How to Virtualize the CPU? Time sharing among processes (running program) so each runs a certain amount of time.
Challenges: Performance reduce overhead/Control over CPU.
About Scheduling: Goals: Max Throughput, Min delay, mean time to completion, mean response time
Timer interrupt: after some time, context switch to a new process
Turnaround Time:Tcompletion-Tarrival (From submission to end)
Response Time:Tfirstrun-Tarrival - Interval between first request& first response
Non-Preemptive Scheduling Pros & Cons (run 1 process to end)
Low scheduling overhead, high throughput,easy. But bad response time, bugs can freeze machine, not fair.
Preemptive Scheduling Pros & Cons: (stop 1 process to another)
Good response time, fair, good for real-time/priority scheduling
Complex, possible bad throughput, high-overhead
Real Time Scheduling – Critical operations must happen on time
Batch: Only care about maximizing throughput
Time sharing: care about fast response times to interactive programs and equal share of the CPU
1.FIFO, long process makes avg turnaround time high non-preemp
2.SJF, different process come diff times, non-preemptive
3.PSJF,preemptive shortest job first bad response good turnaround
4.Round Robin, runs a job for a time slice aiming for equal CPU/delays/scheduling. Wait time better than FIFO more expensive context switches but all processes get a chance. 5.Oracle, all knowing to know how to process in advance
6.MLFQ, uses multiple ready queues and feedback to determine priority. If P(A) > P(B), run A. If equal, run Round Robin. When a job enters, highest priority, if uses all its time reduce priority. After some time move all jobs to top of queue.
Benefits: Good performance for short interactive jobs, fair for long intensive workloads. Predictable real time response.
How does Trap work? Interrupt -> Look up trap in trap table that's loaded on start -> execute subroutine pushing PC/PS on stack -> return to user
Number associated for each interrupt.1st lvl handler sets up execution for kernel will pick 2nd level handler that actually deals with the interrupt and restores things.
About Process?. Processes have states running, ready, and blocked that are used to coordinate. The OS has a process list to hold info about all processes in the system and Process Descriptor (or PCB) which holds info about a process such as state.
Types of Processes? Orphan – when a parent dies before child.
Zombie -A process that ended but has entry in process table.
Why Copy-on-write? If there's a big data area, a copy is expensive. Set up this mode and if the resource is about to be modified, copy it & write to it. Efficient.
What are some Process API's? Each process has a PID. Fork is a system call that creates a new process almost identical, but it has its own registers/PC/PID. Non-deterministic for which one runs unless we use waitpid to synchronize but waiting for child to finish. Exec, load's new code/info for new program after forking.
How to virtualize Memory? Goal: Transparency, so programs behave like they have their own private memory. Efficiency with time and space so use HW. Protection protect processes from one another making sure each process is isolated. Create illusion that the program has its own private memory. Paging.
Memory Mechanisms
Internal Fragmentation: wasted space inside fixed size blocks, Average waste 50%.
Splitting: find a free chunk of memory for the request, split it in two where the 1st goes to caller, 2nd remains on list new start region.
Header: kept in memory containing size of allocated region. When searching for free chunk, it is size of N + size of Header
Strategies/Policies for Memory:
Fixed Partition Allocation: Pre-allocated partitions that are fixed for processes, but not good for sharing memory. Internal fragmentation.
Dynamic Partitions: variable sized chunks but can relocate & expand, subject to external fragmentation.
Best fit: search smallest that's biggest |Worst fit: find largest chunk possible |First Fit: First chunk you find that is big enough, fragments | Next Fit: Pros of first & worst
Buffer Pools: For popular sizes have special pools to manage that size
How does a resource return to free pool? The client does by calling free or close, etc. If the resource is shared increment/decrement reference count. GC issue is if you scan you might not tell if a program will use that resource. Defragmentation goes deeper more frequent garbage collection basically.
Relocating processes in memory? Address Translation changes virtual address from instruction to a physical address. HW interposes on each memory access, enables transparency.
1.Dynamic Relocation, Base register to transform virtual into physical and bounds to ensure address is in bounds.
Physical = virtual + base , internal fragmentation possible.
How do we manage Free space?
1.Paging, for a process's address space divide logical segments (code, heap, stack) into fixed-sized units called pages. Physical memory similarly divided into fixed-size slots called page frames. Utilize a page table per process to record where each virtual page is placed in physical. We use VPN to look up the PTE to find the right PFN. Within the page table, map virtual addresses to physical. Use valid bit to indicate whether translation is valid. Virtual address goes to TLB, if hit then to physical, else to page table and if there we've found the frame install to TLB and retry Otherwise, page fault find it in the disk.
2.Segmentation, pieces of the address space are relocated into memory variable sizes. They have 1 base & bound pair in our MMU for code, stack, and heap allowing us to place each segment independently in physical. Use the offset added to the base to map back to actual location in physical memory. Sharing memory segments like code using protection bits to specify level of protection.
How to speed up Paging? Translation-Lookaside-Buffers: part of MMU, hardware cache of popular virtual to physical address translations. For each memory reference HW checks TLB to see if the desired translation is there and if so, it's performed quickly without page table. If miss, we raise exception to trap handler to look at table and update TLB.
Mechanisms to Disk: We can swap pages to disk if we keep track of location and check a present bit to see if it's in disk. When we look at a PTE and not there, page fault and OS handles through disk I/O and updates page table. During this, process is blocked. Performance sucks if fault and context switch is high.
Page Replacement Policy: Goal: minimize cache misses.
AMAT = Tm(cost of access mem) +(Td(cost disk access) * Pmiss)
Compulsory Miss- cache empty to begin with
Capactity miss- miss bc cache ran out of space, had to evict
1.Optimal,replaces next page furthest in future by delaying page fault but uses the oracle.
2.FIFO,Random trash can't determine importance of blocks.
3.LRU, the idea is the near future is similar to recent past so if we haven't used a page recently there's a good chance, we won't use it soon. Utilizes locality(special and temporal), look at history to see what's important. Replace least recently used page.
4.Most frequently use and most recently used but trash.
Implementing LRU Replacement Policy: Clock Algorithm:
Organize pages in circular list, traverse through pages checking reference bit and if there's a 0 that implies page hasn't been recently used.
Dirty Page: modified page means it must be written to disk to evict
Tied to performance
Clean Page: unmodified so eviction is simple without additional I/O
Demand Paging: OS brings pages into memory when accessed "on demand". Prefetching is when you bring a page early but should have reasonable success. Clustering is grouping pending writes.
Working Set: Set of pages used by a process in a time interval, there's a sweet spot for min page faults and working set size.
Implementation: Page stealing, between processes if one has a smaller set and needs more you steal from the other that needs less.
Ideal mean-time-between-page-faults = time slice length
Thrashing: When the sum of working sets for each process exceeds available memory, constant swapping of pages and no one will have enough pages in memory.
Coordination of operations with other processes:
Goals: simplicity, convenience, generality, efficiency, robust
Flow Control:make sure fast sender doesn't overwhelm a slow receiver.
1.Coordination of operations with other processes-Network connections, IPC can be lost, limited throughput & high latency. Complex.
2. Exchange of data between processes, pipelines
Shared Memory: OS arrange processes to share read/write segments but only within local machine. However, bug can wreck.
Out of Band Signal we have a reserved channel for important requests enabling possibility for preemption for queued operations.
Threads: Each has their own stack, PC, registers. Same address space. TCB states.
Kernel Threads: in kernel mode..For User implemented level threads, kernel doesn't know anything about them.. As a result, when a user-mode thread issues a system call that blocks, all threads stop since OS doesn't know of them but there may be other threads that are runnable. Also, exploiting multi-processors, the OS cannot parallelize if it does not know a process has multiple threads. So, threads can't execute in parallel.