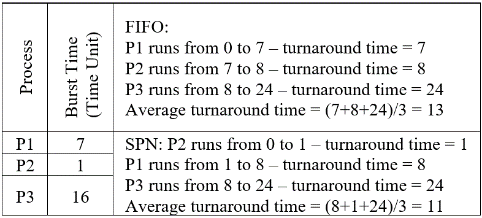
**~~●●~~** ~~Disable all interrupts => KERNEL MODE. Prevents hardware events from being serviced in a timely manner causing devices to malfunction. With preemption, the scheduler is not guaranteed to be notified at and-of-quantum. Read the time-of-day clock =>USER MODE. Reading a value does not change/expose kernel state, other processes’ state, or change external hardware state. Set the time-of-day clock => KERNEL MODE. The clock is shared and no longer has a reliable value. No computation (like timers) is reliable. Change the memory map => KERNEL MODE. Changing the may give access to the kernel memory or other processes’ memory exposing data or changing data inappropriately. Synchronize Multi-processor memory accesses => USER MODE. This takes some possible race conditions and makes them predictable. Used for spinlocks or other synchronization. Note, however, it may have a performance penalty, but cannot cause any incorrect behavior by kernel or other processes. Interrupt: Event generated by H/W external to the processor. Causes “interrupt behavior” where CPU is put into kernel mode and a jump is made to a specific location. Exception: Software generated event. Causes “interrupt behavior” Trap: “expected” exception in correct program flow. Usually used to request kernel services Fault: “unexpected” exception, may or may not result from correct flow. Sometimes (page fault) the state of the CPU/memory can be adjusted in order to let the instruction be restarted. Other times this causes the process to be terminated. Must all the General Purpose Registers be saved when handling an interrupt? Why or why not. => Interrupts are hardware generated events, like device completion or timer expiration. When an interrupt occurs, the kernel will perform some operations and then restart the thread code at the point where the interrupt occurred. If the kernel did not save the registers, it would be impossible to restart the thread correctly. In essence, interrupts are “invisible”: they happen but they do not influence the behavior of the program. Define monolithic kernel, compare to microkernel: => A monolithic kernel essentially is designed to have all the system services encapsulated in one module (image) with a tight communication and sharing of structures between the individual OS components. Microkernel which takes the different tact of providing a set of minimal functions in the kernel and implement higher level OS features in other modules (running either in user mode or kernel mode). Monolithic kernel if it can be made bug free can on average perform better than a microkernel system that has some additional communication overhead between components.~~

Draw the complete state diagram for kernel threads. Label each state, transition and describe what actions transpire for each transition. (Hint: start with a NEW state and a TERMINATED state with three other states in-between)

NEW 🡪 READY 🡪 RUNNING 🡪 TERMINATED  
 🡩 🡨 🡫  
 🡩 🡫   
 🡩 🡫   
 WAITING  
NEW 🡪 READY: processes creation  
READY 🡪 RUNNING: Processing scheduled to execute  
RUNNING 🡪 WAITING: Waiting for something to happen  
RUNNING 🡪 READY: scheduling algorithm preempts process time slice  
RUNNING 🡪 TERMINATED: Exit  
WAITING 🡪 READY: Whatever the process was waiting for has completed and now the processes is ready to run again.

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 failure 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++/ Objective-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 techniqueMPI, 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 two 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 five challenges of multicore programming? Dividing activities, balance, data splitting, data dependency, testing and debugging.

●●CPU Scheduling: •FCFS: non-preemptive •Round robin: preemptive. When the quantum time allocated for a process finishes. •Shortest Remaining Time Next: preemptive. When a new process arrives and it has a shorter service time as compared to the running process. What happens if the time slice allocated in a Round Robin Scheduling is very large? And what happens if the time slice is very small? If time slice is very large, it results in FCFS scheduling. If time slice is too small, the processor through put is reduced, since more time is spent on context switching.

Assume that P1 and P3 are ready for execution, but P2 becomes ready after 2 time

units. Can SPN still perform well? How can we modify the scheduling algorithm to still

achieve an optimal execution order that minimizes the average turnaround time. What is

the resulting execution schedule? If SPN is used, the turnaround time will be similar to that of the FIFO. P1 runs from 0 to 7, P2 runs from 7 to 8, and finally P3 runs from 8 to 24. The scheduling algorithm can be modified by preempting the running process if a shorter process is added to the system, or shortest remaining time next. In this case:

P1 runs from 0 to 2

P2 preempts P1 and runs from 2 to 3 – turnaround time =1

P1 resumes from 3 to 8 – turnaround time = 8

P3 runs from 8 to 24 – turnaround time = 24

Therefore, the average turnaround time is 11.

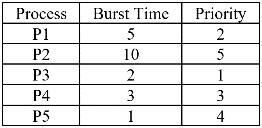
●●Deadlocks: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Resources: •Preemptable resource: can be taken away from process with no negative effects •Nonpreemptable resource: cannot be taken away without causing a failure. 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.

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

