process with a higher priority is waiting, preempt CPU and assign CPU to higher priority process.

Hierarchical paging The page table itself is paged.
Practical relevance: Decreases weight of page table.

Acyclic Graph Directories Graph with no cycles.
The same file or subdirectory may be in two different directories. Allows directories to share subdirectories and files.
Any changes by one person are immediately visible to all others in the same shared directory/file.
Implementation: link (pointer to other file or subdirectory). Name of real file included in link. Link is resolved when referenced to locate real file.
Issues:

- searching (multiple paths to same file, don't want to traverse shared structures more than once)
- deletion (when can the space allocated to a shared file be deallocated and reused?)
  Solutions: remove file whenever anyone deletes it (leaves dangling pointers to now nonexistent files), (symbolic links) only remove link, leave

links until attempt to use them (illegal access exception), preserve file until all references are deleted (need to keep count of number of references).

- Ensuring there are no cycles.
  Solution: general graph directory. Problem: garbage collection.

Replay attack Consists of the malicious or fraudulent repeat of a valid data transmission.
E.g.: repeated request to transfer money.

Logical memory space The memory, as viewed by any application.
Addresses are relative and are translated to a physical address either at compile time, load time or execution time.

Sequential access in memory Information in the file is processed in order, one record after the other.
Works on sequential-access devices as well as on random-access ones.

Enhanced second-chance algorithm *Second chance* algorithm: FIFO, but when page selected, inspect reference bit.
If value == 0, proceed replacement, if == 1, give page second chance (clear reference bit, set arrival time to current time) and move on to select next FIFO page. **Enhanced**:
Consider reference and modify bit as ordered pair, 4 classes:

- (0, 0) neither recently used nor modified - best to replace
- (0, 1) not recently used but modified - not quite as good, because page needs to be written back to disk before replacement
- (1, 0) recently used but clean - will probably be used again soon
- (1, 1) recently used and modified - probably will be used again soon, and the page will be need to be written back to disk before replacement

We replace first page in lowest nonempty class.
Difference with non-enhanced: preference to modified pages, reduces I/O.

`test_and_set()` A pointer to a lock is passed as a parameter. The passed value of the lock is returned after the lock is set to true. The `test_and_set()` is called in a while loop, before entering a critical section. If initial value of lock was false, critical section can be entered as `test_and_set()` returned `false`. When the lock was `true`, `test_and_set()` returns `true` and the while loop is executed again, critical section is not entered as other process is in critical section. Each `test_and_set()` is executed **atomically** (uninterrupted). Satisfies mutual exclusion requirement, but not bounded waiting.

Nonpreemptive scheduling Once a process acquires the CPU, it is not interrupted. The next process can only acquire the CPU once the previous process ended and released the CPU.

Circular wait (**bij deadlocks**) Set of waiting processes exists such that the first one is waiting for a resource held by the next one (and so on) and the last one is waiting for a resource held by the first one.

Worst fit Allocate largest hole.
Produces largest leftover hole.
Bad choice, best fit / first fit are better in terms of decreasing storage utilization and time.

Absolute path name Path from the root directory to the specified file, giving the directory names on the path.
Not relative to the current directory (relative path name).

Certificate authority Digital certificates are public keys digitally signed by a trusted party.
This trusted party receives proof of identification from some entity (certificate authority) and certifies that the public key belongs to that entity.
Certificate authorities have their public keys included in web browsers before they are distributed.

Static linking Every process has its own copy of the system libraries.
Syslib treated like any other object module and combined by loader into binary image.

Page replacement When increasing multiprogramming, over-allocated memory (no free frames when page fault occurs).
Swap out process, free all frames, reduce level of MP $\approx$ page replacement.
Need to select victim frame and update associated page and frame tables.
Frame-allocation algorithm and page-replacement.
Page replacement:
FIFO (Belady's anomaly: page-fault rate may increase as the number of allocated frames increases).
Optimal page replacement: LRU.
LRU-approximation: additional-reference-bits, second-chance, enhanced second-chance.
Additional methods: page-buffering (free-frame pool).

Deadlock criteria Criteria that must hold in order to have a possible deadlock.
Mutual exclusion: at least one resource must be nonsharable.

Hold and wait: when process requests resource, it holds other resources while waiting on its requested resource.
No preemption: resources are not preempted once allocated.
Circular wait: set of waiting processes exists such that the first one is waiting for a resource held by the next one (and so on) and the last one is waiting for a resource held by the first one.

Scheduling criteria Criteria for comparing scheduling algorithms.
CPU utilization: keep CPU as busy as possible.
Throughput: number of processes complete per time unit.
Turnaround time: interval from time of submission to time of completion (time spent in job queue + ready queue + executing + I/O).
Waiting time: sum of periods spent waiting in ready queue.
Response time: time it takes to start responding, not time it takes to output respond.

Trap door Hole in the software that only the designer of that program is capable of using.

File Logical storage unit.

Volume Any entity containing a file system.