Compare the advantages and disadvantage of preemptive over a non-preemptive scheduling strategy? => The advantage of a preemptive scheduler is that it allows for more equitable sharing and response time when running multiple processes/threads. A preemptive scheduler also will not allow a CPU bound or runaway thread to essentially shut down the system. << >> A disadvantage of a preemptive scheduler is that there is more bookkeeping needed to handle interrupting the thread because the thread is no longer reentering the system (via a system) at well-defined points. In most systems for programming considerations the first page in a processes virtual address is usually marked as invalid. Why should this be true? (Hint: think pointers) And why isn't this a complete solution to the problem it's trying to solve? => The first page is often marked as invalid to catch cases where a programmer has incorrectly coded a dereference of a NULL pointer. This is an incomplete solution for all invalid pointer accesses since many other pages may be valid but for which there should be no valid pointer in the program. It is also incomplete when the dereference through NULL occurs with a large offset (say, a big field offset in a struct). ●● **_F_** The size in bytes of a variable of type (char *) is 1 byte. **_T_** In a C program the expressions a[i] and *(a + i) are equivalent. **_T_** The strdup function calls malloc. **_F_** On a 1-CPU computer, a program that runs in time T will run in time T/n if decomposed in n threads. **_F_** The "S" in SMP stands for Simultaneous. => Symmetric. **_F_** A call to "strcmp" will show in the truss output. **_T_** The time command in mentor could show that the user time is larger than the real time. **_T_** A long time quantum may cause a program to finish sooner. **_T_** The arguments of a system call are checked in kernel mode. **_F_** The file descriptors of a process are closed when a process calls execvp(). **_T_** A process that uses pipes may hang due to unclosed file descriptors. **_T_** A program that runs with non-preemptive scheduling runs faster than one with preemptive scheduling. **_F_** Most of the processes' CPU bursts do not finish before a context switch. **_F_** Programs that run round-robin scheduling have faster average response time than programs that run SJF. **_F_** When a process calls fork, the number of open file objects in the kernel is duplicated. **_F_** POSIX threads are better than Solaris threads because the former are faster. **_T_** The input/output redirection to files can be done by the child. **_T_** Kernel threads in a process share the same file descriptors. **_F_** A section of code that is guarded by sema_wait/sema_post calls can be executed by only one thread at a time. **_F_** A process table entry contains one set of registers for each user and kernel level thread in a process. **_T_** The two primary purposes of an operating system are to manage the resources of the computer and to provide a convenient interface to the hardware for programmers. **_F_** When the CPU is in kernel mode, the CPU is prevented from accessing areas of memory that are not owned by the current process. **_T_** An interrupt table contains the addresses of the handlers for the various interrupts. **_F_** Each thread of a process has its own virtual address space. **_F_** Every time a clock interrupt occurs, a context switch from one process to another is performed. **_F_** In a virtual memory system, a virtual address and a physical address must be the same size. **_T_** In a virtual memory system, a virtual page and a physical page frame must be the same size. **_T_** In a multiprogrammed system using partitioning (i.e. each process occupies a contiguous block of memory), addresses can be relocated at run time using base registers. **_F_** In a multiprogrammed system using partitioning, the Best Fit strategy (where a process is placed in the smallest hole in memory large enough for the process) provides the most effective use of memory. **_T_** The operating system kernel consists of the portion of the operating system that is always running. **_T_** The difference between a program and a process is that a process is an active entity, whereas a program is a passive entity. **_T_** System calls can be run only in kernel mode. **_F_** Interrupts can be triggered only by hardware. **_T_** In UNIX systems, the exec() system call causes the calling process to run a different program. **_F_** Named pipes in UNIX require a parent-child relationship between the communicating processes. **_F_** Concurrency means that multiple tasks can execute simultaneously if multiple cores or processors are available, whereas parallelism means that multiple tasks can achieve progress via serial execution on a single core or processor. **_T_** It is possible to create a thread library without any user-level support. **_F_** Each thread has its own register set and virtual memory space. **_T_** Practical solutions to the critical section problem require hardware support. **_T_** A microkernel is a kernel that is stripped of all nonessential core components. **_T_** A system call is triggered by software. **_F_** Privileged instructions can be executed directly in user mode. **_T_** The OS scheduler is invoked when a process finishes execution. The kernel will gain control of the CPU so it can select another process to execute on it. **_T_** Two devices can each generate interrupts at the same time. This is why an interrupt manager is needed. **_T_** The kernel gets back CPU control, when a process makes a blocking read() call. The kernel will gain control of the CPU so it can select another process to execute on it. **_F_** A semaphore is a synchronization primitive that can be used in user level threads. Semaphores requires system calls and therefore it is not suitable for thread libraries used by ULT. **_F_** In programs, we can use pthread locks in place of semaphores to perform the same functionality. Semaphores can be used for synchronization in addition to mutual exclusion. However, locks can only be used for mutual exclusion. **_T_** Suppose that a multi-threaded program requires a lot of I/O, it is better to (from program execution time) to have kernel-level threads. Kernel-level is better so that if a thread gets blocked by an I/O request (which is usually a system call), other threads can continue executing. **_T_** A CPU bound process that is allocated memory pages fewer than its working set will behave like an I/O bound process. if a process is allocated memory pages that are a lot fewer than its working set, then thrashing will occur and many page faults will happen. The process will be blocked so often and will behave like an I/O bound process. **_F_** The purpose of the TLB is to place the contents of the most frequently accessed page frames in the L2 cache of the CPU. The TLB is a dedicated cache. **_F_** In a computer system with a single CPU and we do not know the runtime of each process, we can use the Shortest Remaining Time Next scheduling. The SRTN algorithm requires knowledge of the estimated execution time of each job. **_T_** Contiguous allocation of files leads to external disk fragmentation. Contiguous allocation leads to external disk fragmentation since disk are divided into blocks the only disk space waste is in the block itself.

●● Enumerate the fields of a process table entry: Process ID, Process state, Saved Registers, File Descriptors, Page Table System call: Process control, File management, Device management, Information maintenance, Communications, Protection Mention the checks done by the kernel during the open() system call: • File Permissions: If the file is opened in write-mode, the user should have write permissions to the file either as user, group or others. The same for read-mode. • If the file is opened with the flag O_CREAT and the file does not exist, the user should have write permissions to the directory the file will be created into. What are the steps involved in a context switch? => Save registers in process-table entry • Jump to timer interrupt-handler • Change the state of the process from running to ready • Choose the next process to run from the ready processes. • Set this process in running state. • Restore the registers of the next process in the CPU • Return from interrupt. Disadvantages and advantages of using kernel threads vs. user threads? ***Advantages of kernel-threads:*** • User-level threads use non-preemptive scheduling so one non-cooperative thread may hang others. That does not happen with kernel-threads that use preemptive scheduling. • Kernel-level threads may use multiple processors in SMP machines. ***Disadvantages of kernel-threads:*** • Context Switch of kernel threads is slower because it needs to switch to kernel-mode • Programs may take longer because of the context switch overhead of preemptive scheduling. Advantages and disadvantages of using threads vs. using processes? ***Advantages of threads:*** • Context switches among threads is faster than among processes. • Thread creation is faster than process creation ***Disadvantages of threads:*** • If one thread crashes the entire process crashes. • Thread synchronization is necessary to prevent the corruption of shared data structures. What factors have to be considered when choosing the length of a quantum time? Response Time, Context Switch Overhead, Average CPU burst length What does the CPU hardware do when a trap or interrupt occurs? Describe the specific steps that the hardware performs (but just the hardware - not the OS or any process). When a trap or interrupt occurs, the CPU pushes the program counter and other registers (e.g. the stack pointer, status register, etc.) onto the stack, switches to kernel mode, and jumps to the address contained in the interrupt table entry corresponding to the interrupt that occurred. When a process executes a fork() system call, a duplicate process (i.e. the child process) is created. How does the code in the processes – since it is identical in both the parent and child processes – know which process is the parent and which is the child? =>In the child process the fork() call returns 0, whereas in the parent process the fork() call returns the PID of the child process. Give a simple example of code that would operate differently in the parent and in the child. => `if (fork() == 0) printf("I'm the child.\n"); else printf("I'm the parent.\n");` What is the CPU scheduler? Whenever the CPU becomes idle, the OS must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process. Describe the four conditions that CPU scheduling decisions may take place. => CPU scheduling decisions may take place when a process: (1) Switches from running to waiting state: • The result of an I/O request • Wait for the termination of one of the child processes (2) Switches from running to ready state: An interrupt occurs (3) Switches from waiting to ready: Completion of I/O (4) Terminates. What is a dispatcher? The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler; this involves: • switching context • switching to user mode • jumping to the proper location in the user program to restart that program. Describe the five popular criteria using in CPU scheduling algorithm? CPU utilization, Throughput, Turnaround time, Waiting time, and Response time. Describe the five popular criteria using in CPU scheduling algorithm. Describe the benefits of thread pools => (1) Servicing a request with an existing thread is usually faster than waiting to create a thread. (2) A thread pool limits the number of threads that exist at any one point. This is particular important on systems that cannot support a large number of concurrent threads. Describe 4 benefits of using multithreading => Responsiveness, resource sharing, economy, and utilization of MP Architectures. In the original UNIX operating system, a process executing in kernel mode may not be pre-empted. Explain why this makes (unmodified) UNIX unsuitable for real-time applications. => In a real-time system, the OS must always be able to pre-empt a lower-priority process in order to allow a higher-priority process to run. This is not possible in unmodified UNIX if the lower-priority process is executing a system call when the higher-priority process needs to run. Assume process P1 has threads T1 and T2. Will T1 and T2 continue to run after P1 exits? => No - because T1 and T2 are part of the address space of P1, the threads have no independent existence and will "disappear" when P1 exits. (This contrasts with child processes, which do have an independent existence from the parent process.) Give a reason why a context switch between threads may be cheaper than a context switch between processes. => A process has a much larger "state" than a thread, because a thread shares much of its state with the process that created it (e.g., virtual address space, file descriptor table, etc.). Thus, a switch between two threads within a process typically will require less time to save/restore state than a switch between two processes. Briefly describe what is involved in a process context switch. Process p is running. The current state of p is in the CPU, while p's PCB contains an out-of-date copy. The OS interrupts p and saves the current CPU state in p's PCB. The OS chooses process q to continue. The current CPU state is overwritten with q's last state from q's PCB. As q continues running, the saved state in q's PCB gets out of date. q's current state is maintained in the CPU. Next, the OS interrupts q and saves the current CPU state in q's PCB. The OS then restarts p by restoring p's saved state in the CPU. As p continues running, the saved state in the PCB gets out of date. In UNIX programming, the fork() system call creates a child process that is a clone of the parent process. What is the one difference between the parent process and the child process when the fork is complete? A parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process. On the success of a fork() system call, the PID of the child process is returned to the parent process and 0 is returned to the child process. On the fail-ure of a fork() system call, -1 is returned to the parent process and a child process is not created. OpenMP (for C/C++/FORTRAN) and Grand Central Dispatch (for C/C++/ Object -ive-C/Swift) are technologies with similar goals: To allow people who are not experts in parallel programming techniques to "parallelize" existing programs. Briefly describe how these technologies work, including a short example of each technique MPI, OpenMP, Libraries... Briefly describe the concept of thread-local storage, and how it can be useful.

Thread-local storage (TLS) is a computer programming method that uses static or global memory local to a thread. • The objects are non-trivial to construct• An instance of the object is frequently needed by a given thread •The application pools threads, such as in a typical server (if every time the thread-local is used it is from a new thread, then a new object will still be created on each call!) •It doesn't matter that Thread A will never share an instance with Thread B; •It's not convenient to subclass Thread. Provide 2 kind of programming in which multithreading does not provide better performance than a single-threaded solution? (1) Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables and current working directory. Which of the following components of program state are shared across threads in a multithreaded process (Register values, Heap memory, Global variables, Stack memory)? The thread of multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack. Describe 5 challenges of multicore programming? Dividing activities, balance, data splitting, data dependency, testing and debugging.

●●Deadlocks: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Resources: •Preemptable resource: can be taken away from process with no negative effects •Non-preemptable resource: cannot be taken away without causing a failure. 4 conditions must hold for a deadlock to occur: •Mutual exclusion condition -> each resource is either currently assigned to exactly one process or is available •Hold and Wait -> processes holding resources that were granted earlier can request new resources •No-preemption -> resources given to a process cannot be taken away from a process; must be released by the process holding them •Circular wait -> must be a circular list of 2 or more processes, each of which is waiting for a resource held by the next member of the chain. 4 strategies to deal with deadlocks: •ignore the problem •detection and recovery •dynamic avoidance by careful resource allocation •prevention by structurally negating one of the four conditions. Deadlock Detection Algorithm: •Deadlock Detection with One Resource of Each Type • Deadlock Detection with Multiple Resources of Each Type •Banker's Algorithm for a Single Resource •Banker's Algorithm for Multiple Resources Deadlock Prevention: • Mutual Exclusion Condition (avoid assigning a resource unless necessary and try to make sure that as few processes as possible may claim the resource)•Hold and Wait Condition (require all processes to request resources before starting execution; if all resources are available, process will be allocated what is needed and can run to completion, otherwise nothing will be allocated and the process just waits; another way is to require a process requesting a resource to first temporarily release resources it holds then get everything it needs at once) •No-Preemption (some resources can be virtualized) •Circular Wait (have a rule where a process can have only one resource (must release one to get another) ; or have processes number requested resources in order) Starvation: a job cannot be completed due to certain policies, and it ends up being starved. •A deadlocked state occurs whenever every process in a set is waiting for an event that can only be caused by another process in the set •One necessary condition for deadlock is mutual exclusion, which states that at least one resource must be held in a non-sharable mode. •One necessary condition for deadlock is hold and wait, which states that a process must be holding one resource and waiting to acquire additional resources. •One necessary condition for deadlock is no preemption, which states that a resource can be released only voluntarily by the process holding the resource. •One necessary condition for deadlock is circular wait, which states that there is a chain of waiting processes whereby P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, and Pn is waiting for a resource held by P0. •The witness software product is a lock-order verifier that uses mutual-exclusion locks to protect critical sections. •In a system resource-allocation graph, a directed edge from a process to a resource is called a request edge. •A cycle in a resource-allocation graph is a necessary and sufficient condition for a deadlock in the case that each resource has exactly one instance. •To handle deadlocks, operating systems most often pretend that deadlocks never occur. •_T_ An unsafe state may lead to a deadlocked state. Which of the following data structures in the banker's algorithm is a vector of length m, where m is the number of resource types? Available

Assume there are three resources, R1, R2, and R3, that are each assigned unique integer values 15, 10, and 25, respectively. What is a resource ordering which prevents a circular wait? R2, R1, R3 •A CPU could be preempted from a process. Explain what has to happen for a set of processes to achieve a deadlocked state? For a set of processes to exist in a deadlocked state, every process in the set must be waiting for an event that can be caused only be another process in the set. Thus, the processes cannot ever exit this state without manual intervention. Describe the four conditions that must hold simultaneously in a system if a deadlock is to occur? For a set of processes to be deadlocked: at least one resource must remain in a nonsharable mode, a process must hold at least one resource and be waiting to acquire additional resources held by other processes, resources in the system cannot be preempted, and a circular wait has to exist between processes. What are the three general ways that a deadlock can be handled? A deadlock can be prevented by using protocols to ensure that a deadlock will never occur. A system may allow a deadlock to occur, detect it, and recover from it. Lastly, an operating system may just ignore the problem and pretend that deadlocks can never occur. What is the difference between deadlock prevention and deadlock avoidance? Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions for deadlock cannot hold. Deadlock avoidance requires that the operating system be given, in advance, additional information concerning which resources a process will request and use during its lifetime. Describe 2 protocols to ensure that the hold-and-wait condition never occurs in a system? One protocol requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls. An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. •Before it can request any additional resources, however, it must release all the resources that it is currently allocated. What is one way to ensure that a circular-wait condition does not occur? One way to ensure that this condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing order of enumeration. This can be accomplished by assigning each resource type a unique integer number to determine whether one precedes another in the ordering. What does a claim edge signify in a resource-allocation graph? A claim edge indicates that a process may request a resource at some time in the future. This edge resembles a request edge in direction, but is represented in the graph by a dashed line. Describe a wait-for graph and how it detects deadlock? If all resources have only a single instance, then we can define a deadlock-detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges. To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph. What factors influence the decision of when to invoke a detection algorithm? The first factor is how often a deadlock is likely to occur; if deadlocks occur frequently, the detection algorithm should be invoked frequently. The second factor is how many processes will be affected by deadlock when it happens; if the deadlock-detection algorithm is invoked for every resource request, a considerable overhead in computation time will be incurred. Describe two methods for eliminating processes by aborting a process? The first method is to abort all deadlocked processes. Aborting all deadlocked processes will clearly break the deadlock cycle; however, the deadlocked processes may have to be computed for a long time, and results of these partial computations must be discarded and will probably have to be recomputed later. The second method is to abort one process at a time until the deadlock cycle is eliminated. Aborting one process at a time incurs considerable overhead, since, after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked. Name three issues that need to be addressed if a preemption is required to deal with deadlocks? First, the order of resources and processes that need to be preempted must be determined to minimize cost. Second, if a resource is preempted from a process, the process must be rolled back to some safe state and restarted from that state. The simplest solution is a total rollback. Finally, we must ensure that starvation does not occur from always preempting resources from the same process. Describe how a safe state ensures deadlock will be avoided? A safe state ensures that there is a sequence of processes to finish their program execution. Deadlock is not possible while the system is in a safe state. However, if a system goes from a safe state to an unsafe state, deadlock is possible. One technique for avoiding deadlock is to ensure that the system always stays in a safe state. This can be done by only assigning a resource as long as it maintains the system in a safe state.

●●Processes, Threads, and Synchronization

Waiting for data to be read from a disk => Blocked Busy waiting for a lock to be released => Running Having just called wait() on a condition variable and the condition is not satisfied => Blocked Having just completed an I/O and waiting to get scheduled again on the CPU => Ready Suppose that two processes: P1 andP2 are running for a long time in a system. Neither process performs any system calls that might cause it to block, and there are no other processes in the system. P1 has 2 threads and P2 has 1 thread. What percentage of the CPU time will P1 get in the following two cases: (a) Threads of the processes are user level threads (ULT) => If the threads are user threads, the threads of each process map to one kernel thread, so each process will get a share of the CPU. The kernel is unaware that P1 has two threads. Thus, P1 will get 50% of the CPU. (b) Threads of the processes are kernel level threads (KLT) => If the threads are kernel threads, they are independently scheduled and each of the three threads will get a share of the CPU. Thus, the 2 threads of P1 will get 2=3 of the CPU time. That is, P1 will get 66% of the CPU. List three causes for switching from a user process into the OS kernel: System calls, Exceptions, Interrupts How does a system call differ from a normal function call? Identify two distinct differences => Possible solutions: •syscall invokes kernel code, however, function call invokes user (application) code •syscall switches the mode to privileged mode, however, the mode will remain in user mode when doing function call •the transfer is made to a fixed location in memory when a syscall is made, however, the transfer is made to a caller specified location when a function call is made Synchronization: Using atomic operations to ensure cooperation between threads Mutual exclusion: Ensuring that only one thread does a particular thing at a time Critical section: Piece of code that only one thread can execute at once.

Module 2: 1/0

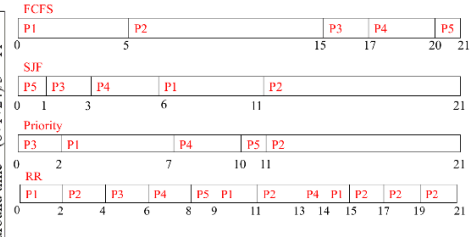
Time	P1	P3	P5	P7	P8
22	Blocked for IO	Blocked for IO	Ready/running	Blocked for IO	Ready/running
37	Ready/running	Ready/running	Swapped out	Blocked for IO	Ready/running
47	Ready/running	Ready/running	Swapped out	Blocked for IO	Terminated

Process	Burst Time	Priority
P1	5	2
P2	10	5
P3	2	1
P4	3	3
P5	1	4

FIFO:
P1 runs from 0 to 7 – turnaround time = 7
P2 runs from 7 to 8 – turnaround time = 8
P3 runs from 8 to 24 – turnaround time = 24
Average turnaround time = (7+8+24)/3 = 13

SPN: P2 runs from 0 to 1 – turnaround time = 1
P1 runs from 1 to 8 – turnaround time = 8
P3 runs from 8 to 24 – turnaround time = 24
Average turnaround time = (8+1+24)/3 = 11

Process	Burst Time (Time Unit)
P1	7
P2	1
P3	16



In a system resource-allocation graph, a directed edge from a process to a resource is called a request edge.
To handle deadlocks, operating systems most often pretend that deadlocks never occur.
How can deferred cancellation ensure that thread termination occurs in an orderly manner as compared to asynchronous cancellation? Ans: In asynchronous cancellation, the thread is immediately cancelled in response to a cancellation request. There is no insurance that it did not quit in the middle of a data update or other potentially dangerous situation. In deferred cancellation, the thread polls whether or not it should terminate. This way, the thread can be made to cancel at a convenient time.
Would a common ready queue, or per-processor ready queues, provide better support for processor affinity scheduling: A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens:
The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Sometimes hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
Assembly Routine reloads register and other state information, returns to user space to continue execution.
One necessary condition for deadlock is mutual exclusion, which states that at least one resource must be held in a non-sharable mode.
In a system resource-allocation graph, a directed edge from a process to a resource is called a request edge.
To handle deadlocks, operating systems most often pretend that deadlocks never occur.
How can deferred cancellation ensure that thread termination occurs in an orderly manner as compared to asynchronous cancellation? Ans: In asynchronous cancellation, the thread is immediately cancelled in response to a cancellation request. There is no insurance that it did not quit in the middle of a data update or other potentially dangerous situation. In deferred cancellation, the thread polls whether or not it should terminate. This way, the thread can be made to cancel at a convenient time.
Would a common ready queue, or per-processor ready queues, provide better support for processor affinity scheduling?
Explain why intel recently added a fifth level of hierarchy to the x86-64 page tables? With 5 level paging, the virtual address size increases from a 256 TB maximum to 128 PB while the physical address size threshold goes from 64 TB to 4 PB. This big set of patches to increase the virtual or physical address space capacity of the Linux kernel for future Intel x86_64 hardware can currently be found via this kernel mailing list thread.
Describe the virtual memory page fault mechanism in your own words? A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens: •The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers. •An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it. •Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Sometimes hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred. •Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem. •If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page. •If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed. •As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in. •When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state. •Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it. •Assembly Routine reloads register and other state information, returns to user space to continue execution.
_F_ Concurrency means that multiple tasks can execute simultaneously if multiple cores or processors are available, whereas parallelism means that multiple tasks can achieve progress via serial execution on a single core or processor.
_T_ It is possible to create a thread library without any user-level support.
_F_ Each thread has its own register set and virtual memory space.
_T_ Practical solutions to the critical section problem require hardware support.
_F_The value of a counting semaphore can range only between 0 and 1.
_F_ Ordering resources and requiring the resources to be acquired in order prevents the circular wait from occurring and therefore prevents deadlock from occurring.
_F_ A system in an unsafe state will ultimately deadlock.
_T_ The banker's algorithm is useful in a system with multiple instances of each resource type.
_T_The circular-wait condition for a deadlock implies the hold-and-wait condition.
_T_ The wait-for graph scheme is not applicable to a resource allocation system with multiple instances of each resource type.
_F_ Deadlock prevention and deadlock avoidance are essentially the same approaches for handling deadlock.
_F_ Protocols to prevent hold-and-wait conditions typically also prevent starvation.
_F_ If a resource-allocation graph has a cycle, the system must be in a deadlocked state.