Chapter 1

1.1 Basic Elements:
1.1.1 Processor: Controls operations of computer and performs data processing functions. CPU(Central Processing Unit)
1.1.2 Main Memory: Stores Data and programs (volatile).
1.1.3 I/O Modules: Move data between computer and external environment
1.1.4 System Bus: Communication among processors, main memory, I/O Modules

MAR (Memory Address Register): Specifies address in memory for next read or write
MBR (Memory Buffer register): Contains data to be written into memory of which receives data read from memory;
I/OAR: Specifies a particular I/L Device
I/OBR: Used for exchanging data between an I/O Module and processor
I/O module: transfers data from external devices to processor and memory

1.2 Processor Registers:
User-visible Register: Avoids main memory references by optimizing register use (C-Compiler)
Control and Status registers: used to control operation of processor, for controlling execution of processors

User-Visible Registers:
Data Registers: can be assigned to a variety of functions by the programmer (distinguish between floating-point/integer registers
Address Registers: contain main memory addresses of data and instructions
Index register: calculating an address by adding an index to a base value
Segment Pointer: Memory divided into segments -> memory reference = reference to particular segment and offset within the segment
Stack pointer: dedicated register-> points to the top of the stack (push, pop)
Differences: subroutine call will result in automatic saving of all user-visible registers or not

Control and Status Registers:
Program Counter: (PC) Contains address of the next instruction to be fetched
Instruction Register (IR): Contains instruction most recently fetched
PSW: Program Status Word: Contains status information -> contains condition codes and SREG and so
Condition Codes: Typically set by processor as result of operations.
Interrupt registers: one pointer to each interrupt-handling routine
OS support: certain types of control information are of specific utility to the operating system

1.3 Instruction Execution:
Consists of two steps: fetching instructions from memory one at a time -> executing each instruction -> Instruction Cycle (processing required for a single instruction)
Fetch State and execute stage

Instruction Fetch and Execute:
Processor fetches instruction from memory -> PC holds address of the next instruction to be fetched -> loaded into instruction register (IR): bits that specify action the processor is to take:
Action can be divided into four categories:
Processor-memory: Data may be transferred from processor to memory or from memory to processor
Processor-I/O: Data may be transferred to or from a peripherical device

Data-processing: processor may perform arithmetic or logic operation on data
Control: instruction may specify that the sequence of execution is altered

AC (Accumulator): Processor contains a single data register
I/O Function: Data can be exchanged directly between an I/O module and processor
Direct exchange between memory and I/O task -> DMA

1.4 Interrupts:

Classes for interrupts: Program, Timer, I/O, Hardware failure
Used to avoid waiting for slower I/O devices

Interrupts and the Instruction Cycle:
Processor can be enganged in executing other intstructions while an I/O operation is in progress
I/O Program: consists only of the preparation code and actual I/O command -> control returns to program while external device is busy -> when finished -> interrupt request signal to processor by device -> suspends operation of current program -> branches off to interrupt handler
(Interrupt stage is added to instruction cycle)
Problem: if I/O-routine is to be executed while external devices isn't ready -> program hangs

Interrupt Processing:

Device completes I/O-operation:

1. device issues an interrupt signal to the processor.
2. processor finishes execution of the current instruction before responding to the interrupt
3. Processor tests for pending interrupt request -> acknowledgment send to device -> acknowledgment allows device to remove interrupt signal
4. Processor transfers control to interrupt routine
5. Processor loads program counter with entry location of interrupt handling routine
6. interrupt handler -> saves all registers on the stack, stack pointer updated, PC updated
7. Interrupt handler -> processes interrupt
8. Interrupt processing complete -> saved register restored
9. restore PSS and PC

Multiple Interrupts:
Approaches:

1. Disable Interrupts while an interrupt is being processed -> pending interrupts will be handled afterwards
2. define Priorities for interrupts

Multiprogramming:
If time required to complete I/O operation is much greater than code between I/O calls -> proc is idle: solution: have multiple user programs active at the same time
If interrupt occurs: -> when interrupt handling is finished -> pass control to some other pending program with a higher priority

1.5 Memory Hierarchy:
2 key characteristics: cost, capacity, access time
memory hierarchy solves problem of greater capacity and slower access speed -> going down this hierarchy:

a. Decreasing cost per bit
b. Increasing capacity
c. Increasing access time
d. Decreasing frequency of access to the memory by the processor

➔ KEY for hierarchy: decreasing frequency of access
Hit Ratio: fraction of all memory accesses that are found in the faster memory (cache)
(hit = accesses word found in cache, else: miss)
Locality of reference: (basis for d.): occurs in loops for example
Possible to organize data across hierarch in the ordering of the frequency of accesses
Level 2: contains all program instructions and data
Level 1: current clusters can be temporarily placed here
Main memory often expended with smaller high-speed cache
Secondary Memory/Auxiliary Memory: External, non-volatile memory
In Software: Disk writes are clustered

1.6 Cache Memory:
Motivation:
Different speed processor -> main memory -> solution: cache (fast memory between processor and main memory)

Cache Principles:
Large, slow main memory
Small, fast cache, contains copy of portion of main memory
Attempt to read a byte -> processor checks whether byte in cache or not
Main memory consists of up to 2^n addressable words -> memory consists of a number of fixed-length blocks of K words each.
Cache consists of C slots of K words each -> contains subset of blocks of main memory
If cache miss occurs -> transferred into cache (contains tag which identifies which block is stored)

Cache Design:
Cache Size: small caches -> better performance
Block Size: unit of data exchanged between cache and main memory
Mapping function: determines which cache location the block will occupy
Replacement algorithm: chooses, which block to replace when a new block is to be loaded into the cache and cache is full -> LRU algorithm
Write policy: if contents of block in cache is altered -> necessary to write it back to main memory

1.7 I/O Communication Techniques:

Programmed I/O:
When encountering I/O operation -> executes instruction by calling appropriate I/O module.
-> IO module performs action and sets bits in the I/O status register (processor periodically checks status of I/O module.

Control: used to active external device
Status: used to test various status conditions
Transfer: User to read and/or write data between processor registers and external devices

Interrupt-driven I/O:
Processor issues command to a module and then go on to do some other work
I/O module interrupts processor to request service

Direct memory Access (DMA):
Interrupt driven I/O -> Processor responsible for transferring data between memory and an I/O module

Drawback:

1) I/O transfer rate is limited by the speed of the processor
2) The processor is tied up in managing I/O transfer

➔ but useful when large modules of data are to be moved

DMA module needs:

Whether read or write is requested
The address of the I/O device involved
Starting location in memory to read data from or write data to
Number of words to be read or written

DMA module needs to take control of the bus to transfer data to and from memory

➔ processor does not save context

Stack pointer: Contains address of the top of the stack
Stack base: Contains address of bottom location in reserved block
Stack limit: contains end of the possible stack

Chapter 2

2.1 Operating System Objectives and Functions:
Convenience: An operating system make a computer more convenient to use
Efficiency: An operating system allows the computer system resources to be used in an efficient manner
Ability to evolve: should be able to to permit development, testing, and introducing new system functions

OS as User/Computer Interface
Should mask details of the hardware
Os Provides Services:
Program development:
Editors, debuggers, utility programs
Program execution:
Instructions and data must be loaded into main memory, I/O devices initialized
Access to I/O devices:
Each I/O device requires its own set of instruction -> OS provides uniform Interface
Controlled access to files:
Protection mechanisms, must reflect understand of the nature of the I/O device as well structure of data
System access:
Controls access to system resources
Error detection and response:
Accounting: collect usage statistics for various resources and monitor performance parameters

OS as Resource Manager:
OS' control mechanism unusual in two respects:
- OS functions like normal software
- OS frequently relinquishes control and must depend on the processor to regain control

Kernel / Nucleus: Contains most frequently used functions in the OS and other portions of it

Ease of Evolution:
Hardware upgrade:
New Services: OS has to expand to offer new services
Fixes: fixes are introduced as time passes by

Evolution of OS:
Serial Processing:
Mainly used in 40's-50's, Problems of Scheduling and setup time
Simple Batch Systems:
Introduction of a piece of software: Monitor, user has no longer direct access to the machine

e. batched by computer-operator -> when finished -> branched back to monitor
f. Monitor point of view: controls sequence of events -> part of monitor must always reside in main memory: resident monitor; rest of it consists of utilities and common functions that are loaded as subroutines to the user program; monitor reads in one job at a time -> when job completed -> returns control to the monitor
g. Processor point of view: Sometime, the processor is executing instructions from the portion of main memory containing the monitor; once job is read in -> processor will be instructed to go to the start of the user program

JCL (Job Control Language): with each job, instructions are included in a primitive form of JCL -> used to provide instructions to the monitor

Monitor (batch OS): simple computer program.
3 Memory protection: executing user program mustn't alter memory area containing monitor
4 Timer: used to prevent a single job from monopolizing the system
5 Privileged instructions: Certain machine-level instructions can only be executed by the monitor
6 Interrupts: flexibility in relinquishing control to and regaining control from user pr.
User Mode: certain areas of memory are protected from the user's use and in which certain instructions may not be executed
Kernel Mode: privileged instructions may be executed in which protected areas of memory may be accessed

Multiprogrammed Batch Systems:
Problem with automatic job sequencing: Processor often idle
Uniprogramming: just one program at a time -> blocks at I/O operation
Multiprogramming/Multitasking: While waiting for I/O operation monitor can switch to another program

h. relies on computer HW features: I/O interrupts, DMA

Time-Sharing Systems:
Multiprogramming can also be used to handle multiple interactive jobs -> time sharing, because processor time is shared among multiple users.
CTSS (Compatible Time-Sharing System):
At each clock interrupt -> control assigned to another processor -> loaded into memory.
(Partial loading: if a piece of the Job was still in the main memory -> only part of job is loaded)

2.3 Major Achievements:
Processes:
Definitions:
7 a Program in execution
8 an instance of a program running on a computer
9 entity that can be assigned to and executed on a processor
10 a unit of activity by a single sequential thread
Three Major lines of development:
1. Multiprogramming -> designed to keep processor and I/O devices simultaneously busy
2. General-purpose time sharing:
3. Real-time transaction processing systems: number of users are entering queries or updates against a database.
-> principal tool at the beginning: Interrupt (activity could be suspended by occurrence of a defined event) -> processor would branch to interrupt-handling routine
Errors were caused:
11 Improper Synchronization: can lead to signals being lost or duplicate signals being received
12 Failed mutual exclusion: if simultaneous accesses occur on the same data -> errors
13 Nondeterminate program operation: caused by interfering of processes with each other
14 Deadlocks: two programs hung up waiting for each other

Process consists of three components:
An executable program
The associated data needed by the program
The execution of the program (internal data by which the OS is able to supervise and control the process) -> context: contains everything the OS needs for managing the process
Base and limit registers define region in memory occupied by the process.
Base register: starting address of the region of memory and limit = size of the region
PC is interpreted relative to the base register and must not exceed the value in the limit register.

Process realized as a data structure; can either be executing or awaiting execution.
Entire State of the process: contained in its context

Memory Management:
Process isolation:
OS must prevent independent processes from interfering with each other's memory
Automatic allocation and management:
Programs should be dynamically allocated across the memory hierarchy as required.
Support of modular programming:
Programmers should be able to create, destroy, resize modules dynamically
Protection and access control:
OS must provide ways to allow accessing memory by various users
Long-term storage:

Virtual memory: allows programmers to address memory from a logical point of view
Paging systems: pages (fixed-size block) ->
Virtual Address: consisting of a page number + offset within page
Paging system provides dynamic mapping: between virtual address used in the program and a real address -> physical address
All pages of a process are maintained on disk. When process is executing -> some of its pages in main memory, some are not -> if not -> missing page reloaded
Virtual processor: has access to a virtual memory (may be a linear address space or a collection of segments)
Process Isoliatoin achieved by giving each process a unique nonoverlapping virtual memory.
Storage consists of directly addressable main memory and lower-speed auxiliary memory that is accessed indirectly by loeading blocks into main memory.
Prorams reference locations using virtual addresses -> mapped into real main memory

Information Protection and Security:
Availability: Concerned with protecting system against interruption
Confidentiality: Users cant read data they shouldn't
Data Integrity: Protection of data from unauthorized modification
Authenticity: Proper verification of identity of users and validity of messages

Scheduling and Resource Management:
Fairness: all processes have same chance to access resource
Differential responsiveness: OS has to discriminate between classes of jobs
Efficiency: OS should attempt to maximize throughput

OS maintains number of queues: list of processes waiting for some resource.
Short-term scheduler picks a process out of this list (round-robin technique)
Or priority levels.
Long-therm queue: list of new jobs waiting to use the processor
I/O queue for each I/O device

System Structure:
SW be modular

2.4 Developments leading to modern OS
Monolithic Kernel: larger kernels, which includes scheduling, file system, networking, device drivers….
Microkernel Architecture: only a few essential functions to the kernel

Multithreading: process divided into threads which run concurrently
Thread: dispatchable unit of work
Process: collection of one or more threads and associated system resources.
Symmetric Multiprocessing (SMP):
1. There are multiple processors
2. Theses processors share the same main memory and I/O facilities, interconnected by a communication bus
3. All processors can perform the same functions (=symmetric)

Chapter 3

Chapter3:
Trace: Listing of instructions that execute fort hat process
Dispatcher-Program: switches between different processes
Process Control Block: information related to each process, including current sate and location in memory
Process spawning: when the OS creates a new process at the behalf of another process
State transitions:
Null -> New: new Process created
New -> Ready: OS takes process to state ready when it is prepared to take on an additional process
Ready->Running: OS chooses one process to be in the ready state
Running -> Exit: running process has terminated
Running -> Ready: process has reached maximum time to execute (pre-emption occurs when a higher prioritised process needs the CPU)
Running -> blocked: process is waiting for something it requested
Blocked->Ready: when the event the process is waiting for occurs

Need4Swapping:
When there are to many processes in the main memory, which are all in the state blocked -> some of them are put onto disk (swapping) into suspend queue. -> new State (suspend)

i. 4 states:
    1) Ready: Proc in main memory and available for execution
    2) Blocked: Proc in main memory and waiting for an event
    3) Blocked/Suspend: Process in secondary memory and awaiting an event
    4) Ready/Suspend: process in secondary memory -> available for execution as soon as it is loaded into main memory

OS control Structures:
Memory Tables: used to keep track of both main and secondary memory.
15 allocation of main memory to processes
16 the allocation of secondary memory to processes
17 any protection attributes of blocks of main or VM such as which processes may access certain shared memory regions
18 any information needed to manage VM

I/O Tables: used by OS to manage I/O devices and channels ->
File Tables: provide information about existence of files and their location on secondary memory + status + attributes
Process tables: for managing processes.
Collection of attributes: process control block
Program + data+ stack + attributes = process image
[…]
Process-Based Operating System:
OS implemented as collection of system processes
Major kernel functions organized as separate processes

Unix SVR4 Process Management:
Two categories of processes:
System processes: run in kernel mode
User processes: run in user mode

Process States:

2 running states: user mode / kernel mode (table 3.9)
Process Description:
Process in Unix rather complex set of data structures (user-level, register, system-level context)
User-level context: contains basic elements of user's program and can be generated directly from a compiled object file – text area: hold program's instructions

Process Control:
Fork:

- ➔ Allocates a slot in process table for new process
- ➔ assigns unique process ID to child process
- ➔ makes copy of process image of parent
- ➔ increments counters for any files owned by the parent
- ➔ child -> ready to un
- ➔ returns ID number of child to parent process

Chapter 4

4.1)

- Resource ownership: OS performs protection function to prevent unwanted interference between processes with respect to resources
- Scheduling/execution: execution of a process follows execution path; process has execution state + dispatch priority

Unit of dispatching: thread, lightweight process
Unit of resource ownership: process, task

Multithreading:
Refers to ability of OS to support multiple threads of execution within single process
Process defined as unit of resource allocation and a unit of protection
19 a virtual address space that holds the process image
20 protected access to processors, other processes, files and I/O resources
Within a process, there may be one or more threads with the following:
21 A thread execution state
22 A saved thread context when not running
23 Execution stack
24 Per-thread static storage for local variables
25 Access to memory and resources of its process
Single threaded process model:
26 control block + user address space = representation of a process
27 user + kernel stacks
while process is running: processor registers are controlled by that process
Multithreaded: separate stacks for each thread

j. all threads of a process share its state and resources
k. Advantages: Foreground/Background work, Asynchronous processing, Speed, modularity

Thread Functionality:
Thread have states and operations:
Spawn: when process is spawned -> thread for that process is also spawned
Block: When thread needs to wait for an event, it will block -> processor now can turn to another thread which is in ready state
Unblock: thread moved to ready queue
Finish: register context and stacks are deallocated

Thread Synchronization:
Needed because threads shouldn't interfere with each other

User-Level and Kernel-Level Threads:
User-Level threads(ULTs):

Thread management done by application, kernel not aware of existence of threads
At any time application may spawn a new thread -> done by threads library (restores context when switching between threads)
Advantages: thread switching does not require kernel mode privileges
Scheduling: application specific
ULTs can run on any OS
Disadvantages:
When ULT executes system call -> all threads in a process are blocked
Multithreaded application cant take advantage of multiprocessing

l. solution: jacketing (convert blocking system call into non-blocking system call)
m. thread calls an application level I/O jacket routine -> checks whether I/O device is busy

Kernel-Level Threads:
No thread management code but API to kernel thread facility
28 Kernel can simultaneously schedule multiple threads from the same process on multiple processors
29 If thread is blocked, kernel can switch over to another one
30 Disadvantage: mode switch when switching between threads

Combined:
Thread creation in user space -> multiple ULTs from single application mapped onto some KLTs.

Other Arrangements:
Traditionally: 1:1 relationship between threads and processes
Multiple threads within single process: many-to-one relationship
Many-to-many relationship … p. 170

Many-2-Many:
➔ Entire program can be implemented as a single process
➔ main program and I/O subprogram can be implemented as two separate processes
➔ main program and I/O subprogram treated as single activity

One-to-Many:
Clouds: Thread: unit of activity from user's perspective
Process: virtual address space with an associated process control block
Threads actually span machine boundaries

Symmetric Multiprocessing:
Single Instruction Single Data (SISD) stream: single processor executes a single instruction stream to operate on data stored in a single memory.
Single Instruction multiple Data (SIMD) stream: single instruction controls the simultaneous execution of processing elements
Multiple Instruction single data (MISD) stream: sequence of data is transmitted to a set of processors.
Multiple instruction multiple data (MIMD) stream: A set of processors simultaneously execute different instruction sequences on different data sets.

Shared-memory multiprocessor: if processors share a common memory, each processor accesses this memory.
Master/Slave architecture: OS kernel always runs on a particular processor, others are allowed to execute user programs
Master responsible for scheduling processes and threads
Slave needs service -> sends request -> waits for it until performed
n. Drawbacks: single-point-of-failure
o. Master is performance bottleneck

Symmetric multiprocessor:
Kernel can execute on any processor -> each processor does self-scheduling
Kernel can be constructed as multiple processes or multiple threads/processes
p. must avoid having two processors doing the same process
q. processes mustn't get lost from the queue

SMP Organization:
Multiple processors, each one having its own control unit, arithmetic-logic unit, registers
Shared Memory, I/O devices
Shared bus
Cache in each processor -> must be consistent

Multiprocessor OS System Design Considerations:
Design Issues:
Simultaneous concurrent processes or threads: kernel routines must be executable simultaneously
Scheduling: performed by any processor
Synchronization: effective synchronization because of the multiple active processors, having potential access to shared address space or shared I/O resources.
Memory management: EG. OS must support available HW parallelism
Reliability and fault tolerance: OS should provide graceful degradation in the face of processor failure

4.3 Microkernels:
= small OS core: provides basics for modular extensions.
Architecture:
Monolithic OS:
Layered OS: functions are organized hierarchically and interaction only takes place between adjacent layers.
Microkernel: only absolutely essential core OS functions should be in the kernel

r. less essential services and applications are built on the microkernel -> execute in user mode.
s. OS components external to microkernel -> implemented as server processes (comm. By messages)

Uniform interface on requests made by a process
Extensibility: allows addition of new services
Flexibility: existing features can be changed
Portability: all or at least much of the processor-specific code is in the microkernel
Reliability: achieved by APIs
Distributed system support: clusters controlled by DOS
Object-oriented operating system: -> design of microkernel is oriented at OO-Standards

Microkernel Performance:
Message communication makes big efforts-> disadvantage
Applies only to first generation microkernel -> reintregration of critical server and drivers
Increasing functionality -> reduces number of user-kernel mode switches and address-space process switches.

Low-Level Memory Management:
Protection at the process level:
Concept of paging and virtual memory management: -> external pager: -> thread references a page not in main memory -> page fault -> traps to kernel -> sends message to pager -> allocates frame and gets data -> resume message to application (Grant, Map, Flush)

Interprocess Communication: message includes header + body; port = queue of messages destined for a particular process;
I/O and Interrupt Management:
HW interrupts handled as messages; microkernel generates message for the user-level process

associated with the interrupt

4.4 Windows thread and smp management:

Makes use of two types of process-related objects:
Processes: entity corresponding to a user job or application
Threads: dispatchable unit of work that executes sequentially and is interruptible

Multithreading:
Multithreaded process achieves concurrency without the overhead of using multiple processes

Support for OS Subsystems:
Application sends request to subsystem which forwards this request to the executive -> subsystem returns information to application.

Symmetric Multiprocessing Support:
In absence of affinity restrictions -> microkernel assigns a ready thread to next available processor
Default: soft affinity -> dispatcher tries to assign ready thread to the same processor it last ran on
More restricted variant: Hard affinity

4.5 Solaris Thread and SMP Management:
Multithreaded Architecture:
Process: normal UNIX process -> includes user's address space, stack and process control block
User-level threads: threads library -> in address space of a process -> invisible to OS
Lightweight processes: LWP, mapping between ULTs and kernel threads.
Kernel threads: fundamental entities that can be scheduled and dispatched

t. exactly one kernel thread for each LWP
u. Ults: are controlled by single kernel thread, only one can be executing at a time
v. Advantage: ults only known by application -> cheaper switching

Process Structure:
Unbound threads: threads that share a number of LWPs (runnable, active, sleeping, stopped)
Events can occur:
Synchronization:
w. Thread is placed in sleeping state -> afterwards in runnable state
Suspension:
x. Any thread may cause T1 to be suspended in the stopped state
Preemption:
y. An active thread does something that causes another thread of higher priority become runnable
Yielding:
z. used when there are two processes of the same priority

Interrupts as Threads:
For synchronization between kernel threads and interrupts -> the latter are converted to kernel threads
(reduce overhead) -> interrupts must be blocked
Solaris:
10. Employs set of kernel threads to handle interrupts
11. kernel controls access to data structures and synchronizes

12. Interrupt threads are assigned higher priorities
aa. interrupt occurs -> delivered to processors, which current process is pinned (hold on the processor)

4.6 Linux Process and Thread Management:
Linux Tasks:
Process or task represented by task_struct data structure, contains:
State: execution state of the process
Scheduling Information: real time / priority
Identifiers: unique process identifier, user and group identifier
Interprocess communication: IPC mechanisms
Links: link to parent process, links to its siblings, links to children
Times and timers: process creation time, consumed processor time,
File system: pointers to any files opened by process, current dir, root directory
Address space: defines virtual address space assigned to this process
Processor-specific context: registers and stack information that constitute context of a process

Linux Threads:
Pthread libraries needed for writing user-level multithreaded applications -> all threads mapping into a single kernel-level process
Clone() command used to create processes (fork() with special flags)

Chapter 5

Multiprogramming: management of multiple processes within a uniprocessor system
Multiprocessing: management of multiple processes within a multiprocessor.
Distributed processing: management of multiple processes executing on multiple, distributed computer systems.

Concurrency in three different contexts:
Multiple applications: dynamically sharing of processing time among applications
Structured Applications: appl. can be effectively programmed as a set of concurrent proceses
Operating system structure: OS often implemented as a set of processes or threads

Interleaving:

Overlapping:

Difficulties:
Order of execution of processes important

Process Interaction:
Processes unaware of each other: -> independent processes, not working together -> OS must be concerned about competition
Processes indirectly aware of each other: two processes sharing same resources (cooperation)
Processes directly aware of each other: are able to communicate with each other by PID (cooperation)

Competing processes: 3 control problems:
Mutual exclusion: needed when two processes need access to critical resource

- ➔ problem of deadlock:
- ➔ starvation: always access denied for a process

Cooperation among Processes by Sharing:
Interaction between processes not being directly aware of each other

- ➔ problem of data coherence

Cooperation among Processes by Communication:
Communication done by messages -> but problem of deadlocking, starvation

Requirements for Mutual Exclusion:

1. only one process at a time is allowed to be in its critical section
2. a process that halts in CS mustn't block other processes
3. no deadlock or starvation

5.2 Mutual Exclusion: HW support:
Interrupt Disabling:

- ➔ only works in uniprocessing environment

Special Machine Instructions:

- ➔ machine Instr. Which carry out two actions atomically
- ➔ Test and Set Instruction
- ➔ Exchange Instruction

Semaphores:

- ➔ special variables for process communication (signals)
- ➔ operations on that variable: initialization, semWait for decrementing, semSignal for incrementing semaphore value
- ➔ atomic
- ➔ FIFO queue: strong semaphore
- ➔ Semaphore which does not specify order in which processes are removed: weak semaphore
- ➔ Strong semaphores guarantee freedom from starvation

Mutual Exclusion:

- ➔ achieved by using semaphores

The Producer/Consumer Problem:

- ➔ one or more producers generating some type of data -> placing these in buffer -> single consumer is taking items out of it

Implementation:

- ➔ semWait and semSignal atomic -> only one process at a time manipulate a semaphore with either a semWait or semSignal operation

Monitors:

- ➔ programming language construct that provides equivalent functionality to that of semaphores

Monitor with Signal:

SW module with one or more proceducres, initializing sequence and local data.

3. Local data variables only accessible by monitor's procedure
4. A process enters monitor by invoking one of its procedures
5. only one process in the monitor at a time

- ➔ protection mechanism
- ➔ supports synchronization: condition variables: cwait(c) -> causes process to be placed into queue, csignal(c) -> detects change in condition variable
- ➔ Alternate Model: Notify / Broadcast:
  - At least one process in a condition queue -> a process from that queue runs immediately when another process sends csignal for that condition
  - Latter one must immediately exit or blocked
  - Cnotify(x), causes x condition queue to be notified
  - Cbroadcast causes all processes waiting on a condition to be placed in a Ready State.

f. Message Passing:

Done by send(destination,message)
Receive(soure,message)

Synchronization:

When a process issues a receive primitive:

bb. If a message has presviously been sent -> message received

cc. if there is no waiting message then a) process is blocked until message arrives b) process continues to execute, abandoning the attempt to receive

1) Blocking send, blocking receive
2) Nonblocking send, blocking receive
3) Nonblocking send, nonblocking receive

Addressing:

Direct addressing: send contains specific identifier

Indirect addressing: messages are not sent directly from sender to receiver -> sent to shared

data structure consisting of queues
13. ono-to-one relationship: private communication link
14. many-to-one relationship: useful for client/server interactoin
15. one-to-many relationship: broadcasting
16. many-to-many relationship: concurrent services

Message Format:
Consists of header + body
Queuing Discipline:
FIFO, priority…
Mutual Exclusion:
Process wishing to enter critical section -> attempts to receive a message
If mailbox empty -> process blocked; process acquires message -> performs critical section
If there is a message -> delivered to only one process, others are blocked
If message queue empty -> all processes are blocked

Readers/Writers problem:
Data area shared among a number of processes ->
31 Any number of readers may simultaneously read the file
32 Only one writer at a time may write to the file
33 if a writer is writing to the file, no reader may read it
Readers have priority: semaphores are used to enforce mutual exclusion
Writers have priority: