UNIX
-First OS that was portable was written in a "high-level" language (C)
allowed its users to access—and modify—its source code
.Ideal platform for OS research
Offers unsurpassed tools
Has influenced all recent systems
The early beginnings (I)
MULTICS
Large OS project started in the 60's
Involved MIT, AT&T and General Electric
Wanted to provide a "computing utility" for vast communities of users
Led to many advances in OS technology
Too ambitious for the time
The early beginnings (II)
AT&T quickly withdrew from MULTICS
K. Thompson, having nothing better to do, writes a simple OS for an unused machine
UNIX went through several revisions
Rewritten in C by K. Thompson and D. Ritchie
Ported to a 16-bit machine (PDP-11)
Presented at ACM SOSP in 1973
Coming out of age
AT&T could not at that time sell any computing services-
Made UNIX available at cost to universities
Universities liked
Free time-sharing system
Running on a "cheap" minicomputer
Easy to modify
The triumphant decade (I)
DEC VAX
more powerful version of PDP-11
32 bit architecture
virtual memory
Ö. Babaoğlu and W. Joy (U .C. Berkeley) added virtual memory support to UNIX
DARPA decided to have all ARPANET development done on UNIX
The triumphant decade (II)
UNIX became de facto standard OS for
Minicomputers
Workstations (W. Joy went to Sun)
Internet servers
Two different traditions appeared
Berkeley UNIX (BSD 4.2, 4.3 and 4.4)
AT&T System III and System V
The recent years (I)
MS-DOS did not hurt UNIX:
Smaller and simpler subset of UNIX commands
Windows 3.1 and later did:
More intuitive graphical user interface
UNIX had its own problem:
Very fragmented market prevented development of cheap commercial software
The recent years (II)
Free versions of UNIX:
Based on Berkeley UNIX: FreeBSD, BSD Lite
Written from scratch: GNU, Linux
Latest version of MacOS (MacOS X) is Unix-based and has an open-source kernel
UNIX still dominates the server market
Several ports of UNIX API to Windows
Key features of UNIX
"Small is beautiful" philosophy
Most tasks can be accomplished by combining existing small programs
echo `who | wc -l` users
Written in a high-level language
Modular design
Great file system
THE UNIX FILE SYSTEM
Three types of files
ordinary files:
uninterpreted sequences of bytes
directories:
accessed through special system calls
special files:
allow access to hardware devices
Ordinary files (I)
Five basic file operations are implemented:

open()	returns a file descriptor
read()	reads so many bytes
write()	writes so many bytes
lseek()	changes position of current byte
close()	destroys the file descriptor

Ordinary files (II)
All reading and writing are sequential.
The effect of direct access is achieved by manipulating the offset through lseek()
Files are stored into fixed-size blocks
Block boundaries are hidden from the users
Same as in MS-DOS/Windows
The file metadata
Include file size, file owner, access rights, last time the file was modified,
but not the file name
Stored in the file i-node
Accessed through special system calls:
chmod(), chown(), ...
I/O buffering
UNIX caches in main memory
I-nodes of opened files
Recently accessed file blocks
Delayed write policy
Increases the I/O throughput
Will result in lost writes whenever a process or the system crashes.
Terminal I/O are buffered one line at a time.
Directories (I)
Map file names with i-node addresses
Do not contain any other information!
Directories (II)
Two directory entries can point to the same i-node
Directory subtrees cannot cross file system boundaries unless a new file system is mounted somewhere in the subtree.
To avoid loops in directory structure, directory files cannot have more than one pathname-
"Mounting" a file system (after mount,root of second partition can be accesed as/usr
Directories (III)
With Berkeley UNIX, symbolic links you can write
ln -s /usr/bin/programs /bin/programs
even tough /usr/bin/programs and /bin/programs are in two different partitions
Symbolic links point to another directory entry instead of the i-node.
Special files
Map file names with system devices:

/dev/tty	your terminal screen
/dev/kmem	the kernel memory
/dev/fd0	the floppy drive

Main motivation is to allow accessing these devices as if they were files:
no separate I/O constructs for devices
PROTECTION
File owner can specify three access rights
read,write ,execute
for
herself (user),a group in /etc/group (group),all other users (other)
eg:rwx,rw-r,rw-rw-r
Limitations
No append right
Only the system administrator can change group membership
-Works well for stable groups
-Owner of a file cannot add and remove users at her whim
The set user-ID bit (I)
Suppose I want to let you access your grades but not those of your classmates:
First solution is having one file per student and create as many groups in /etc/group as there are students
Too complicated!
Better having restrictions enforced by an user program
users->my program->file(myprog enforces all access restrictions)
The set user-ID bit (II)
Solution does not work because myprogram will be not able to access the data when students run it
Set user-ID bit can specify that any executable should be executed using the rights of the owner of the file
and not the rights of the user executing it
user->myprod with SUID->file (myprog now runs as if i was running it)
Security risk of SUID
Assume that myprogram can be modified by other users
One of them could replace it by her version of the shell
Whenever she executes her new version of myprogram, she has access to all my files
Be very careful with SUID programs!
File locking
Allows to control shared access to a file We want a one writer/multiple readers policy
Older versions of UNIX did not allow file locking
System V allows file and record locking at a byte-level granularity through fcntl()
Berkeley UNIX has advisory file locks:
like asking people to knock before entering
Version 7 Implementation
Each disk partition contains:
a **superblock** containing the parameters of the file system disk partition
an **i-list** with one i-node for each file or directory in the disk partition and a free list.
the **data blocks** (512 bytes)
A disk partition ("filesystem")
The i-node (I)
Each i-node contains:
The user-id and the group-id of the file owner
The file protection bits,
The file size,
The times of file creation, last usage and last modification,
The i-node (II)
The number of directory entries pointing to the file, and
A flag indicating if the file is a directory, an ordinary file, or a special file.
Thirteen block addresse
The file name(s) can be found in the directory entries pointing to the i-node.
Storing block addresses

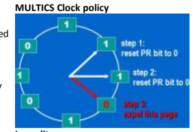
How it works(I):=first ten blocks of file can be directlyfrom i node 10X512=5120bytes
-indirect block contains 512/4=128 addresses
128X512=64kbytes
-with two levels of indication we can access 128X128=16k blocks 16kX512=8 megabytes
-with three levels of indication we can access 128X128X128=2Mbloks 2MX512=1gigabyte
-Max file size is 1gb+8gb+64kb+5kb
Explanation
-File sizes can vary from a few hundred bytes to a few gigabytes with a hard limit of 4 gigabytes
-The designers of UNIX selected an i-node organization that
Wasted little space for small files
Allowed very large files
Discussion
What is the true cost of accessing large files?
-UNIX caches i-nodes and data blocks
-When we access sequentially a very large file we fetch only once each block of pointers
Very small overhead
-Random access will result in more overhead
Berkeley Modifications (I)
BSD introduced the "fast file system" (FFS)
-Superblock is replicated on different cylindersof disk
-disk is divided into cylinder group
-each cylinder group has its own i node table
It minimizes disk arm motions
-free list replaced by bit maps.
-I-node has now 15 block addresses
-Minimum block size is 4K
-15th block address is never used
The new organisation
Berkeley Modifications (I)
Biggest changes are:
Disk partition is subdivided into groups of consecutive cylinders
Bit MAps
Each cylinder group contains a bit map of all available blocks in the cylinder group
The file system will attempt to keep consecutive blocks of the same file on the same cylinder group
block sizes
FFS uses larger blocks allows the division of a single file system block into 2, 4, or 8 fragments that can be used to store
Small files
The tails of larger files
Explanations (I)
Increasing the block size to 4K eliminates the third level of indirection
Keeping consecutive blocks of the same file on the same cylinder group reduces disk arm motions
Explanations (II)
Allocating full blocks and block fragments
-allows efficient sequential access to large files
-minimizes disk fragmentation
Using 4K blocks without allowing 1K fragment would have wasted 45.6% of the disk space
-This would not be necessarily true today
Limitations of Approach (I)
Traditional UNIX file systems do not utilize full disk bandwidth
-Log-structured file systems do most writes in sequential fashion
Crashes may leave the file system in an inconsistent state
-Must check the consistency of the file system at boot time
Limitations of Approach (II)
Most of the good performance of FFS is due to its extensive use of I/O buffering
-Physical writes are totally asynchronous
Metadata updates must follow a strict order
-Cannot create new directory entry before new i-node it points to
-Cannot delete old i-node before deleting last directory entry pointing to it
Example: Creating a file (I)
Example: Creating a file (II)
Limitations of Approach (III)
Out-of-order metadata updates can leave the file system in temporary inconsistent state
-Not a problem as long as the system does not crash between the two updates
-Systems are known to crash
FFS performs synchronous updates of directories and i-nodes
-Requires many more seeks
PROCESS CREATION
Two basic system calls
fork() creates a carbon-copy of calling process sharing its opened files
exec() overwrites the contents of the process address space with the contents of an executable file
fork(): parent fork returns PID of child,whereas child fork returns 0
Typical usage

```
int pid
 if ((pid = fork()) == 0) {
    // child process
    ...
    exec(...);
    _exit(1); //if exec() failed
 }
 while (pid != wait(0));
 // parent waits
```

Observations
Mechanism is quite costly
fork() makes a complete copy of parent address space
very costly in a virtual memory system
exec() thrashes that address space
Berkeley UNIX introduced cheaper vfork() that shares the parent address space until the child does an exec()
INTERPROCESS COMMUNICATIO
Several mechanisms:Pipes,System V shared memory and message queues,BSD sockets
UNIX pipes
Major usage is combining several programs to perform multiple steps of a single task:a\b\c
Standard output of each process in pipe is forwarded to standard input of next process in pipe:stdout of a goes to stdin of b
stdout of b goes to stdin of c
Usage
UNIX toolkit includes many "filters" or programs that perform one specific step on their standard input and return the result on their standard output:
pic mypaper | tbl | eqn | troff -ms
Pipes are not a general IPC mechanism: must be inherited from a common parent process
System V IPC
System V offers:Shared memory,Semaphores
and Message queues.System V semaphores have a very bad user interfaceMessage queues require sending and receiving processes to be on the same machine
BSD Sockets (I)
Message passing
Most general IPC mechanism
Basic building block of Internet and WWW
Sockets have addresses either in
the UNIX domain (same machine)
the Internet domain
BSD Sockets (II)
Three important types of sockets
stream sockets: provide reliable delivery of data -No data will be lost, replicated or arrive out of sequence
datagram sockets: make no such warranties
raw sockets: very low level
BSD Sockets (III)
Programming IPC with sockets is a cumbersome task as system calls are quite complex-Better use a remote procedure call package or write your own
Applications involving small transfer of data over a local-area network usually use datagram sockets - Avoid the overhead of stream sockets
HTTP uses stream sockets
SCHEDULING (I)
Two cases:
All UNIX systems but System V.3:
Process priorities are function of their base priority and their past CPU usage
Processes that have recently used CPU times are penalized
SCHEDULING (II)
System V.3 :
real-time processes have fixed priorities.
time-sharing processes have variable priorities managed by multi-level feedback queues.
system administrator can modify all the parameters of these queues.
MEMORY MANAGEMENT
Somewhat neglected in the earlier versions of Unix
64 KB address space of PDP-11
No memory mapping hardware
Serious work started when UNIX was ported to the VAX
At Bell Labs
At U. C. Berkeley
The VAX
Virtual Address eXtension of PDP family of minicomputers
32 bit addresses
Complicated instruction set (CISC)
Native operating system (VMS) provided virtual memory
No hardware/firmware support for
page-referenced bit
VMS Virtual Memory (I)
Very small page size: 512 bytes
Minimized internal fragmentation
User could define clusters of contiguous pages that were brought together
Emphasis was on efficient use of main memory
memory was very expensive
512 KB was a huge investment
VMS Virtual Memory (II)
Divided process address space into 4 sections according to two most significant bits of virtual address
00: code segment
01: stack segment
10: shared system area
11: reserved
VMS Virtual Memory (III)
VMS Virtual Memory (IV)
Processes running the same program shared the same code segment
One global page table for the shared system area:-stored in physical memory
Separate page tables for code segment and stack segments:-stored in shared system area
(not in physical memory)
VMS Virtual Memory (V)

VMS Page Replacement policy
Hybrid local/global page replacement policy
-One fixed partition per process
1.Great for real-time support
2.Managed by FIFO policy
-One global queue containing expelled pages
At page fault time, VMM could retrieve faulting page from global queue
-Performance close to that of Global LRU
How VMS policy works

Policy tradeoffs
Reclaims require kernel intervention-cost is 2 context switches.
Policy works on-fixed parttitions are large enough to avaoid excessive numbers of reclaims-global queue large enough to keep expelled pages long enough to have a chance to be reclaimed.
BSD Virtual Memory
Main objective was efficient management of main memory
Wanted to select a page replacement policy that
-Kept in memory recently used pages
-Could be efficiently implemented on a -machine lacking a page-referenced bit
Policies being considered
Three page replacement policies
VMS page replacement policy
Sampled Working Set
MULTICS Clock policy
Last two policies required a page-referenced bit
can be simulated by software
Why not VMS policy? (I)/not in present slides
VMS page replacement policy approximates LRU quite well when
Fixed partitions are large enough to avoid excessive number of reclaims from global queue
Global queue large enough to keep expelled pages long enough to have a chance to be reclaimed
Why not VMS policy? (II)
Requires a good estimate of physical requirements of each new process to allocate a resident set that is
neither too small
nor too big
Such determination is not possible under UNIX
How to simulate PR-bit?
PR-bit
set to one when page is referenced
reset to 0 by VMM software
We can simulate it using the valid bit
reset to 0 by VMM software
first access to page causes an interrupt:
VMM software sets the bit to 1
Overhead is two context switches per reset
Why not Sampled Working Sets?
SWS samples page-referenced bits every T time units of virtual time:
if PR bit = 0, it expels the page
if PR bit = 1, it resets the bit
Provides a very good approximation of Denning's Working Set
Requires too many resets
How SWS works

MULTICS Clock policy (I)
Organizes page frames in a circular list
When a page fault occurs, policy looks at next frame in list
if PR bit = 0, the page is expelled and the page frame receives the incoming page
if PR bit = 1, the PR bit is reset and policy looks at next page in list
MULTICS Clock policy


In reality
same diag but step 1 and step 2:markpage invalid.
Algorithm

```
Frame *Clock(Frame *Last_Victim) {
   Frame *Hand;
   int Not_Found = 1;
 - Hand = Last Victim->Next;
   do {
       if (Hand->PR_Bit == 1) {
            Hand->PR_Bit = 0;
            Hand = Hand->Next;
       } else
            Not_Found = 0;
   } while Not_Found;
   return Hand;
} // Clock
```

A first problem
When memory is overused, hand of clock moves too fast to find pages to be expelled
-Too many resets
=Too many context switches
Berkeley UNIX limited CPU overhead of policy to 10% of CPU time
=No more than 300 page scans/second
Other modifications
Berkeley UNIX maintains a pool of free pages instead of expelling pages on demand
Page size was increased to 1KB (a pair of 512-byte physical pages)
Berkeley UNIX added
-Some limited amount of prepaging
-A mechanism allowing to turn off the PR bit simulation
Evolution of the policy
Policy now runs with much more physical memory
Hand now moves too slowly
By the late 80's a two-hand policy was introduced:
=First hand resets simulated PR bit
=Second hand follows first at constant angle and expels all pages whose PR bit = 0
The two-hand policy
THE LEGACY OF UNIX
UNIX proved that an OS
Could be written in a high-level language
Did not need to be architecture-specific
Other major contributions include
Hierarchic file system
Device independence
Standardized interfaces

->**What should be the inheritance attribute of code segment of a process ? shared** cost of reclaiming a page from the global queue containing all pages waiting to be written back two context switches best way to implement unix fork system call why? Using copy on write,,after a fork(),the data segmnt of the parent process is shared with the child process in such a way that no actual copying will take place unless one of the two processes writes into the shared segment before the child does an exec().Copy-on-write works very well because most child processes do an exex within a few instructions after the fork().hence copying is reduced to a minimum.main benefit of exo-kernel approach? Since the exokernel delegates the management of the system resoureces to the user processes it lets them manage these resources more efficiently.main performance advantage of mapped files?File blocks that are resident n main memory can be accessed without kernel intervention,thus saving context switches.main disadvantage of selecting large transfer units in a DSM?large transfer units increase the risk of false sharing.what do Navarro et al. propose to do whenever a process attempts to write into a superpage?why?whenever a process attempts to write into a superpage,the superpage is demoted and replaced by its constituent base pages.This is done to avoid having to flush back the whole superpage when it will be expelled from main-memory. An increase of target_T1?Any attempt to access pages b1How does Mach ensure that all processes accessing a mapped file share the same view of the file? The range of virtual memory addresses allocated to the mapped files will be shared by all processes accessing the file. Consider a 64-bit system using a clustered page table. subblocking with a subblocking factor of two partial =24=3*8 and complete (sublock factor+2)*8=32
The Midway distributed shared memory system uses a request/release scheme similar to that of Munin but requires users to specify which shared variables are associated with each request() call. In a few words, explain how this additional requirement affects (a) the run-time performance of user programs and (b) their portability.(a)With Munin, a workstation must performing a request() must wait until it has received the most recent values of all shared variables. With, Midway, the workstation will only request the most recent values of the shared variables associated with the request() call. As a result, Midway will be faster than Munin.(b) Porting a program to Midway is a much harder task because the programmer porting the program will have to find out first which shared variables are accessed within each request()/release() pair. This is a daunting task for any large complex program.What is the main advantage of shared libraries? How does Mach implement them? Shared libraries save space in main memory by letting all the processes share a single copy of all the functions in the same shared library. (In addition, they also save space on disk and speed up the time it takes to fetch binaries in main memory.) advantage of mapped files? They reduce the number of context switches by letting processes access directly all file blocks that in main memory What is the difference between a partial subblock and a superpage?All blocks of a superpage are brought in and expelled from main memory together while the blocks of partial subblock are brought in and expelled from main memory individually. inheritance attributes Code segment shared Data segment copy Mapped file shareds shared memory segment?.shared Assume now that we want to create another thread within the address space of that process. What do we need to change before forking that thread? Set the inheritance attribute of the data segment to shareWhat is the main difference between complete subblocking and partial subblockingPartial subblocking requires all the pages forming a subblock to occupy contiguous addresses in the process virtual memory as well as in physical memory. Complete subblocking requires all the pages forming a subblock to occupy contiguous addresses in the process virtual memory but lets tem occupy any location in physical memory. What is the best way to implement the UNIX fork() system call? Why The best way to implement the UNIX fork() system call is using copy-on-write. This technique lets the parent and the child process share the same address space until the child performs an exec(). UNIX fork() semantics are preserved by making a two copies of each page that one of the two processes attempts to modify and letting each process access its own copy of the shared page. Copy-on-write works well because most child processes modify very few pages of their address space before they perform an exec().What is the main performance advantage of mapped files?Fewer context switches as the contents of file blocks already in main memory can be accessed without kernel intervention.What is the main performance advantage of mapped files? An increased amount of false sharing. An increase of target_T1? Cache misses caused by accesses to blocks 1 and 9.What distinguishes Corey processes from both regular kernel supported threads and conventional Unix processes?A Corey process can specify which parts of its address space are shared among sibling processes and which parts are not. An increase false sharing by offering a write-shared consistency protocol. Whenever a process is granted access to write-shared data, the page containing these data is marked copy-on-write. Hence the first attempt to modify the contents of the page will result in the creation of target_T1: When the page that was missing was mentioned in the B1 list not in main memory. When you compare partialsubblockingand complete subblockingPartial subblocking Uses the TLB in the most efficient fashion and why? Partial subblockin because it uses a more compact organization for each TLB entries with only one VPN and one PPN for each entry. Uses the main memory in the most efficient fashion and why Complete subblocking because it does not allocate page frames to the subblock pages that are not currently accessed. Target_T1 will increase when a page fault caused by a missing page that is present in B1 occurs. It will _increase target_T1 by one.When does false sharing happen in a distributed shared memory system When two or more unrelated variables located in the same page are accessed at the same time by two or more different processesWhat problem may it causePing-pong effect (This issue is addressed by shared write protocol.) an external pager in the Mach operating system: the file system a memory object managed by that external page a file Munin is said to use an eager release policy. WhyWhat would be the corresponding lazy release policy? An eager release policy distributes all updates as soon as they are released. A lazy policy does not forward the same updates until some other process requests them. When should we use the Munin write-shared protocol? We should use the Munin write shared protocol whenever two or more processes access distinct variables that reside in the same page and none of these variables are accessed by more than one process. This situation is known as false sharing and results in many unnecessary data transfers among the processes, creating the so-called "ping-pong effect." The Munin write shared protocol avoids these transfers. In the ARC cache replacement policy, which events result in an actual change of the size of T1? T1 will increase when a page fault occurs and the size of T1 is lesser than target_T1T1 will decrease when a page in T1 is referenced a second time the size of T1 is greater than or equal to target_T1. Which actions take place when a Treadmarks process performs:ATmk_lock_acquire(…)The process waits until has acquired the requested lock and obtained the most recent released values of all shared variables.b. Tmk_lock_release(…)It releases the specified lock and notifies Treadmarks that it has released new values of its shared dataWhat are Spin external references? (5 points) How does Spin represent them? Why? SPIN externalized references are used to pass pointers to kernel data structures to userlevel applications in a safe manner. The user-level application is given an index into a perapplication table of safe references to kernel data structures. This prevents any tampering with the pointer by the user-level application.Sketch a page table organization that would interact in an effective fashion with a TLB using partial FIG1 subblockingEach page table entry will contain; a) The virtual page number of the first page of the cluster (VPN0) b) The physical page number of the page frame containing the first page of the cluster (PPN0) c) A bitmap indicating which of the two pages are currently residing in main memory (b) d) The address of the next subblock in the hash bucket (NEXT) In addition, the hash function will ignore the least significant bit of the virtual page number to ensure that pages that belong to the same subblock will hash into the same bucket.Explain why the ARC cache replacement policy is said to be: Scan-resistant because it expels quickly pages that are referenced only once as it is the case when one process scans the whole file system Self-tuning:The ARC cache replacement policy is said to be self-tuning because it has no parameter than can be adjusted by the user.Superpage: A superpage is a set of pages that are brought in and out of memory as a single entity. They are aligned both in virtual and in physical memory and their size is a power-of-two multiple of the page size. Valid bit: The valid bit is a bit indicating whether a given page is in main memory. When does false sharing happen in a distributed shared memory system? When two distinct processors simultaneously access two or more distinct variables that are located in the same pageWhat problem does it cause?The page goes back and forth between the two processors. Which feature of Munin addresses that issue? The Munin write-shared consistency protocol Mach copy-on write policy is said to be a lazy policy. Why? What would be the corresponding eager policy? Has this eager policy been implemented? Under which circumstances is it better than copy-on-write?Mach copy-on-write policy is said to be a lazy policy because it delays the creation of separate parent and child copies of a page until the page is modified by either the parent or the child. The corresponding eager policy would be to create separate address spaces for the parent and child process at fork(0 time without waiting for pages to be modified. This eager policy was used by the traditional implementation of the UNIX fork(). It works better than copy-on-write whenever parent and child modify a large number of their pages. What is the main advantage of mapped files? Which memory object is associated with each mapped file?What should be the inheritance and protection attributes of a mapped fileMapped files reduce the number of system calls required to access a file as read(), write() and lseek() operations can be implemented as user-level procedures. The sole operations that require a kernel intervention are the actual transfers of data blocks between the main memory and the disk. The memory object associated with each mapped file is the file itself as it is stored on disk by the file system. The inheritance attribute of a mapped file will always be SHARED. It protection attributes are those specified by the process when it successfully opened the file. In UNIX these are READ-ONLY, WRITE-ONLY and READWRITE. Describe in some detail the page replacement policy of VMSand explain its main respective advantage and disadvantagFIG2 VMS page replacement policy combines some aspects of FIFO and least recently used. VMS assigns to each process a fixed size partition called the process' working set anduses a FIFO replacement policy to manage it. Pages that are expelled from a working set are put at the end of a large global queue of pages waiting to be expelled from main memory. These pages are also marked invalid in the page tables of their respective processes. Every time a page fault occurs, VMS scans first this global queue before attempting to bring the missing page from secondarystorage. Advantages: The mechanism approximates LRU quite well provided that (a) enough pages are allocated to the process working sets to keep the number of false page faults under control, and (b) the global queue is large enough to allow the rescue of pages that had been erroneously expelled from the working set. of their process. In addition, it supports real-time processes. Disadvantages: It does not work well if processes have memory requirements that change over time Assuming a subblocking factor of 4 and 64 bit addresses, the TLB implements full subblocking? (one VPN, four PPN and one next pointer, all 8-byte long the TLB implements partial subblocking?(one VPN, one PPN and one next pointer, all 8-byte long; the valid bit vector is stored in the unused byte of the PPN. Explain how Mach copy-on-write feature speeds up fork() system calls. Most fork() system calls are quickly followed by an exec() with only I/O redirection tasks being performed between the two. As a result, extremely few pages of the child process address space are modified. With copy-on-write, only these very few pages will be duplicated.What is the main difference between superpages and subblocks The main difference between superpages and subblocks is that all pages in a subblock do not need to be simultaneously present in main memory. Which feature of Mach allows it to implement more efficiently the UNIX fork()?How is their solution better than the Berkeley UNIX vfork()?The feature is called _ copy-on-write It is better than the UNIX vfork() because it is more general(vfork() only works when the fork() is quickly followed by an exec()What is false sharing in a distributed shared memory system? What problem does it cause?Which feature of Munin addresses that issue? False sharing happens every time two distinct processors access two distinct variables that happen to be located in the same DSM transfer unit. These transfer units happen to be pages in Munin. False sharing results in continuous exchanges of the whole transfer unit between the two processors. Munin fights of ac copy of the page modified (the twin). At release time, the DSM will perform a word by word comparison of the page and its twin, store the diff in the space used by the twin pages and notify all processes having a copy of the shared data of the update What is the main advantage of mapped files? Why would they be very hard to implement on a conventional virtual memory system?Which features of the Mach virtual memory system make their implementation feasible and why? Mapped files reduce the number of context switches required to implement I/O operations because it allows user processes to access mapped files through library calls instead of through system calls. Mapped files are very hard to implement on a conventional virtual memory system because these systems treat all pages in the data segments of their processes in the same fashion. The two features of Mach that allow the implementation of mapped files are: 1. External pagers: since they allow mapped file pages to be paged from the file system rather than from the swap area of the virtual memory system. 2. Inheritance: since it allows mapped file pages to be inherited in shared mode.8 kb page size ,factor 8 Since each 64-kilobyte superpage contains exactly eight pages, each superpage will be mapped into a single page table entry. If 4kb page size 4 factor the 4 table entries There are two good solutions for handling 512-kilobyte superpages: one could either have eight separate page table entries per superpage or use an "ad hoc" mechanism Diff betwn partia and complete Partial subblocking requires all pages of a subblock to occupycontiguous locations (page frames) in main memory while complete subblocking does not have that restriction Munin migratory variable:A shared variable that is never replicated in different processes: every process accessing it will always get full read and write access even if it only requested read.Migratory variables are variables that migrate among the processes accessing them: every process accessing a migratory variable will always get full read and write access even if it only requested read access Inheritance attribute: A parameter in the Mach address map that specifies whether pages in a given range of addresses are to be copied, shared or ignored at fork() time.ThreadA process executing in the same address space as its parent.What is the main advantage of mapped file systems over conventional systemsFile blocks that are brought into main memory are mapped in the address spaces of the processes that access them. Hence the contents of these blocks can be directly accessed by each process without any additional context switches. What would happen if someone writing a Munin program forgets to insert(a)A request() call before accessing a shared data variable? The process accessing the shared variable will not be guaranteed to have the most recent value of the shared variable. (b) A release() call after accessing that shared data variable? The new value of the shared variable will not be propagated to all the other processes.In conventional shared memory systems, critical sections that consist of a single machine language instruction do not have to be surrounded by lock/unlock— or request/releaspairs.Why?Is this still true with Munin? Justify your answer by considering the case of a critical section reading the value of a single shared variable and storing it into a local variable. In a conventional shared memory system, there is no need to surround single instructions by lock/unlock pairs because single instructions are normally uninterruptible. In Munin, the pair request/release plays a double role: (a) It provides mutual exclusion (b) It ensures that the contents of the virtual shared memory are made consistent at the time the processor issues a request and that the new values are propagated when the process issues the matching release. Omitting the request/release pair around a critical section consisting of a single instruction is thus dangerous as we would not have any guarantee that the instruction will access the most current version of the data in the distributed shared memoryWhat characterizes a self-tuning cache replacement policy?Which feature(s) of ARC make that policy self-tuning? A self-tuning cache replacement policy has no parameters that can be set by the system administrator and can affect the performance of the policy. ARC is self-tuning because it has no parameter than can be set or modified by the user. What is the purpose of Munin write shared consistency protocol?To which DSM pages does it apply? How can Munin detect if the protocol was used in an incorrect fashion?What should it do then?The Munin write-shared protocol was designed to reduce the amount of data communication occasioned by false sharing-It applies to DSM pages that contain variables that are concurrently updated by different processes in such a way that each process accesses a disjoint subset of these variables-A runtime switch can be set to check for conflicting updates to write-shared data-The computation should be aborted whenever conflicting updates are detected, as its results are likely to be incorrect.a subblocking factor of 4 and 64 bit addresses,full 6*8=48bytes. What is the difference between a superpage and a subblock? All the pages of a superpage are brought together in main memory and expelled together from main memory while pages in a subblock do not have to be simultaneously present in main memory.The UNIX fork its costly because it has to copy the contents of the parent address space into the child address space. Berkeley Unix has a vfork() primitive that shares the same address space between parent and child until the child process does an exec(). Mach uses copy-on-write.When would a block be moved from the T1 list to the T2 list?When it is referenced a second time Why do we say that ARC is self-tuning? Because it has no user-tunable parameter.ARC is scan-resistant because a scan of all the files in the system will not alter the size of the portion of the cache dedicated to files that have been referenced more than once. Explain how the exokernel approach can satisfy the needs of (a) applications that want to manage their own resources and (b) applications that prefer to let the system manage these resources The exokernel allows processes that want to manage their own resources to manage themselves. Processes that prefer to let the system manage their own resources can use a libOS. The main problem with the LRU cache replacement policy is that it is not self-tuning. FALSE, LRU has no tunable parameter LibOSes cannot be trusted by the exokernel.TRUE, they are parts of each individual user process