

The exam is worth 100 points. There are 6 questions. Answer all questions.

You are to use **ONLY** the space provided, **NOT** the back of the page.

Please be brief, succinct, and to-the-point in your answers.

You will lose points for incorrect statements--even when your answer is correct!

(40 pts) **1) Circle the correct statements.**

- T** (1) Parallel systems are characterized as having multiprocessors with shared main memory and shared clock.
- T** (2) Real-time OSs perform polling-based I/O, as opposed to interrupt-driven I/O, in order for high-priority processes not to get interrupted and thereby execute within their deadlines.
- (3) In all interrupt-driven OSs, all system calls are executed with disabled interrupts.
- (4) Unix file access right scheme is advanced, allowing users to recursively grant and revoke all access right privileges that they own on a file to other users.
- T** (5) Swapper is an OS process that decides which processes to block temporarily in order to improve the performance of the OS.
- T** (6) Dispatcher is an OS process that is responsible for giving the control of the CPU to the process selected by the CPU scheduler.
- (7) Mutually exclusive use of resources by processes is a sufficient, but not necessary, condition for deadlocks.
- T** (8) In an environment where there is a single instance of each resource type, providing a total ordering of all resources and forcing processes to request resource instances in an increasing order of enumeration of their resource types guarantees a deadlock-free execution.
- (9) Deadlock prevention protocols allow for higher levels of concurrent execution than deadlock avoidance techniques.
- (10) All threads of a process share the same address space except that, in UNIX, new memory added through system calls (e.g., malloc() ) are available only to the thread that issues the system call.
- (11) Implementing threads in the user space results in slower thread executons than implementing threads in the kernel space.
- T** (12) Global variables accessed by threads via system/library calls may create conflicts resulting in erroneous use of global variables. Therefore, one solution is to keep a private copy of a global variable for each thread.

- T** (13) MT-safe procedure libraries allow concurrent thread access to their procedures, and guarantee that the use of the library procedure is “correct”, i.e., the procedure performs as specified under concurrent thread access.
- T** (14) Internal fragmentation in memory management always refers to the waste of physical memory; more specifically, it refers to the unused portion of the physical memory allocated to a process.
- T** (15) Local page replacement policy for handling page faults states that the page to be replaced (i.e., to be kicked out of the physical memory) is selected only from among the pages of the process that caused a page fault.
- T** (16) UNIX file allocation allows for triple-indirection to point to the data blocks of a single file, and, therefore, permits the creation of very large files.
- (17) Buffer cache of UNIX is a linked list of the most recently used character strings for character-special files.
- (18) Client-server model of distributed systems only uses three layers of the 7-layer network communication protocols, namely, the physical layer, the network layer, and the transport protocol layer.
- T** (19) Intel 386 architecture uses segmentation with paging.
- T** (20) An OS can be defined as a set of programs that serve as an interface between the computer hardware and the users, allowing users to use the available hardware.

\*\*\*\*\*

**For all the questions below, please use the following description.**

You work in a small company, called *EOSs-r-US* (“*Embedded OSs are us*”), which designs, builds, and sells a real-time, embedded OS (*EOSs-r-US*) together with its (off-the-shelf built) embedded hardware for managing sensors and devices in expensive car lines (such as BMW 7-series).

In the *EOSs-r-US* environment, real-time processes collectively “manage” a large number of sensors and devices (e.g., reading and saving values, collecting statistics, sending them to company headquarters) and initiate real-time I/O actions on physical devices (e.g., physical devices being brakes, cameras, headlights, rain-sensors and wipers, etc., and actions being: changing the direction of headlights or back-mounted cameras, starting and stopping back-mounted camera video streams, initiating brake actions, etc.).

The embedded hardware architecture of *EOSs-r-US* is sophisticated, containing a Pentium CPU, a GigaByte hard disk to maintain persistent data, SRAM memory of size 512Mbytes, a mobile network interface card, etc.

(15 pts) **2)** You are the main memory architect of the *EOSs-r-US* company. You have been telling your bosses that the present physical-only main memory management scheme of *EOSs-r-US* is really primitive, and you need to upgrade the memory management system with virtual memory with paging. So, you answer their questions listed below.

- (a) What will be the page fault processing speed of your new system with virtual memory with paging? (Choose one, and explain why):

\_\_\_ In the order of nanoseconds. (*O.K. as a futuristic choice*)

XX In the order of milliseconds.

\_\_\_ In the order of seconds. (*Wrong answer*)

Is your answer implementable with the currently available technology?

*Yes.*

- (b) For what type of sensors and sensor functions, the speed in (a) would be perfectly acceptable? List two sensor types and two sensor functions.

*Rain sensors; read-and-activate; deactivate (Msec latency is perfectly acceptable)*

*Temperature sensors; Turning AC/fans on/off.*

*Light Sensors; read-and-activate; deactivate*

*Tire pressure sensor; read-and-warn-driver.*

- (c) List two alternatives for implementing your process page tables, and explain why each would be a good implementation alternative.

*1. Each table entry in a register. Fastest; too expensive; small page table. If large VM is wanted then large page/frame sizes.*

*2. Associative memory. Slower than registers, much faster than main-memory-resident page table.*

*3. Main memory-based page table. Slow to search; large VM; small page/frame sizes.*

- (d) How big would you choose your (i) virtual memory size and (ii) page/frame sizes, and, more importantly, why? Give at least two reasons for the why part.

*VM size—Compute the maximum main memory needs of all applications (now and the future) and the OS: 10MB to 300MB (sensor/device apps don't need much space).*

*page/frame size—Depends on page table size, disk access speeds, app realtime constraints.*

*Here is an approximation: Page table entry count: 10K (at 2 bytes/entry, 20K bytes);*

*page size: 10K bytes → 100MB VM; VM/Physical Memory Ratio: 1/5*

*Reasons:*

*(1) We know what the apps are (and will be like), what the max memory needs of processes are, and what the VM size should be.*

*(2) Small page table size means we can use associative memory for page table access—fast address conversion and fast page fault processing.*

*(3) Small page size: less page faults (usually), and fast disk page transfer (but the dominating cost is I/O head mvmt).*

(10 pts) **3)** You are the main memory architect of *the EOSs-r-US* company. You have been telling your bosses that the present physical-only main memory management scheme of *EOSs-r-US* is really great, that there is no need for a virtual memory-based memory management scheme; and that all that one needs is

- (i) an upgrade to a *partitioned physical memory management* scheme for multiprogramming, and
- (ii) adding additional physical memory to the hardware architecture, whenever there is a need.

(a) Would this scheme require the processes to be relocatable? Why or why not?

*Always. Moving processes in and out of different partitions requires relocatable code.*

(b) Why is this a great idea? Give at least three reasons.

*--Way faster than VM management.*

*--Main memory is cheap, and we have a controlled environment, allowing us to decide memory needs.*

*--No page faults.*

*--Much better than a single partition memory management.*

(c) Why is this a terrible idea? Give at least three reasons.

*--Internal fragmentation.*

*--External fragmentation.*

*--Bounded by the physical memory space. Can't run too many processes concurrently; bounded by physical memory size.*

*--Variable-size partitioning is better.*

- (15 pts) 4) You are the CPU scheduler guru of the *EOSs-r-US* company, and you are great at ripping freeware Unix code, and adding your own CPU scheduler algorithm and code, which makes you revered within the company.
- (a) Give two reasons as to why the use of preemption would be great in your CPU scheduler.
- Without preemption, high-priority, safety-critical processes may not be able to get to the CPU, and miss their deadlines.
  
  - Preemption improves the turnaround time of high-priority processes (and deteriorates the turnaround time of other (preempted) processes).
- (b) Give two reasons as to why the use of process priorities would be great in your CPU scheduler.
- Realtime processes must use priorities to distinguish between processes that must complete within their deadlines and processes that would be good to finish within their deadlines, etc..
  
  - Priorities allow hard deadline processes to complete within their deadlines (as much as possible).
  
  - Safety-critical processes can run with the highest-levels of priorities.
  
  - Direct use of priorities starves low-priority processes. With aging priorities (i.e., monotonically increasing priorities in time), we allow low-priority processes to complete, and prevent livelocks.
- (c) Which CPU scheduling algorithm would you use, and why?
- Single ready list ordered with aging process priorities (revised at fixed intervals) and preemption. The top of the list is always the next process to execute.
    - \*\* Not fair; but high priority processes finish first.
    - \*\* This scheme requires the application designers to be considerate of their CPU use. (This answer is used for the next question)
  
  - Multilevel feedback queues with aging priorities and preemption.
- (d) Explain why the CPU utilization will be great with your design choice in (c) above? Give three reasons.
- Minimal CPU scheduling overhead. Minimal aging priority re-computation.
  
  - Highest priority process-first CPU utilization; good for high-priority processes.
  
  - Processes with hard deadlines are identified as having high priorities, and get preferential treatment by moving to the front of the ready queue (list).
  
  - The process with the most urgent need to complete gets to the CPU by preempting others, giving it a chance to complete.
- (e) Explain why the ABS (automatic braking system) sensors (or, more correctly, processes managing ABS sensors) would be best served with your CPU scheduler. Give two reasons.
- ABS application is given the highest priority and always executes first.
  
  - Whenever the ABS app comes from I/O, it gets to the CPU as fast as possible.

(10 pts) 5) What deadlock management technique would you use with your EOSs-r-US operating system and why? Give at least three reasons.

*(i) Highest-priority, safety-critical processes: Deadlock prevention by resource preemption (whatever the consequences may be on the process that loses the resource). This will take the highest-priority processes out of deadlock contention.*

*(ii) All other processes: Deadlock detection by frequent deadlock checks. All processes in this category are preemptible when resources they hold are needed by highest-priority processes.*

*--Other techniques (i.e., other protocols and avoidance-based techniques) interfere with the fastest completion of highest-priority, safety-critical, short-deadline processes, making them miss their deadlines (On the other hand, frequent deadlock detection wastes CPU cycles; a compromise!).*

*--Because we have a specialized environment where, as the OS designer, we know all the existing applications (and the near-future applications) and their resource needs and resource utilization patterns, there will be less likelihood of resource contention, making deadlocks a non-issue.*

*--Again, because we know the applications, we will have a very good idea as to where and when deadlocks can occur and among which processes (by analyzing their resource types and resource needs); so, deadlock detection will be very fast.*

(10 pts) 6) Your EOSs-r-US OS operating system actually participates in a (mobile) computer network, and communicates with processes in other computers (e.g., those at the BMW headquarters in US) routinely sending the collected statistics, as well as the “state of the car” info using socket-based communication. This allows BMW to monitor the health of your expensive car, and to schedule “at-home” maintenance.

(a) List three different types of sockets that you can use for this communication, and state one advantage of each for this environment.

--Raw sockets: Fastest possible; but company-specific transport/network protocols.

--Datagram sockets: Potentially unreliable logical connection (i.e., at the transport layer), but fast. May be useful for time-constrained transmissions, or in low-bandwidth networks as the messages get sent fast. Usually implemented on top of UDP/IP (unreliable and unreliable, respectively) protocols.

--Stream packets: Reliable logical connection. Good for info that needs to be delivered. Usually implemented on top of TCP/IP (reliable and unreliable, respectively) protocols.

--Sequence packet sockets--??

NOTES:

1) TCP/UDP are transport layer protocols, not socket types. There are a lot of protocols; see <http://www.networksorcery.com/enp/topic/ipsuite.htm>.

2) IP is a network layer protocol, not a socket type.

3) Reliability is a property that can be discussed at three layers (namely, data link, network, and transport layers), and it means different things at different layers.

4) Connection-oriented/connectionless communication is a property that can be discussed at two layers (namely, transport and network layers), and it means different things at different layers.

(b) From among the socket types listed in (a), choose one, and list two more advantages as to why your choice is a great one.

--Raw:

- Car communications mostly use simple messages, and does not need/use sophisticated protocols.
- Easy to implement.
- Fast. Too many cars and messages result in network congestion; so, we need fast communication.

--Datagram:

- Most messages are single packet (and do not need ordering).
- Little of data duplication.
- Also fast. Too many cars and messages result in network congestion; so, we need fast communication.

--Stream: This is possibly hard to defend in a mobile network.

(c) How do you recommend handling the information collected about the car when the network is not reliable, and the links are down for long periods of time?

--Queue the messages in the hard-disk (i.e., implement infinite-capacity links).