Table 1.2 Characteristics of Two-Level Memories

	Main Memory Cache	Virtual Memory (Paging)	Disk Cache
Typical access time ratios	5 : 1	1000 : 1	1000 : 1
Memory management system	Implemented by special hardware	Combination of hardware and system software	System software
Typical block size	4 to 128 bytes	64 to 4096 bytes	64 to 4096 bytes
Access of processor to second level	Direct access	Indirect access	Indirect access

Table 1.3 Relative Dynamic Frequency of High-Level Language Operations

Study	[HUCK83]	[KNUT71]	[PATT82]		[TANE78]
Language	Pascal	FORTRAN	Pascal	C	SAL
Workload	Scientific	Student	System	System	System
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Other	—	7	6	1	6

Table 1.1 Classes of Interrupts

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Table 10.1 Synchronization Granularity and Processes

Grain Size	Description	Synchronization Interval (Instructions)
Fine	Parallelism inherent in a single instruction stream.	<20
Medium	Parallel processing or multitasking within a single application	20-200
Coarse	Multiprocessing of concurrent processes in a multiprogramming environment	200-2000
Very Coarse	Distributed processing across network nodes to form a single computing environment	2000-1M
Independent	Multiple unrelated processes	(N/A)

Table 10.2 Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

Table 10.3 Execution Profile of Five Aperiodic Tasks

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

Table 10.4 Value of the RMS Upper Bound

n	$n(2^{1/n} - 1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
•	•
•	•
•	•
∞	$\ln 2 \approx 0.693$

Table 10.5 Execution Profile for Problem 10.1

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
• • •	• • •	• • •	• • •
B(1)	0	10	50
B(2)	50	10	100
• • •	• • •	• • •	• • •
C(1)	0	15	50
C(2)	50	15	100
• • •	• • •	• • •	• • •

Table 10.6 Execution Profile for Problem 10.2

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	100
B	20	20	30
C	40	20	60
D	50	20	80
E	60	20	70

Table 11.2 Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Table 11.4 RAID Levels

Category	Level	Description	I/O Request Rate (Read/Write)	Data Transfer Rate (Read/Write)	Typical Application
Striping	0	Nonredundant	Large strips: Excellent	Small strips: Excellent	Applications requiring high performance for noncritical data
Mirroring	1	Mirrored	Good/Fair	Fair/Fair	System drives; critical files
Parallel access	2	Redundant via Hamming code	Poor	Excellent	
	3	Bit-interleaved parity	Poor	Excellent	Large I/O request size applications, such as imaging, CAD
Independent access	4	Block-interleaved parity	Excellent/Fair	Fair/Poor	
	5	Block-interleaved distributed parity	Excellent/Fair	Fair/Poor	High request rate, read-intensive, data lookup
	6	Block-interleaved dual distributed parity	Excellent/Poor	Fair/Poor	Applications requiring extremely high availablity

Table 11.1 I/O Techniques

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Table 11.3 Disk Scheduling Algorithms [WIED87]

Name	Description	Remarks
	Selection according to requestor	
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
	Selection according to requested item	
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load sensitive

Table 11.5 Device I/O in UNIX

	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X

Table 11.6 Physical Characteristics of Disk Systems

Head Motion	**Platters**
Fixed head (one per track)	Single platter
Movable head (one per surface)	Multiple platter
Disk Portability	**Head Mechanism**
Nonremovable disk	Contact (floppy)
Removable disk	Fixed gap
	Aerodynamic gap (Winchester)
Sides	
Single sided	
Double sided	

Table 11.7 Typical Disk Drive Parameters

Characteristics	**Seagate Cheetah 36**	**Western Digital Enterprise WDE18300**
Capacity	36.4 GB	18.3 GB
Minimum track-to-track seek time	0.6 ms	0.6 ms
Average seek time	6 ms	5.2 ms
Spindle speed	10000 rpm	10000 rpm
Average rotational delay	3 ms	3 ms
Maximum transfer rate	313 Mbps	360 Mbps
Bytes per sector	512	512
Sectors per track	300	320
Tracks per cylinder (number of platter surfaces)	24	8
Cylinders (number of tracks on one side of platter)	9801	13614

Table 11.8 Optical Disk Products

CD

Compact Disk. A nonerasable disk that stores digitized audio information. The standard system uses 12-cm disks and can record more than 60 minutes of uninterrupted playing time.

CD-ROM

Compact Disk Read-Only Memory. A nonerasable disk used for storing computer data. The standard system uses 12-cm disks and can hold more than 650 Mbytes.

CD-R

CD Recordable. Similar to a CD-ROM. The user can write to the disk only once.

CD-RW

CD Rewritable. Similar to a CD-ROM. The user can erase and rewrite to the disk up to 1000 times.

DVD

Digital Versatile Disk. A technology for producing digitized, compressed representation of video information, as well as large volumes of other digital data. Both 8 and 12 cm diameters are used, with a double-sided capacity of up to 15.9 Gbytes. The basic DVD is read-only (DVD-ROM).

DVD-R

DVD Recordable. Similar to a DVD-ROM. The user can write to the disk only once.

DVD-RW

DVD Rewritable. Similar to a DVD-ROM. The user can erase and rewrite to the disk up to 1000 times.

Magneto-Optical Disk

A disk that uses optical technology for read and magnetic recording techniques assisted by optical focusing. Both 3.25-inch and 5.25-inch disks are in use. Capacities above 5 Gbyte are common.

Table 12.1 Grades of Performance for Five Basic File Organizations [WIED87]

	Space		Update		Retrieval		
	Attributes		Record Size				
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excellent, well suited to this purpose ≈ $O(r)$
B = Good ≈ $O(o \times r)$
C = Adequate ≈ $O(r \log n)$
D = Requires some extra effort ≈ $O(n)$
E = Possible with extreme effort ≈ $O(r \times n)$
F = Not reasonable for this purpose ≈ $O(n^{>1})$

where
r = size of the result
o = number of records that overflow
n = number of records in file

Table 12.3 File Allocation Methods

	Contiguous	Chained	Indexed	
Pre-Allocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

Table 12.2 Information Elements of a File Directory

Basic Information	
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
Address Information	
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Table 12.4 Information in a UNIX Disk-Resident Inode

File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

Table 12.5 Capacity of a UNIX File

Level	Number of Blocks	Number of Bytes
Direct	10	10K
Single Indirect	256	256K
Double Indirect	256 × 256 = 65K	65M
Triple Indirect	256 × 65K = 16M	16G

Table 12.6 Windows NTFS Partition and Cluster Sizes

Volume Size	Sectors per Cluster	Cluster Size
≤ 512 Mbyte	1	512 bytes
512 Mbyte - 1 Gbyte	2	1K
1 Gbyte - 2 Gbyte	4	2K
2 Gbyte - 4 Gbyte	8	4K
4 Gbyte - 8 Gbyte	16	8K
8 Gbyte - 16 Gbyte	32	16K
16 Gbyte - 32 Gbyte	64	32K
> 32 Gbyte	128	64K

Table 12.7 Windows NTFS File and Directory Attribute Types

Attribute Type	Description
Standard information	Includes access attributes (read-only, read/write, etc.); time stamps, including when the file was created or last modified; and how many directories point to the file (link count).
Attribute list	A list of attributes that make up the file and the file reference of the MFT file record in which each attribute is located. Used when all attributes do not fit into a single MFT file record.
File name	A file or directory must have one or more names.
Security descriptor	Specifies who owns the file and who can access it.
Data	The contents of the file. A file has one default unnamed data attribute and may have one or more named data attributes.
Index root	Used to implement folders.
Index allocation	Used to implement folders.
Volume information	Includes volume-related information, such as the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or folder.

Note: shaded rows refer to required file attributes; the other attributes are optional.

Table 13.2 Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Table 13.1 Client/Server Terminology

Applications Programming Interface (API)
A set of function and call programs that allow clients and servers to intercommunicate

Client
A networked information requester, usually a PC or workstation, that can query database and/or other information from a server

Middleware
A set of drivers, APIs, or other software that improves connectivity between a client application and a server

Relational Database
A database in which information access is limited to the selection of rows that satisfy all search criteria

Server
A computer, usually a high-powered workstation, a minicomputer, or a mainframe, that houses information for manipulation by networked clients

Structured Query Language (SQL)
A language developed by IBM and standardized by ANSI for addressing, creating, updating, or querying relational databases

Table 14.1 Distributed Deadlock Detection Strategies

Centralized Algorithms		Hierarchical Algorithms		Distributed Algorithms	
Strengths	**Weaknesses**	**Strengths**	**Weaknesses**	**Strengths**	**Weaknesses**
•Algorithms are conceptually simple and easy to implement •Central site has complete information and can optimally resolve deadlocks	•Considerable communications overhead; every node must send state information to central node •Vulnerable to failure of central node	•Not vulnerable to single point of failure •Deadlock resolution activity is limited if most potential deadlocks are relatively localized	•May be difficult to configure system so that most potential deadlocks are localized; otherwise there may actually be more overhead than in a distributed approach	•Not vulnerable to single point of failure •No node is swamped with deadlock detection activity	•Deadlock resolution is cumbersome because several sites may detect the same deadlock and may not be aware of other nodes involved in the deadlock •Algorithms are difficult to design because of timing considerations

Table 15.1 Security Threats and Assets

	Availability	Secrecy	Integrity/Authenticity
Hardware	Equipment is stolen or disabled, thus denying service.		
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

Table 15.5 Average Time Required for Exhaustive Key Search

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/μs	Time required at 10^6 decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	2^{31} μs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} μs = 5.9×10^{36} years	5.9×10^{30} years

Table 15.2 Observed Password Lengths

Length	Number	Fraction of Total
1	55	.004
2	87	.006
3	212	.02
4	449	.03
5	1,260	.09
6	3,035	.22
7	2,917	.21
8	5,772	.42
Total	13,787	1.0

Table 15.3 Passwords Cracked from a Sample Set of 13,797 Accounts [KLEI90]

Type of Password	Search Size	Number of Matches	Percentage of Passwords Matched
User/account name	130	368	2.7%
Character sequences	866	22	0.2%
Numbers	427	9	0.1%
Chinese	392	56	0.4%
Place names	628	82	0.6%
Common names	2,239	548	4.0%
Female names	4,280	161	1.2%
Male names	2,866	140	1.0%
Uncommon names	4,955	130	0.9%
Myths and legends	1,246	66	0.5%
Shakespearean	473	11	0.1%
Sports terms	238	32	0.2%
Science fiction	691	59	0.4%
Movies and actors	99	12	0.1%
Cartoons	92	9	0.1%
Famous people	290	55	0.4%
Phrases and patterns	933	253	1.8%
Surnames	33	9	0.1%
Biology	58	1	0.0%
System dictionary	19,683	1,027	7.4%
Machine names	9,018	132	1.0%
Mnemonics	14	2	0.0%
King James bible	7,525	83	0.6%
Miscellaneous words	3,212	54	0.4%
Yiddish words	56	0	0.0%
Asteroids	2,407	19	0.1%
TOTAL	62,727	3,340	24.2%

Table 15.4 Virus Propagation Times

Virus	Year launched	Type	Time it took to be most prevalent	Estimated damages
Jerusalem, Cascade, Form	1990	.exe file	3 years	$50 million for all viruses over five years
Concept	1995	Word macro	4 months	$50 million
Melissa	1999	E-mail enabled Work macro	4 days	Up to $385 million
Love letter	2000	E-mail enabled, VBS-based	5 hours	Up to $15 billion

Source: www.icsa.net

Table 2.1   Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 K	100 K	80 K
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 2.2 Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	22%	43%
Memory use	30%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	**Time Sharing**
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 2.4 Operating System Design Hierarchy

Level	Name	Objects	Example Operations
13	Shell	User programming environment	Statements in shell language
12	User processes	User processes	Quit, kill, suspend, resume
11	Directories	Directories	Create, destroy, attach, detach, search, list
10	Devices	External devices, such as printers, displays, and keyboards	Open, close, read, write
9	File system	Files	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, destroy, open, close, read, write
7	Virtual memory	Segments, pages	Read, write, fetch
6	Local secondary store	Blocks of data, device channels	Read, write, allocate, free
5	Primitive processes	Primitive processes, semaphores, ready list	Suspend, resume, wait, signal
4	Interrupts	Interrupt-handling programs	Invoke, mask, unmask, retry
3	Procedures	Procedures, call stack, display	Mark stack, call, return
2	Instruction set	Evaluation stack, microprogram interpreter, scalar and array data	Load, store, add, subtract, branch
1	Electronic circuits	Registers, gates, buses, etc.	Clear, transfer, activate, complement

Shaded area represents hardware.

Table 2.5 Some Areas Covered by the Win32 API [RICH97]

Atoms	Networks
Child controls	Pipes and mailslots
Clipboard manipulations	Printing
Communications	Processes and threads
Consoles	Registry database manipulation
Debugging	Resources
Dynamic link libraries (DLLs)	Security
Event logging	Services
Files	Structured exception handling
Graphics drawing primitives	System information
Keyboard and mouse input	Tape backup
Memory management	Time
Mutimedia services	Window management

Table 2.6   NT Microkernel Control Objects [MS96]

Asynchronous Procedure Call	Used to break into the execution of a specified thread and to cause a procedure to be called in a specified processor mode.
Interrupt	Used to connect an interrupt source to an interrupt service routine by means of an entry in an Interrupt Dispatch Table (IDT). Each processor has an IDT that is used to dispatch interrupts that occur on that processor.
Process	Represents the virtual address space and control information necessary for the execution of a set of thread objects. A process contains a pointer to an address map, a list of ready thread containing thread objects, a list of threads belonging to the process, the total accumulated time for all threads executing within the process, and a base priority.
Profile	Used to measure the distribution of run time within a block of code. Both user and system code can be profiled.

Table 3.6 Pentium EFLAGS Register Bits

Control Bits	Operating Mode Bits
AC (Alignment check) Set if a word or doubleword is addressed on a nonword or nondoubleword boundary.	**NT (Nested task flag)** Indicates that the current task is nested within another task in protected mode operation.
ID (Identification flag) If this bit can be set and cleared, this processor supports the CPUID instruction. This instruction provides information about the vendor, family, and model.	**VM (Virtual 8086 mode)** Allows the programmer to enable or disable virtual 8086 mode, which determines whether the processor runs as an 8086 machine.
RF (Resume flag) Allows the programmer to disable debug exceptions so that the instruction can be restarted after a debug exception without immediately causing another debug exception.	**VIP (Virtual interrupt pending)** Used in virtual 8086 mode to indicate that one or more interrupts are awaiting service.
IOPL (I/O privilege level) When set, causes the processor to generate an exception on all accesses to I/O devices during protected mode operation.	**VIF (Virtual interrupt flag)** Used in virtual 8086 mode instead of IF. **Condition Codes**
DF (Direction flag) Determines whether string processing instructions increment or decrement the 16-bit half-registers SI and DI (for 16-bit operations) or the 32-bit registers ESI and EDI (for 32-bit operations).	**AF (Auxiliary carry flag)** Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation using the AL register.
IF (Interrupt enable flag) When set, the processor will recognize external interrupts.	**CF (Carry flag)** Indicates carrying our or borrowing into the leftmost bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
TF (Trap flag) When set, causes an interrupt after the execution of each instruction. This is used for debugging.	**OF (Overflow flag)** Indicates an arithmetic overflow after an addition or subtraction.
	PF (Parity flag) Parity of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
	SF (Sign flag) Indicates the sign of the result of an arithmetic or logic operation.
	ZF (Zero flag) Indicates that the result of an arithmetic or logic operation is 0.

Table 3.1 Reasons for Process Creation

New batch job	The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Table 3.2 Reasons for Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource or a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.
Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

Table 3.3 Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.

Table 3.4 Typical Elements of a Process Image

User Data	
	The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.
User Program	
	The program to be executed.
System Stack	
	Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.
Process Control Block	
	Data needed by the operating system to control the process (see Table 3.6).

Table 3.5 Typical Elements of a Process Control Block (page 1 of 2)

Process Identification

Identifiers

Numeric identifiers that may be stored with the process control block include

- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Processor State Information

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Control and Status Registers

These are a variety of processor registers that are employed to control the operation of the processor. These include

- *Program counter:* Contains the address of the next instruction to be fetched
- *Condition codes:* Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- *Status information:* Includes interrupt enabled/disabled flags, execution mode

Stack Pointers

Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

Table 3.5 Typical Elements of a Process Control Block (page 2 of 2)

Process Control Information
Scheduling and State Information
This is information that is needed by the operating system to perform its scheduling function. Typical items of information: •*Process state:* defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted). •*Priority:* One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable) •*Scheduling-related information:* This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running. •*Event:* Identity of event the process is awaiting before it can be resumed.
Data Structuring
A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent-child (creator-created) relationship with another process. The process control block may contain pointers to other processes to support these structures.
Interprocess Communication
Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.
Process Privileges
Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.
Memory Management
This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.
Resource Ownership and Utilization
Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

Table 3.7 Typical Functions of an Operating System Kernel

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- Management of process control blocks

Memory Management

- Allocation of address space to processes
- Swapping
- Page and segment management

I/O Management

- Buffer management
- Allocation of I/O channels and devices to processes

Support Functions

- Interrupt handling
- Accounting
- Monitoring

Table 3.8 Mechanisms for Interrupting the Execution of a Process

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

Table 3.9 UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Table 3.10 UNIX Process Image

	User-Level Context
Process Text	Executable machine instructions of the program
Process Data	Data accessible by the program of this process
User Stack	Contains the arguments, local variables, and pointers for functions executing in user mode
Shared Memory	Memory shared with other processes, used for interprocess communication
	Register Context
Program Counter	Address of next instruction to be executed; may be in kernel or user memory space of this process
Processor Status Register	Contains the hardware status at the time of preemption; contents and format are hardware dependent
Stack Pointer	Points to the top of the kernel or user stack, depending on the mode of operation at the time or preemption
General-Purpose Registers	Hardware dependent
	System-Level Context
Process Table Entry	Defines state of a process; this information is always accessible to the operating system
U (user) Area	Process control information that needs to be accessed only in the context of the process
Per Process Region Table	Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute
Kernel Stack	Contains the stack frame of kernel procedures as the process executes in kernel mode

Table 3.11 UNIX Process Table Entry

Process Status	Current state of process.
Pointers	To U area and process memory area (text, data, stack).
Process Size	Enables the operating system to know how much space to allocate the process.
User Identifiers	The **real user ID** identifies the user who is responsible for the running process. The **effective user ID** may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.
Process Identifiers	ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.
Event Descriptor	Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.
Priority	Used for process scheduling.
Signal	Enumerates signals sent to a process but not yet handled.
Timers	Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.
P_link	Pointer to the next link in the ready queue (valid if process is ready to execute).
Memory Status	Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.

Table 3.12 UNIX U Area

Process Table Pointer	Indicates entry that corresponds to the U area.
User Identifiers	Real and effective user IDs. Used to determine user privileges.
Timers	Record time that the process (and its descendants) spent executing in user mode and in kernel mode.
Signal-Handler Array	For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).
Control Terminal	Indicates login terminal for this process, if one exists.
Error Field	Records errors encountered during a system call.
Return Value	Contains the result of system calls.
I/O Parameters	Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.
File Parameters	Current directory and current root describe the file system environment of the process.
User File Descriptor Table	Records the files the process has open.
Limit Fields	Restrict the size of the process and the size of a file it can write.
Permission Modes Fields	Mask mode settings on files the process creates.

Table 3.13 VAX/VMS Process States

Process State	Process Condition
Currently Executing	Running process.
Computable (resident)	Ready and resident in main memory.
Computable (outswapped)	Ready, but swapped out of main memory.
Page Fault Wait	Process has referenced a page not in main memory and must wait for the page to be read in.
Collided Page Wait	Process has referenced a shared page that is the cause of an existing page fault wait in another process, or a private page that is in the process of being read in or written out.
Common Event Wait	Waiting for shared event flag (event flags are single-bit interprocess signaling mechanisms).
Free Page Wait	Waiting for a free page in main memory to be added to the collection of pages in main memory devoted to this process (the working set of the process).
Hibernate Wait (resident)	Process puts itself in a wait state.
Hibernate Wait (outswapped)	Hibernating process is swapped out of main memory.
Local Event Wait (resident)	Process in main memory and waiting for local event flag (usually I/O completion).
Local Event Wait (outswapped)	Process in local event wait is swapped out of main memory.
Suspended Wait (resident)	Process is put into a wait state by another process.
Suspended Wait (outswapped)	Suspended process is swapped out of main memory.
Resource Wait	Process waiting for miscellaneous system resource.

Table 4.1 Thread Operation Latencies (μs) [ANDE92]

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Table 4.2 Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Table 4.3 Windows 2000 Process Object Attributes

Process ID	A unique value that identifies the process to the operating system.
Security Descriptor	Describes who created an object, who can gain access to or use the object, and who is denied access to the object.
Base priority	A baseline execution priority for the process's threads.
Default processor affinity	The default set of processors on which the process's threads can run.
Quota limits	The maximum amount of paged and nonpaged system memory, paging file space, and processor time a user's processes can use.
Execution time	The total amount of time all threads in the process have executed.
I/O counters	Variables that record the number and type of I/O operations that the process's threads have performed.
VM operation counters	Variables that record the number and types of virtual memory operations that the process's threads have performed.
Exception/debugging ports	Interprocess communication channels to which the process manager sends a message when one of the process's threads causes an exception.
Exit status	The reason for a process's termination.

Table 4.4 Windows 2000 Thread Object Attributes

Thread ID	A unique value that identifies a thread when it calls a server.
Thread context	The set of register values and other volatile data that defines the execution state of a thread.
Dynamic priority	The thread's execution priority at any given moment.
Base priority	The lower limit of the thread's dynamic priority.
Thread processor affinity	The set of processors on which the thread can run, which is a subset or all of the processor affinity of the thread's process.
Thread execution time	The cumulative amount of time a thread has executed in user mode and in kernel mode.
Alert status	A flag that indicates whether the thread should execute an asynchronous procedure call.
Suspension count	The number of times the thread's execution has been suspended without being resumed.
Impersonation token	A temporary access token allowing a thread to perform operations on behalf of another process (used by subsystems).
Termination port	An interprocess communication channel to which the process manager sends a message when the thread terminates (used by subsystems).
Thread exit status	The reason for a thread's termination.

Table 5.1 Process Interaction

Degree of Awareness	**Relationship**	**Influence that one Process has on the Other**	**Potential Control Problems**
Processes unaware of each other	Competition	•Results of one process independent of the action of others •Timing of process may be affected	•Mutual exclusion •Deadlock (renewable resource) •Starvation
Processes indirectly aware of each other (e.g., shared object)	Cooperation by sharing	•Results of one process may depend on information obtained from others •Timing of process may be affected	•Mutual exclusion •Deadlock (renewable resource) •Starvation •Data coherence
Processes directly aware of each other (have communication primitives available to them)	Cooperation by communication	•Results of one process may depend on information obtained from others •Timing of process may be affected	•Deadlock (consumable resource) •Starvation

Table 5.2 Possible Scenario for the Program of Figure 5.12

	Producer	Consumer	s	n	Delay
1			1	0	0
2	waitB(s)		0	0	0
3	n++		0	1	0
4	**if** (n==1) (signalB(delay))		0	1	1
5	signalB(s)		1	1	1
6		waitB(delay)	1	1	0
7		waitB(s)	0	1	0
8		n--	0	0	0
9		SignalB(s)	1	0	0
10	waitB(s)		0	0	0
11	n++		0	1	0
12	**if** (n==1) (signalB(delay))		0	1	1
13	signalB(s)		1	1	1
14		**if** (n==0) (waitB(delay))	1	1	1
15		waitB(s)	0	1	1
16		n--	0	0	1
17		signalB(s)	1	0	1
18		**if** (n==0) (waitB(delay))	1	0	0
19		waitB(s)	0	0	0
20		n--	0	–1	0
21		signalB(s)	1	–1	0

Shaded areas represent the critical section controlled by semaphore s.

Table 5.3 Purpose of Semaphores in Figure 5.19

Semaphore	Wait Operation	Signal Operation
max_capacity	Customer waits for space to enter shop.	Exiting customer signals customer waiting to enter.
sofa	Customer waits for seat on sofa.	Customer leaving sofa signals customer waiting for sofa.
barber_chair	Customer waits for empty barber chair.	Barber signals when that barber's chair is empty.
cust_ready	Barber waits until a customer is in the chair.	Customer signals barber that customer is in the chair.
finished	Customer waits until his haircut is complete.	Barber signals when done cutting hair of this customer.
leave_b_chair	Barber waits until customer gets up from the chair.	Customer signals barber when customer gets up from chair.
payment	Cashier waits for a customer to pay.	Customer signals cashier that he has paid.
receipt	Customer waits for a receipt for payment.	Cashier signals that payment has been accepted.
coord	Wait for a barber resource to be free to perform either the hair cutting or cashiering function.	Signal that a barber resource is free.

Table 5.4 Design Characteristics of Message Systems for Interprocessor Communication and Synchronization

Synchronization
- Send
  - blocking
  - nonblocking
- Receive
  - blocking
  - nonblocking
  - test for arrival

Addressing
- Direct
  - send
  - receive
    - explicit
    - implicit
- Indirect
  - static
  - dynamic
  - ownership

Format
- Content
- Length
  - fixed
  - variable

Queuing Discipline
- FIFO
- Priority

Table 5.5 State of the Process Queues for Program of Figure 5.29

Readers only in the system	•*wsem* set •no queues
Writers only in the system	•*wsem* and *rsem* set •writers queue on wsem
Both readers and writers with read first	•*wsem* set by reader •*rsem* set by writer •all writers queue on *wsem* •one reader queues on *rsem* •other readers queue on *z*
Both readers and writers with write first	•*wsem* set by writer •*rsem* set by writer •writers queue on *wsem* •one reader queues on *rsem* •other readers queue on *z*

Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Principle	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources.	Requesting all resources at once.	•Works well for processes that perform a single burst of activity. •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary •Subject to cyclic restart
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Preempts without much use •Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible.	Invoke periodically to test for deadlock.	•Never delays process initiation •Facilitates on-line handling	•Inherent preemption losses

Table 6.3 Windows 2000 Synchronization Objects

Object Type	Definition	Set to Signaled State When	Effect on Waiting Threads
Process	A program invocation, including the address space and resources required to run the program	Last thread terminates	All released
Thread	An executable entity within a process	Thread terminates	All released
File	An instance of an opened file or I/O device	I/O operation completes	All released
Console Input	A text window screen buffer. (e.g., used to handle screen I/O for an MS-DOS application)	Input is available for processing	One thread released
File Change Notification	A notification of any file system changes.	Change occurs in file system that matches filter criteria of this object	One thread released
Mutex	A mechanism that provides mutual exclusion capabilities for the Win32 and OS/2 environments	Owning thread or other thread releases the mutant	One thread released
Semaphore	A counter that regulates the number of threads that can use a resource	Semaphore count drops to zero	All released
Event	An announcement that a system event has occurred	Thread sets the event	All released
Waitable Timer	A counter that records the passage of time	Set time arrives or time interval expires	All released

Note: Shaded rows correspond to objects that exist for the sole purpose of synchronization.

Table 6.2 UNIX Signals

Value	Name	Description
01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work
02	SIGINT	Interrupt
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump
04	SIGILL	Illegal instruction
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing
06	SIGIOT	IOT instruction
07	SIGEMT	EMT instruction
08	SIGFPT	Floating-point exception
09	SIGKILL	Kill; terminate process
10	SIGBUS	Bus error
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space
12	SIGSYS	Bad argument to system call
13	SIGPIPE	Write on a pipe that has no readers attached to it
14	SIGALARM	Alarm clock; issued when a process wishes to receive a signal after a period of time
15	SIGTERM	Software termination
16	SIGUSR1	User-defined signal 1
17	SIGUSR2	User-defined signal 2
18	SIGCLD	Death of a child
19	SIGPWR	Power failure

Table 7.1 Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main_memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation.	Improved memory utilization and reduced overhead compared to dynamic partitioning.
Virtual-Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual-Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Table 7.2 Address Binding

(a) Loader

Binding Time	Function
Programming time	All actual physical addresses are directly specified by the programmer in the program itself.
Compile or assembly time	The program contains symbolic address references, and these are converted to actual physical addresses by the compiler or assembler.
Load time	The compiler or assembler produces relative addresses. The loader translates these to absolute addresses at the time of program loading.
Run time	The loaded program retains relative addresses. These are converted dynamically to absolute addresses by processor hardware.

(b) Linker

Linkage Time	Function
Programming time	No external program or data references are allowed. The programmer must place into the program the source code for all subprograms that are referenced.
Compile or assembly time	The assembler must fetch the source code of every subroutine that is referenced and assemble them as a unit.
Load module creation	All object modules have been assembled using relative addresses. These modules are linked together and all references are restated relative to the origin of the final load module.
Load time	External references are not resolved until the load module is to be loaded into main memory. At that time, referenced dynamic link modules are appended to the load module, and the entire package is loaded into main or virtual memory.
Run time	External references are not resolved until the external call is executed by the processor. At that time, the process is interrupted and the desired module is linked to the calling program.

Table 8.1 Characteristics of Paging and Segmentation

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory frames for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes

Table 8.3 Operating System Policies for Virtual Memory

Fetch Policy	**Resident Set Management**
Demand	Resident set size
Prepaging	Fixed
	Variable
Placement Policy	Replacement Scope
	Global
Replacement Policy	Local
Basic Algorithms	
Optimal	**Cleaning Policy**
Least recently used (LRU)	Demand
First-in-first-out (FIFO)	Precleaning
Clock	
Page buffering	**Load Control**
	Degree of multiprogramming

Table 8.4 Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process. •Page to be replaced is chosen from among the frames allocated to that process.	•Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Table 8.5 UNIX SVR4 Memory Management Parameters (page 1 of 2)

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to zero when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Table 8.5 UNIX SVR4 Memory Management Parameters (page 2 of 2)

Page Frame Data Table Entry

Page State
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.

Table 8.6 Average Search Length for one of *N* items in a Table of Length *M*

Technique	**Search Length**
Direct	1
Sequential	$\frac{M+1}{2}$
Binary	$\log_2 M$
Linear hashing	$\frac{2 - N/M}{2 - 2N/M}$
Hash (overflow with chaining)	$1 + \frac{N-1}{2M}$

Table 9.3 Characteristics of Various Scheduling Policies

	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
FCFS	max[w]	Nonpreemptive	Not emphasized	May be high, especially if there is a large variance in process execution times	Minimum	Penalizes short processes; penalizes I/O bound processes	No
Round Robin	constant	Preemptive (at time quantum)	May be low if quantum is too small	Provides good response time for short processes	Minimum	Fair treatment	No
SPN	min[s]	Nonpreemptive	High	Provides good response time for short processes	Can be high	Penalizes long processes	Possible
SRT	min[$s - e$]	Preemptive (at arrival)	High	Provides good response time	Can be high	Penalizes long processes	Possible
HRRN	$\max\left(\frac{w+s}{s}\right)$	Nonpreemptive	High	Provides good response time	Can be high	Good balance	No
Feedback	(see text)	Preemptive (at time quantum)	Not emphasized	Not emphasized	Can be high	May favor I/O bound processes	Possible

w = time spent in system so far, waiting and executing
e = time spent in execution so far
s = total service time required by the process, including e

Table 9.5 A Comparison of Scheduling Policies

							Mean
	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	
FCFS	Finish Time	3	9	13	18	20	
	Turnaround Time (T_r)	3	7	9	12	12	8.60
	T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$	Finish Time	4	18	17	20	15	
	Turnaround Time (T_r)	4	16	13	14	7	10.80
	T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$	Finish Time	3	17	11	20	19	
	Turnaround Time (T_r)	3	15	7	14	11	10.00
	T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN	Finish Time	3	9	15	20	11	
	Turnaround Time (T_r)	3	7	11	14	3	7.60
	T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT	Finish Time	3	15	8	20	10	
	Turnaround Time (T_r)	3	13	4	14	2	7.20
	T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	Finish Time	3	9	13	20	15	
	Turnaround Time (T_r)	3	7	9	14	7	8.00
	T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$	Finish Time	4	20	16	19	11	
	Turnaround Time (T_r)	4	18	12	13	3	10.00
	T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$	Finish Time	4	17	18	20	14	
	Turnaround Time (T_r)	4	15	14	14	6	10.60
	T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

Table 9.1 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Table 9.2 Scheduling Criteria

	User Oriented, Performance Related
Turnaround time	This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.
Response time	For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.
Deadlines	When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.
	User Oriented, Other
Predictability	A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.
	System Oriented, Performance Related
Throughput	The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.
Processor utilization	This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.
	System Oriented, Other
Fairness	In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.
Enforcing priorities	When processes are assigned priorities, the scheduling policy should favor higher-priority processes.
Balancing resources	The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Table 9.6 Formulas for Single-Server Queues with Two Priority Categories

Assumptions:
1. Poisson arrival rate.
2. Priority 1 items are serviced before priority 2 items.
3. First-in-first-out dispatching for items of equal priority.
4. No item is interrupted while being served.
5. No items leave the queue (lost calls delayed).

(a) General Formulas

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 T_{s1}; \;\; \rho_2 = \lambda_2 T_{s2}$$

$$\rho = \rho_1 + \rho_2$$

$$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$

$$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

(b) No interrupts; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 - \rho_1}$$

$$T_{r2} = T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}$$

(c) Preemptive-resume queuing discipline; exponential service times

$$T_{r1} = 1 + \frac{\rho_1 T_{s1}}{1 - \rho_1}$$

$$T_{r2} = 1 + \frac{1}{1 - \rho}\left(\rho_1 T_{s1} + \frac{\rho T_s}{1 - \rho}\right)$$

Table 9.7 Response Time Ranges

Greater than 15 seconds

This rules out conversational interaction. For certain types of applications, certain types of users may be content to sit at a terminal for more than 15 seconds waiting for the answer to a single simple inquiry. However, for a busy person, captivity for more than 15 seconds seems intolerable. If such delays will occur, the system should be designed so that the user can turn to other activities and request the response at some later time.

Greater than 4 seconds

These are generally too long for a conversation requiring the operator to retain information in short-term memory (the operator's memory, not the computer's!). Such delays would be very inhibiting in problem-solving activity and frustrating in data entry activity. However, after a major closure, delays from 4 to 15 seconds can be tolerated.

2 to 4 seconds

A delay longer than 2 seconds can be inhibiting to terminal operations demanding a high level of concentration. A wait of 2 to 4 seconds at a terminal can seem surprisingly long when the user is absorbed and emotionally committed to complete what he or she is doing. Again, a delay in this range may be acceptable after a minor closure has occurred.

Less than 2 seconds

When the terminal user has to remember information throughout several responses, the response time must be short. The more detailed the information remembered, the greater the need for responses of less than 2 seconds. For elaborate terminal activities, 2 seconds represents an important response-time limit.

Subsecond response time

Certain types of thought-intensive work, especially with graphics applications, require very short response times to maintain the user's interest and attention for long periods of time.

Decisecond response time

A response to pressing a key and seeing the character displayed on the screen or clicking a screen object with a mouse needs to be almost instantaneous—less than 0.1 second after the action. Interaction with a mouse requires extremely fast interaction if the designer is to avoid the use of alien syntax (one with commands, mnemonics, punctuation, etc.) .

Table 9.8 Notation for Queuing Systems

λ	=	arrival rate; mean number of arrivals per second
T_s	=	mean service time for each arrival; amount of time being served, not counting time waiting in the queue
ρ	=	utilization; fraction of time facility (server or servers) is busy
w	=	mean number of items waiting to be served
T_w	=	mean waiting time for (including items that have to wait and items with waiting time = 0)
r	=	mean number of items resident in system (waiting and being served)
T_r	=	mean residence time; time an item spends in system (waiting and being served)

Table B.1 Key Object-Oriented Terms

Term	Definition
Attribute	Data variables contained within an object.
Containment	A relationship between two object instances in which the containing object includes a pointer to the contained object.
Encapsulation	The isolation of the attributes and services of an object instance from the external environment. Services may only be invoked by name and attributes may only be accessed by means of the services.
Inheritance	A relationship between two object classes in which the attributes and services of a parent class are acquired by a child class.
Message	The means by which objects interact.
Method	A procedure that is part of an object and that can be activated from outside the object to perform certain functions.
Object	An abstraction of a real-world entity.
Object Class	A named set of objects that share the same names, sets of attributes, and services.
Object Instance	A specific member of an object class, with values assigned to the attributes.
Polymorphism	Refers to the existence of multiple objects that use the same names for services and present the same interface to the external world but that represent different types of entities.
Service	A function that performs an operation on an object

Table B.2 Key Concepts in a Distributed CORBA System

CORBA Concept	Definition
Client application	Invokes requests for a server to perform operations on objects. A client application uses one or more interface definitions that describe the objects and operations the client can request. A client application uses object references, not objects, to make requests.
Exception	Contains information that indicates whether a request was successfully performed.
Implementation	Defines and contains one or more methods that do the work associated with an object operation. A server can have one or more implementations.
Interface	Describes how instances of an object will behave, such as what operations are valid on those objects.
Interface definition	Describes the operations that are available on a certain type of object.
Invocation	The process of sending a request.
Method	The server code that does the work associated with an operation. Methods are contained within implementations.
Object	Represents a person, place, thing, or piece of software. An object can have operations performed on it, such as the `promote` operation on an employee object.
Object instance	An occurrence of one particular kind of object.
Object reference	An identifier of an object instance.
OMG Interface Definition Language (IDL)	A definition language for defining interfaces in CORBA.
Operation	The action that a client can request a server to perform on an object instance.
Request	A message sent between a client and a server application.
Server application	Contains one or more implementations of objects and their operations.