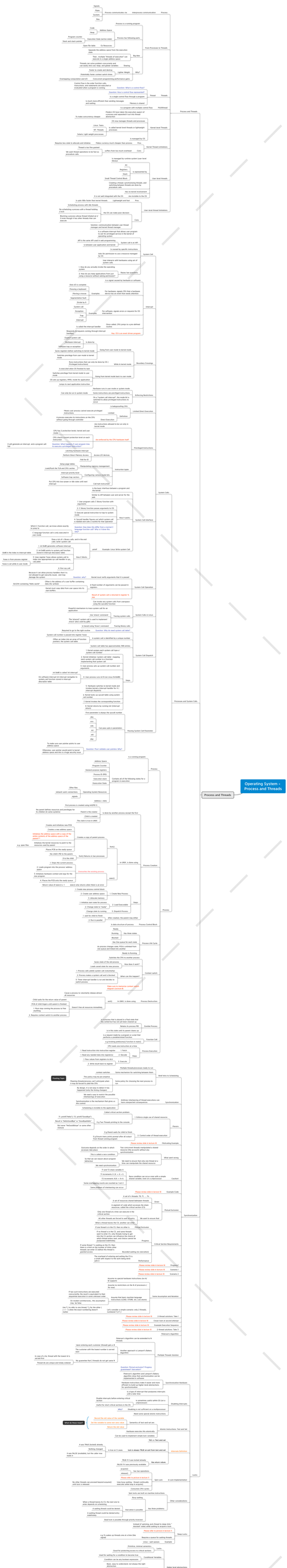
Operating System - Process and Threads
Process and Threads
Process and Threads
Signals
Pipes
Sockets
Files
Process communicate via
Interprocess communication
Process
Process is a running program
Code
Heap
Address Space
Program counter
Stack and stack pointer
Execution State (active state)
Process has following parts
Open file table
Os Resources
Process to Threads
Separate the address space from the execution
Thread
Thread is stated
Thread
Threads
Pipe - multiple "threads of execution" can
execute in a single address space
Big data
Threads can solve problem concurrency and
can easily share code, heap, and global variables
Sharing
Faster to create and destroy
Latency
Latter Merges
Why?
Potentially faster context switch time
Overlapping computation and I/O
Concurrent programming performance gain
Concurrency
Concurrency and parallelism are not the same
Concurrency and parallelism are not mutually exclusive
evaluated when a program is running
Question: What is a context Esue?
Question: How is context Flow represented?
Process
Is a single control flow through a program
Thread
Threads
Is much more efficient than starting threads
and calling
Thread is stated
Multithread
Is a program with multiple control flow
Process and Threads
Modern OS have taken the execution aspect of
a process and separated it out into thread
execution
To make concurrency cheaper
OS view threads threads and processes
Linux - Tasks
NT - Threads
Solaris - Light weight processes
Is collaborative level threads in lightweight
processes
Kernel level Threads
is managed by OS
Requires less state to allocate and initialize
Makes currency much cheaper than process
Pros
Concurrent Thread Limitations
Threads suck fine grained
Cons
We want thread operations to be fast as
procedure calls
suffers from too much overhead
Cons
is managed by runtime system (user level
library)
Requires
Stack
Is represented by
User level threads
Linux Thread Model
Creating a thread, synchronizing threads, and
scheduling threads should run entirely in
user mode
Has no kernel involvement
Is a set not integrated with the OS
Pros
Os invisible to the OS
Is calle 100x faster than kernel threads
Lightweight and fast
Pros
Scheduling process with idle threads
Os scheduling a process with a thread holding
a spin lock
Os OS can make poor decision
User level thread limitations
Blocking a process whose thread related yet a
0 state though it has other threads that can
execute
Cons
Solution: communication between user thread
manager and kernel thread manager
Is a software interrupt that drives user program
to call an operating system block or a
operating system
JRE's file same API used in web programming
Is between user application and kernel
System call is an API
Is caused by specific instructions
Asks for permission to use a resource managed
by the os
System Call
Jave libraries calls hardware using set of
system calls
1. Map file into memory
2. Write to memory
3. Read from memory
Solaris has epoll(4)
2. Has the pros keep applications from just
using a resource without asking permission
Solaris has epoll(4)
Interrupt
is a signal caused by hardware or software.
Disk I/O is complete
Pressing a keyboard
Pressing a mouse
Segmentation fault
Divide by 0
System call
Exception
Trap
Interrupt
Examples
For hardware, signals CPU that a hardware
device has an event that needs attention
For software, signals service or requests for OS
interruption
Once called, CPU jumps to a pre-defined
routine
Requests/responses coming through interrupt
handler
Kernel system call
Hardware interrupt
Is done by
Software trap or exception
Boundary Crossing
Saves registers before switching to kernel mode
Going from user mode to kernel mode
Switches everything from user mode to kernel
mode
Runs instructions that can only be done by OS (
kernel mode)
Writes to kernel mode
Is executed when OS finished its task
Switches everything from kernel mode to user
mode
Going from kernel mode back to user mode
OS sets up registers, PRTs, mode for application
Jumps to next application instruction
Enforcing Restrictions
Hardware runs in user mode or system mode
Can only be run in system mode
Some instructions are privileged instructions
Os or "system call interrupt" the mode bit is
switched to allow privileged instructions to
run
Privileged Instructions
Is implementing CPU
Limited Direct Execution
Means user process cannot execute privileged
instructions
Limited
Definition
Direct Execution
Enforces executes its instructions on the CPU
without going through control
Are instructions allowed to be run only in
kernel mode
CPU has 2 protection levels: kernel and user
mode
Are enforced by the CPU hardware itself
CPU (MMU) prevent protection level on each
instruction
Question: What hapbbly if user program tries
to execute a privileged instruction?
A will generate an interrupt and a program will
run
Loading Kernel Interrupt
Perform Direct Memory Access
Access I/O devices
Pop the top
Interrupt Stack
Instruction Types
Jetop page tables
Configuring memory management
unit
Load/Save the TLB and CPU caches
Interrupt priority level
Software trap service
Configuring various hardware
Are CPU into low power or idle (until next
interrupt)
Call hash instruction
System Calls
Is the basic interface between and program
and os
Provides a set between os and server for the
web
Is between os and server for the
web
Is the system call
1 User program calls C library function with
arguments
Is a set of system calls
2. C library function passes arguments to OS
3. Execute special trap instruction to trap to system
call handler
Trap to kernel
Step 4 works
System Call Interface
4. Special Handler figures out which system call
is needed and calls a handler for that operation
Question: How does this differ from a normal
function call? Why is it more this way
Which C function call, we know where exactly
to jump to
C language function call is only executed in a
one scale
Exec()
Fork()
Wait()
Kill()
Exit()
Sleep()
Sleep for 5 Mins
Sleep
Sleep
Sleep