CS 354

2019年12月10日 2:27

Linked list head and tail are actual nodes

Module 7: Ports
Port system: a global pool of messages that are linked onto a free list
An individual port can be created/deleted dynamically
Each port has specific number of messages can be stored

Semaphores are used to
– Block a sender if the port is full
– Block a receiver if the port is empty

Ptsend (port Id + msg)
Wait sender sem and signal receiver sem
Dequeue one message node from free list, add message to that node and add into port queue

Ptrecv (port Id)
Wait receiver sem and signal sender sem
Dequeue first message from port queue, add into free list

Ptreset and Ptdelete -calls Ptclear
add an extra argument specifying the disposition function after port is reset or deleted

1. Each port has sequence number, it increments when port is created, reset and deleted
   Ptsend and ptrecv will check sequence number after wait returns, so it can know if port has been reset, abort if changed
2. Port that was reset has state PTLIMBO
3. Deferred rescheduling: resched_cntl(DEFER_START) at start of reset, resched_cntl(DEFER_STOP) after all operations performed

Ptclear() - clear a port	
Set ptstate as limbo Ptseq++ Walk through msg list and call dispose func on each Link msg list to free list	

Module 8: Device Interrupts
I/O using interrupts
Processor starts a device, enables interrupts and continues with other computation
The device – Performs the requested operation – Raises an interrupt on the bus
Processor hardware – Checks for interrupts after each instruction is executed, and invokes an interrupt function if an interrupt is pending – Has a special instruction used to return from interrupt mode and resume normal processing

Process is always running and interrupt is like a function occurs between two instructions
Current process executes interrupt code because context switch is expensive

Modern interrupt software:
Interrupt dispatcher: common function for all interrupts
Interrupt handler: one handler for each device, invoked by dispatcher
Upon finished, handler returns to dispatcher and then dispatcher resets instruction pointer and enables interrupts again

interupt vector array: an array of all codes of interrupt handlers
Each interrupt has a unique integer called IRQ, which used by hardware as index into interupt vector array
Controller hardware raises IRQ excpetion for device interrupt, which then invokde IRQ dispatcher and calls the correct handlers

an interrupt handler must keep interrupts disabled until it finishes touching global data structures
Null process execute nterrupt handler because its always eligible to execute
Handler cannot call wait, but can call send or signal, because interrupt can occur while null process is running

A process may be blocked waiting for the data and has higher priority than current process, we don't want current process to continue execute after return, so handler must call resched.recall that interrupt is disabled when executing dispatcher and handler, what if an interrupt calls resched and new process enables interrupts?
It's safe because: each interrupt handler leaves global data in a valid state before rescheduling and no function enables interrupts unless it previously disabled them (disable /restore)

Device Driver:
A set of funcs that performs I/O operations on a given device
Interrupt handler functions, device controller functions, read/write functions

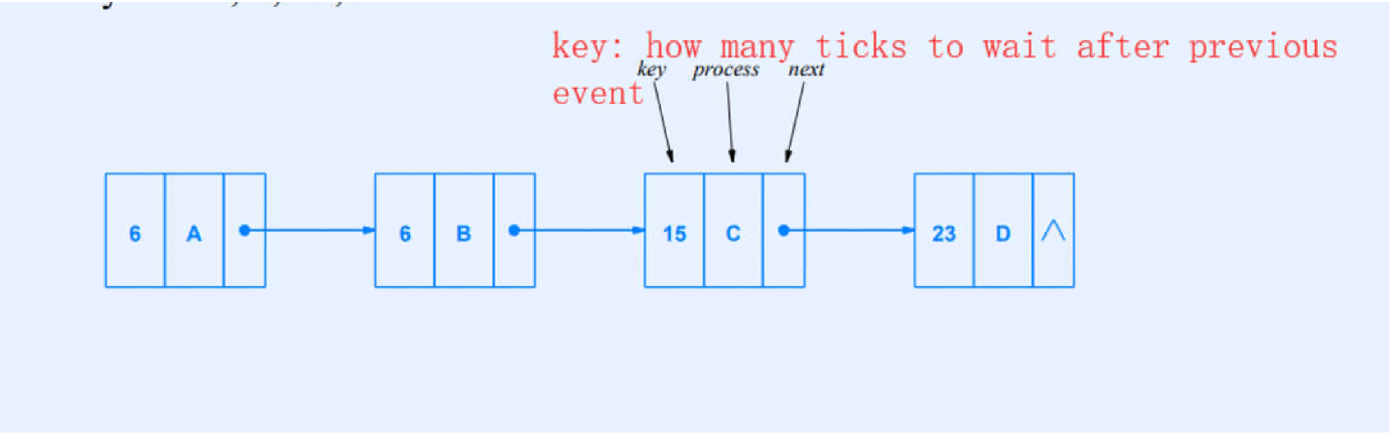
Two conceptual part in DD:

1. Functions executed by an application (read/write, data transfer)
2. Device-specific interrupt handler (interact with device to get I/O)

Clock management (timed event):
preemption event (timeslicing): Guarantees that a given process cannot run forever, switch processes
Sleep event: requested by a process to delay for a specified time

event queue: Time event list (The Delta List) - One tick is 1 ms

Insertd: Insert a process in delta list using delay as the key



A Clock Interrupt Handler
Decrement the key (or the counter), call wakeup or resched if counter reaches 0

Recvtime: Wait specified time to receive a message and return

----

Device management, Device-Independent I / O (interface)
Devices Interface: common interface for all devices

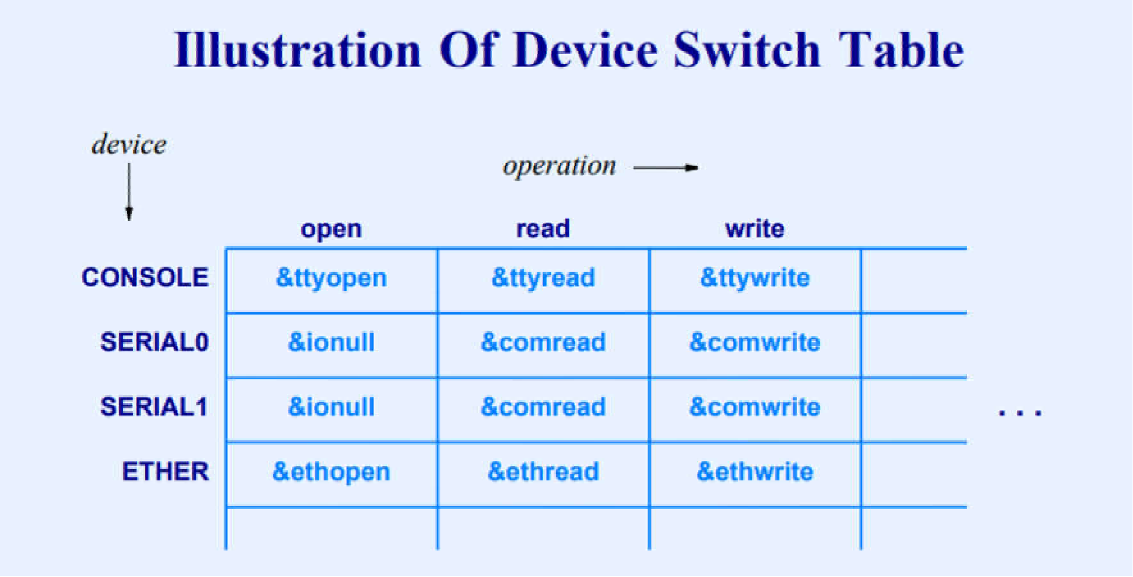


Illustration Of Device Switch Table

device ↓

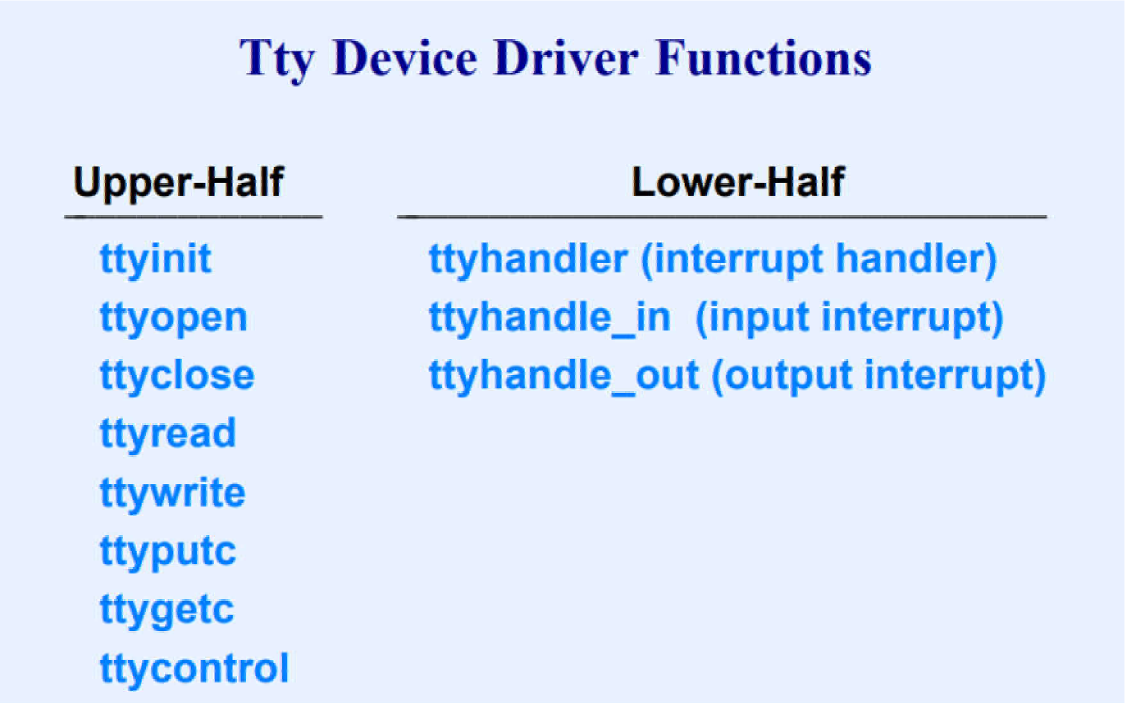
operation →

	open	read	write
CONSOLE	&ttyopen	&ttyread	&ttywrite
SERIAL0	&ionull	&comread	&comwrite
SERIAL1	&ionull	&comread	&comwrite
ETHER	&ethopen	&ethread	&ethwrite

. . .

DST specifies which device-specific function to execute when an operation is executed on a device

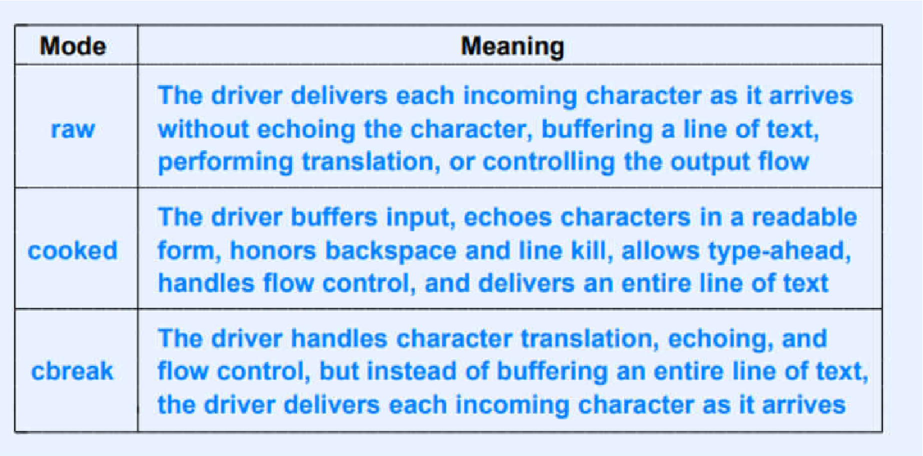
Universal Asynchronous Receiver and Transmitter (UART)
E.g keyboard as input and screen as output



Tty Device Driver Functions

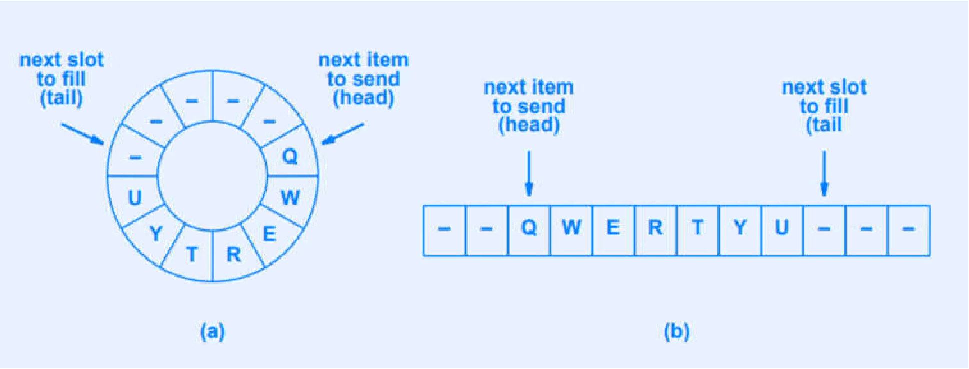
Upper-Half	Lower-Half
ttyinit	ttyhandler (interrupt handler)
ttyopen	ttyhandle_in (input interrupt)
ttyclose	ttyhandle_out (output interrupt)
ttyread	
ttywrite	
ttyputc	
ttygetc	
ttycontrol	

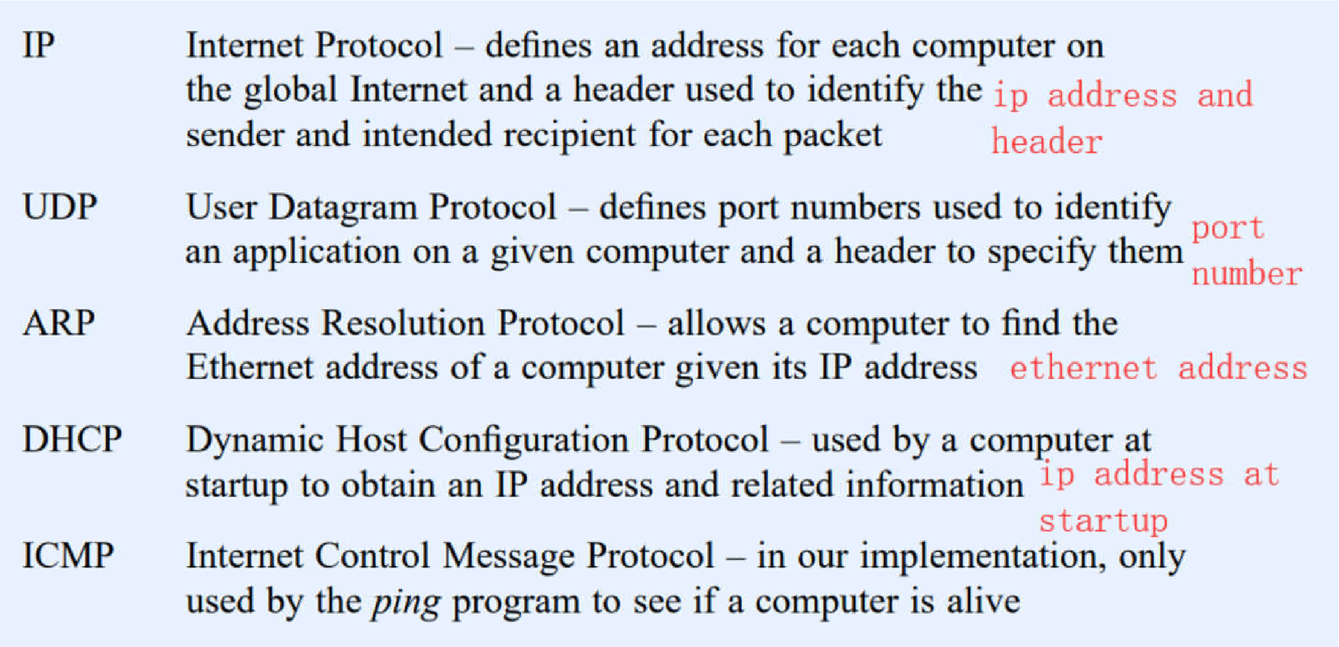
Our driver maintains independent buffers for input and output
Our driver uses semaphores to synchronize upper and lower halves



Mode	Meaning
raw	The driver delivers each incoming character as it arrives without echoing the character, buffering a line of text, performing translation, or controlling the output flow
cooked	The driver buffers input, echoes characters in a readable form, honors backspace and line kill, allows type-ahead, handles flow control, and delivers an entire line of text
cbreak	The driver handles character translation, echoing, and flow control, but instead of buffering an entire line of text, the driver delivers each incoming character as it arrives

Head and tail: next item to send and next slot to fill





IP	Internet Protocol – defines an address for each computer on the global Internet and a header used to identify the sender and intended recipient for each packet	ip address and header
UDP	User Datagram Protocol – defines port numbers used to identify an application on a given computer and a header to specify them	port number
ARP	Address Resolution Protocol – allows a computer to find the Ethernet address of a computer given its IP address	ethernet address
DHCP	Dynamic Host Configuration Protocol – used by a computer at startup to obtain an IP address and related information	ip address at startup
ICMP	Internet Control Message Protocol – in our implementation, only used by the *ping* program to see if a computer is alive	

Port number specifies which application to use in destination

udp_register - register endpoint information (remote IP, port and local UDP port)
udp_send - called by an application to send a UDP packet to a previously-registered endpoint
udp_recv - called by an application to receive a UDP packet from a previouslyregistered remote endpoint

Command Interpreter：
比如zsh，unix shell
Var in shell in by default local var, declare as global by prefixing export

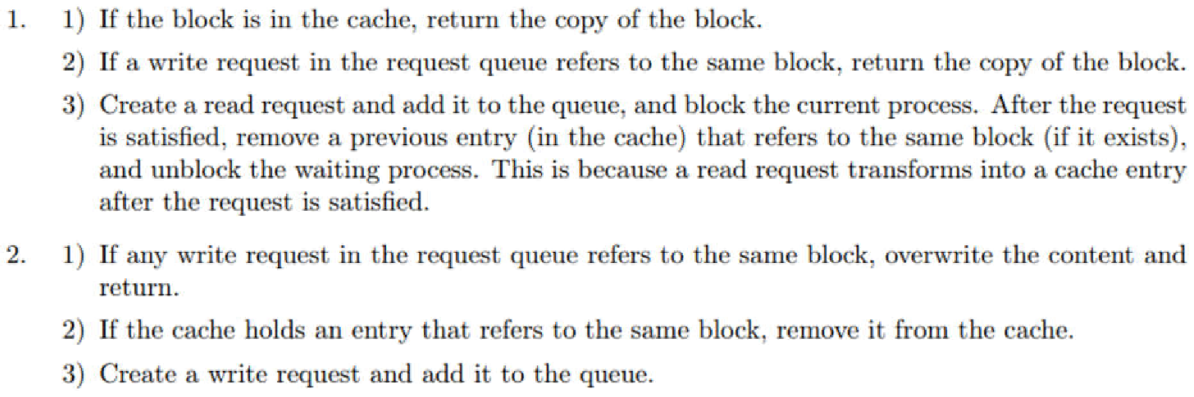
The shell always creates a process to execute a command

(A copy of the environment is kept for each process, changes only affect local copy)

```
QQQ=CS354 # Define variable QQQ
export QQQ # Export QQQ to the environment
echo $QQQ # Print the current value of QQQ
QQQ=CS503 # Redefine QQQ
echo $QQQ # Print the value of QQQ
```

Result is:

```
CS354
CS354
```



1. 1) If the block is in the cache, return the copy of the block.

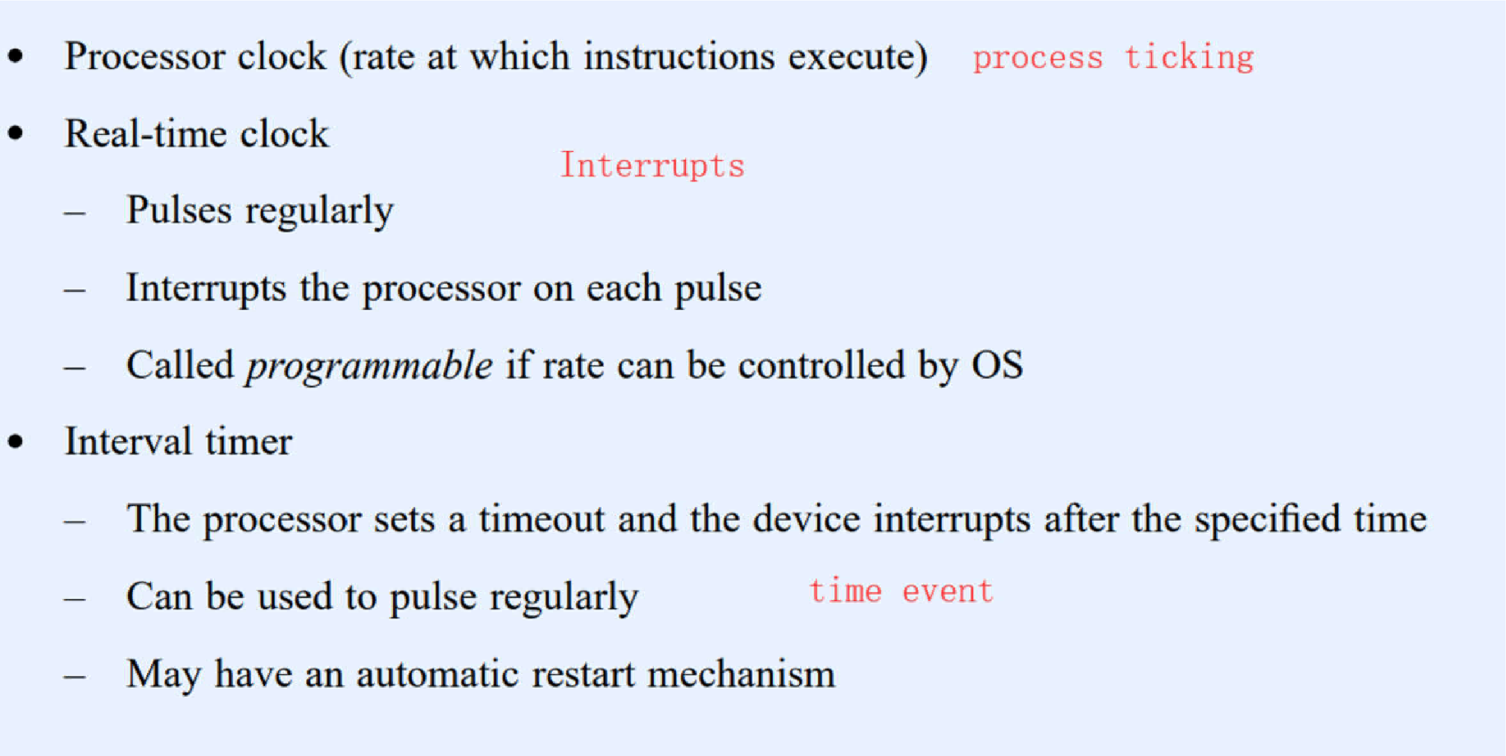
   2) If a write request in the request queue refers to the same block, return the copy of the block.

   3) Create a read request and add it to the queue, and block the current process. After the request is satisfied, remove a previous entry (in the cache) that refers to the same block (if it exists), and unblock the waiting process. This is because a read request transforms into a cache entry after the request is satisfied.

2. 1) If any write request in the request queue refers to the same block, overwrite the content and return.

   2) If the cache holds an entry that refers to the same block, remove it from the cache.

   3) Create a write request and add it to the queue.

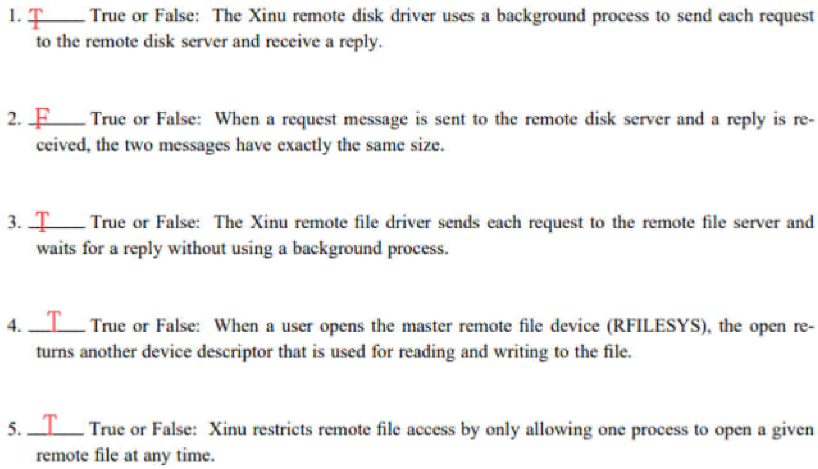


- Processor clock (rate at which instructions execute) process ticking
- Real-time clock
  - Pulses regularly
  - Interrupts the processor on each pulse
  - Called *programmable* if rate can be controlled by OS

Interrupts

- Interval timer
  - The processor sets a timeout and the device interrupts after the specified time
  - Can be used to pulse regularly time event
  - May have an automatic restart mechanism

In Xinu, a delta list allows multiple timed events to occur at the same time



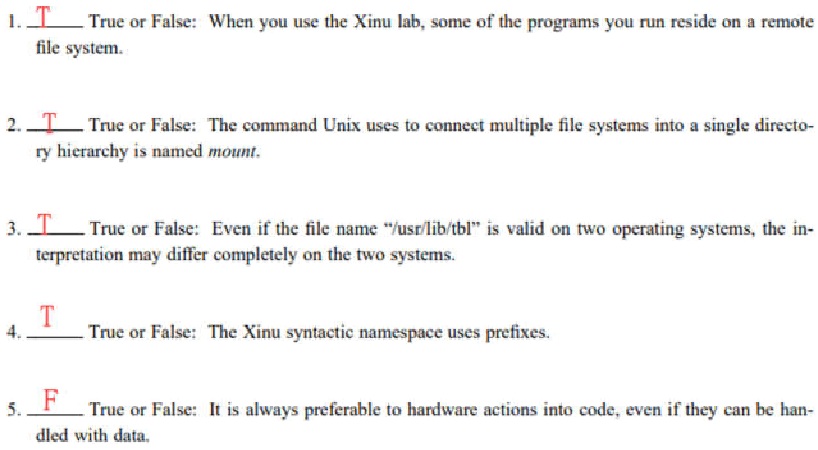
1. T True or False: The Xinu remote disk driver uses a background process to send each request to the remote disk server and receive a reply.

2. F True or False: When a request message is sent to the remote disk server and a reply is received, the two messages have exactly the same size.

3. T True or False: The Xinu remote file driver sends each request to the remote file server and waits for a reply without using a background process.

4. T True or False: When a user opens the master remote file device (RFILESYS), the open returns another device descriptor that is used for reading and writing to the file.

5. T True or False: Xinu restricts remote file access by only allowing one process to open a given remote file at any time.



1. T True or False: When you use the Xinu lab, some of the programs you run reside on a remote file system.

2. T True or False: The command Unix uses to connect multiple file systems into a single directory hierarchy is named *mount*.

3. T True or False: Even if the file name "/usr/lib/tbl" is valid on two operating systems, the interpretation may differ completely on the two systems.

4. T True or False: The Xinu syntactic namespace uses prefixes.

5. F True or False: It is always preferable to hardware actions into code, even if they can be handled with data.