Part1

Monday, April 22, 2019 12:29 PM

Ostrich Algorithm

Not getting into deadlock

- Many situations may result in deadlock but don't have to
  - In previous example, A could release R before C requests R, resulting in no deadlock
  - Can we always get out of it this way
- Find ways to
  - Detect deadlock and reverse it
  - Stop it from happening in the first place

Detecting deadlocks using graphs

- Process holdings and requests in the table and in the graph
- Graph contains a cycle => deadlock
  - Easy to pick out by looking a it
  - Need to mechanially detect deadlock
  - Not all processes are deadlocked A, C, F, not deadlocked
  -

Process	Holds	Wants
A	R	S
B		T
C		S
D	U	S, T
E	T	V
F	W	S
G	V	U

Deadlock detection algorithm

- General idea: try to find cycles in the resource allocation graph
- Algorithm: depth-first search at each node
  - Mark arcs as they are traversed
  - Build list of visited nodes
  - If node to be added is already on list, a cycle exists
- Cycle is deadlock

Deadlock conditions:

1. mutual exclusion
   a. The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.
2. hold and wait or partial allocation
   a. The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.
3. no pre-emption
   a. The processes must not have resources taken away while that resource is being used.

Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.

4. resource waiting or circular wait
    a. A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

Recovering from deadlock

- Recovery through preemption
    - Take a resource from some other process
    - Depends on nature of the resource and the process
- Recovery through rollback
    - Checkpoint a process
    - Use saved state to restart the process if it's in deadlock
    - May present a problem if the process affects lots of external things
- Recovery through killing processes
    - Crudest but simplest way to break a deadlock: kill one of the processes in the deadlock cycle
    - Other processes can get its resources
    - Try to choose a process that can be rerun from the start
        - Pick one that hasn't run too far already

Backup = checkpointing

Eliminating mutual exclusion

- Some devices such as printer can be spooled
    - Only the printer daemon uses printer resource
    - This eliminates deadlock for printer
- Not all devices can be spooled
- Principle:
    - Avoid assigning resource when not absolutely necessary
    - As few processes as possible actually claim the resource

Attacking no preemption

- This is not usually a viable option
- Consider a process given the rinter
    - Halway through its job, take away the printer
    - Confusion ensues!
- May work for some resources
    - Forcible take away memory pages, suspending the process
    - Process may be able to resume with no ill effects
    - Disk drives - Yes and no

Attacking circular wait

- Assign an order to resources
- Always acquire resources in numerical order
    - Need not acquire them all at once!
- Circular wait is prevented
    - A process holding resource n can't wait for resource m if m<n
    - No way to complete a cycle
        - Place processes above the highest resource they hold and below any they’re requesting
        - All arrows point up!

What does FreeBSD do?

- What resources are at issue
    - Locks and semaphores: one holder at a time
    - Physical resources: not typically a concern
        - There are millions of interchangeable pages
        - Limited resources like printer only used for a short time and then released
    - Goal: prevent deadlock
        - Mechanism must be low-cost (fast and efficient)

Condition	Prevented by
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	
Circular wait	

Example two phase locking

- Phase one
    - Process tries to lock all data it needs, one at atime
    - If needed data found locked, start over
- Phase two
    - Perform updates
    - Release locks
- Note similarly to requesting all resources at once
- This is often used in databases
- What condition does this avoid?

Resource trajectories

Memory Management

Wednesday, April 24, 2019 11:52 AM

Memory is merely the process of tuning into vibrations that have been left behind in space and time

Basic memory management
- Components include
  - Operating system perhaps with device drivers
  - Single process
- Goal: lay these out in memory
  - Memory protection may not be an issue (only one program)
  - Flexibility may still be useful (allow OS changes, etc.)

Fixed partitions: multiple programs
- Fixed memory partitions
  - Divide memory into fixed spaces
  - Assign a proces to a space when its free
- Mechanisms

How many processes are enough?
- Several memory partitions
- Lots of processes wanting to use the CPU
- Tradeoff
  - More processes utilize the cpu better
  - Fewer processes use less memory
- How many processes do we need to keep

Multiprogrammed system performance
- Arrival and work requirements of 4 jobs
- CPU utilization for 1 - 4 jobs with 80% I/O wait
- Sequence of events as jobs arrive and finish
  - Number show amount of CPU time jobs get in each interval
  - More processes => better utilization, less time per process

Base and limit registers
- Special CPU registsers: base and limit
  - Access to the registers limited to system mode
  - Registers contain
    - Base: start of the process's memory partition
    - Limit: length of the process's memory partition
  - Address generation
    - Physical address: location in actual memory
    - Logical address: location from the process's point of view
    - Physical address: base + logical address
    - Logical address: larger than limit

Swapping
- Memory allocation changes as
  - Processes come into memory

    - Processes leave memory
        - Swapped to disk
        - Complete execution
    - Gray areas are unused memory

Swapping: leaving room to grow

- Need to allow for programs to grow
    - Allocate more memory for data
    - Larger stack

Tracking memory usage

- Operating system needs to track allocation state of memory
    - Regions that are available to hand out

Virtual Memory

Friday, April 26, 2019 12:44 PM

How else does the OS use virtual memory?

- Loading without the page cache
  - Copy .text segment into free memory
  - Copy .data segment into free memory
  - Link
  - Execute Program
- Loading with page cache
  - Map virtual address 0-0x1000 to P's.text
  - Map virtual address 0x1000-0x2000 to P's.data
  - Execute Program

Modern OS Memory management

- Kernel keeps track of files and their pages
- Processes have a set of maps they've made
  - But the page tables don't reflect those maps yet

Let's look at some numbers

- How big do you want your pages? 4K? 1M?
  - X86 uses 4K

Translation Lookaside Buffer = a cache of page table entries!/

TLB Invalidation
File at address space overridden, must flush data

After map, update page table, invalidate TLB (x86: invlpg), invalidate other core's TLB, wait for response; only then signal next thread

Circular buffer - write bit by bit from head until reach end, then write to head

- What to do?
  - Map the circular buffer twice next to each other
  - Only needs one cp then since you can keep writing without stop