1. Timeslices

Problem

On a single CPU system, consider the following workload and conditions:

- 10 I/O-bound tasks and 1 CPU-bound task
- I/O-bound tasks issue an I/O operation once every 1 ms of CPU computing
- I/O operations always take 10 ms to complete
- Context switching overhead is 0.1 ms
- I/O device(s) have infinite capacity to handle concurrent I/O requests
- All tasks are long-running

Now, answer the following questions (for each question, round to the nearest percent):

1. What is the **CPU utilization** (%) for a round-robin scheduler where the timeslice is 20 ms?
2. 2. What is the **I/O utilization** (%) for a round-robin scheduler where the timeslice is 20 ms?

Solution

1. CPU utilization
    - ((10 * 1ms) + 1 * 20ms) / ((10 * 1ms) + 1 * 20ms + 11 * 0.1ms) = 30 / 31.1 = 97%
2. I/O utilization
    - the first I/O request is issued at 1 ms time
    - the last I/O request is issued at (1 ms + 9 * 1ms + 9 * 0.1) = 10.9 ms, and it will take 10 ms to complete, so it will complete at 20.9 ms
        - total time there are I/O requests being processed is 20.9 - 1 = 19.9 ms
    - the CPU bound task will complete at time 31.1 ms (which is also the total time for the set of tasks)
    - 19.9 / 31.1 = 64%

2. Linux O(1) Scheduler

Problem

For the next four questions, consider a Linux system with the following assumptions:

- uses the O(1) scheduling algorithm for time sharing threads
- must assign a time quantum for thread T1 with priority 110
- must assign a time quantum for thread T2 with priority 135

Provide answers to the following:

1. Which thread has a **"higher"** priority (will be serviced first)?
2. Which thread is assigned a **longer time quantum**?

3. Assume T2 has used its time quantum without blocking. What will happen to the value that represents its priority level when T2 gets scheduled again (lower/decrease, higher/increase, same)?
4. Assume now that T2 blocks for I/O before its time quantum expired. What will happen to the value that represents its priority level when T2 gets scheduled again (lower/decrease, higher/increase, same)?

Solution

1. t1 (in O(1) lower priority => greater numerical value; so t1 has higher priority)
2. t1 (in O(1) lower priority tasks => smaller time quantum; so t1 will have longer time quantum)
3. increase (since it used up all of its quantum without blocking, it means that it's CPU bound, so it's priority level should be lowered, which means the numerical value of its priority should be increased)
4. decrease (since it blocked, it's more I/O intensive, will get a boost in priority, which means that the numerical value will be decreased)

3. Hardware Counters

Problem

Consider a quad-core machine with a single memory module connected to the CPU's via a shared "bus". On this machine, a CPU instruction takes 1 cycle, and a memory instruction takes 4 cycles.

The machine has two hardware counters:

- counter that measures IPC
- counter that measures CPI

Answer the following:

1. What does IPC stand for in this context?
2. What does CPI stand for in this context?
3. What is the highest IPC on this machine?
4. What is the highest CPI on this machine?

Solution

1. instructions per cycle
2. cycles per instruction
3. 4 (best case scenario: if all cores execute CPU-bound instructions, 4 instructions can be completed in a single cycle)
4. 16 (worst case scenario: if all cores issue a memory instruction at the same time, the instructions will contend on the shared bus and the shared memory controller. so one of them can take up to 4 * 4 cycles = 16 cycles to complete)

4. Synchronization

Problem

In a multiprocessor system, a thread is trying to acquire a locked mutex.

1. Should the thread spin until the mutex is released or block?
2. Why might it be better to spin in some instances?
3. What if this were a uniprocessor system?

Solution

1. If owner of mutex is running on a another CPU -> spin; otherwise block
2. Owner may release mutex quickly; may be faster to wait for owner to complete than to pay overhead for context switching twice.
3. On uniprocessor, always block, there is no way for owner to be running at the same time, since there is only CPU.

5. Spinlocks

Problem

For the following question, consider a multi-processor with write-invalidated cache coherence.

Determine whether the use of a dynamic (exponential backoff) delay has the **same, better, or worse performance** than a test-and-test-and-set ("spin on read") lock. Then, explain why.

Make a performance comparison using each of the following metrics:

1. Latency
2. Delay
3. Contention

Solution

1. same. if you look at the pseudocode for the two algorithms, if the lock is free then exactly the same operations will be executed in both cases.
2. worse. because of the delay, the lock may be released while delaying, so the overall time that it will take to acquired a freed lock will be potentially longer
3. better. in the delayed algorithm, some of the invalidations are not observed because of the wait, so those will not trigger additional memory accesses, therefore overall memory bus/interconnect contention is improved.

6. Page Table Size

Problem

Consider a 32-bit (x86) platform running Linux that uses a single-level page table. What are the **maximum number of page table entries** when the following page sizes are used?

1. regular (4 kB) pages?
2. large (2 MB) pages?

Solution

1. regular 4kB pages: 32 - 12 bits = 20bits => need 2^20 entries for each virtual page
2. large 2MB pages: 32 - 21bits = 11 bits => need 2^11 entries for each virtual page

7. PIO

Problem

Answer the following questions about PIO:

1. Considering I/O devices, what does PIO stand for?
2. List the steps performed by the OS or process running on the CPU when sending a network packet using PIO.

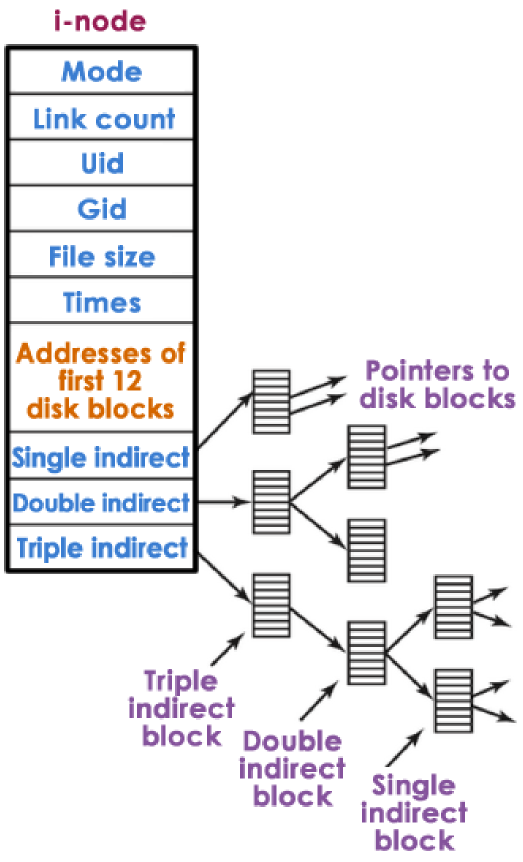
Solution

1. programmed I/O
2. write 'command' to the device to send a packet of appropriate size; copy the header and data in a contiguous memory location; copy data to NIC / device data transfer registers; read status register to confirm that copied data was successfully transmitted; repeat last two steps as many times as needed to transmit the full packet, until end-of-packet is reached.

8. inode Structure

Problem

Assume an inode has the following structure:



Also assume that **each block pointer element is 4 bytes**.

If a block on the disk is 4 kB, then what is the **maximum file size** that can be supported by this inode structure?

Solution

For 4KB = 4kB / 4 = 1k elements: (12 * 4kB) + (1k * 4kB) + (1k^2 * 4kB) + (1k^3 * 4kB) = …

Approx (using last term only) = (2^10)^3 * (2^2)*(2^10) = 2^(30+2+10) = 4TB

9. RPC Data Types

Problem

A RPC routine get_coordinates() returns the N-dimensional coordinates of a data point, where each coordinate is an integer.

Write the elements of the C data structure that corresponds to the 3D coordinates of a data point.

Solution

The return value will be an array of N integers. Since it's not specified what is the value of N, the type of the return argument in XDR will be defined as a variable length array of integers:
int data<>

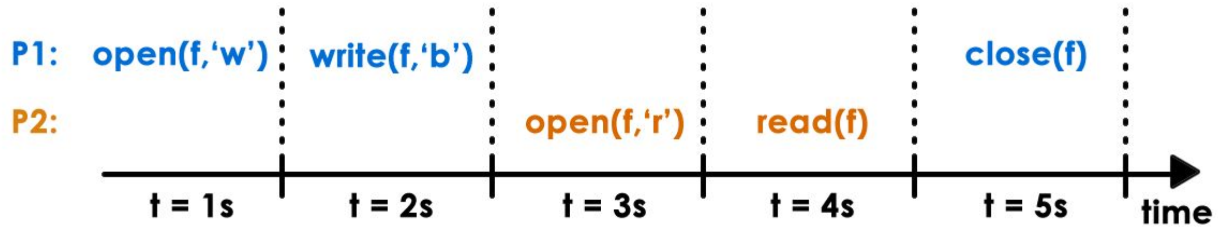
When compiled with rpcgen, this data type will correspond to a data structure in C with two elements, an integer—len—that correspond to the value of N, and a pointer to an integer—val—which will correspond to the pointer to the integer array.

- int len (or length)
- int * val (or value…)

10. DFS Semantics

Problem

Consider the following timeline where 'f' is distributed shared file and P1 and P2 are processes:



Other Assumptions:

- 't' represents the time intervals in which functions execute
- the 'w' flag means write/append
- the 'r' flag means read
- the original content of 'f' was "a"
- the read() function returns the entire contents of the file

For each of the following DFS semantics, what will be read -- **the contents of 'f'** -- by P2 when t = 4s?

1. UNIX semantics
2. NFS semantics
3. Sprite semantics

Solution

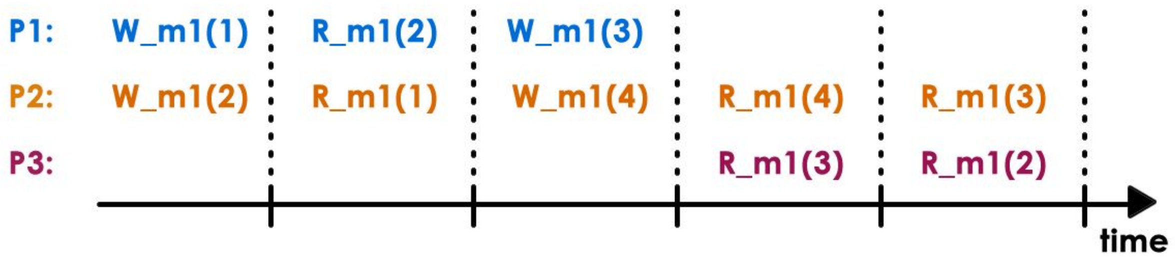
- UNIX: ab (all updates are instantaneously visible)
- NSF: a (session semantics, + period updates, but for files that's 3sec, so the write at t=2s would not propagate)

- Sprite: ab (server will check with current writer P1 what's the most recent value before returning to reader P2)

11. Consistency Models

Problem

Consider the following sequence of operations by processors P1, P2, and P3 which occurred in a distributed shared memory system:



Notation:

- R_m1(X) => X was read from memory location m1 **(does not indicate where it was stored)**
- W_m1(Y) => Y was written to memory location m1
- Initially all memory is set to 0

Answer the following questions:

1. Name all processors (P1, P2, or P3) that observe causally consistent reads.
2. Is this execution causally consistent?

Solution

1. P1 and P2 only.
2. No. W_m1(3)@P1 and W_m1(2)@ P2 are causally related (via R_m1(2) @P1), so these two writes cannot be seen in reverse order, as currently seen by P3. P3 doesn't observe causally consistent reads.

12. Distributed Applications

Problem

You are designing the new image datastore for an application that stores users' images (like Picasa). The new design must consider the following scale:

- The current application has 30 million users
- Each user has on average 2,000 photos
- Each photo is on average 500 kB
- Requests are evenly distributed across all images

Answer the following:

1. Would you use replication or partitioning as a mechanism to ensure high responsiveness of the image store?
2. If you have 10 server machines at your disposal, and one of them crashes, what's the percentage of requests that the image store will not be able to respond to, if any?

Solution

1. Partitioning. Current data set is ~30 PB, so cannot store it on one machine
2. 10% since requests are evenly distributed.