Quiz 3 Study Guide

Monday, November 15, 2021 12:28 PM

CHAPTER 7
Uniprocessor scheduling

Name	Description	Advantages	Disadvantages	Max-min fairness
FIFO	Do each task in the order it arrives (queue), run task until it is finished	Minimizes switching between tasks (overhead), has best throughput when we have a fixed number of tasks that only need the processor	If a task with little work to do lands behind a task with a lot of work, the system will seem inefficient. If a task arrives taking 1 second, and a few arrive behind it taking 1 ms, the response time will be over 1 second on average. Ignoring switching overhead, FIFO can become worst possible policy for average response time	Is the definition of fairness, each task waits its turn
SJF	Always schedule task with least work to do first, even ahead of those that arrived earlier, least work is defined as work left to do on a task, not the amount of work total that needs to be done	Has best average response time, there are not alternatives that are optimal in this regard	Impossible to implement as it requires knowledge of the future. Terrible response time variation as short tasks get done immediate, but long tasks may be stuck forever behind smaller tasks pushed ahead of it. Can suffer from starvation due to this. Can be gamed by programs dividing large tasks into smaller ones to try and get work done more quickly	Relatively unfair as tasks waiting depends on other tasks
RR	Tasks take turns running on the process for a limited period of time, schedular assigns processor to the first task in the ready list, sets a timer interrupt for some delay called the quantum time. At the end of quantum time, if task has not completed, the task is preempted and the processor is given to the next task in the ready list. Task is then put on back of ready list where it can wait its next turn	No possibility of starvation as it will eventually reach the front of the queue	Finishes tasks at roughly the same time in some scenarios where tasks require similar amounts of time. SJF and FIFO would be much better in this scenario for average response time. This shows FIFO and Round Robin are optimal and pessimal in different scenarios.	Somewhere between FIFO and SJF

Max-min Fairness - iteratively maximizes the minimum allocation given to a particular process (user, application, or thread) until all resources are assigned. If all processes are computer-bound, we maximize the minimum by giving each process exactly the same share of the process (Round Robin). If a processor does not need its entire allotted time we redistribute this time among the remaining processes. If one of those processes does not need its entire allotted time this is repeated.
Multi-level Feedback Queue - popular scheduling algorithm used by most commercial operating systems (Windows, Mac, Linux). Designed to achieve several goals:

Responsiveness: run short tasks quickly as in SJF
Low overhead: minimize number of preemptions, as in FIFO, and minimize the time spent making scheduling decisions
Starvation-freedom" all tasks should make progress, as in round robin
Background tasks: defer system maintenance tasks, such as disk defragmentation, so they do not interfere with user work
Fairness: assign (non-background) processes approximately their max-min fair share of the processor

MFQ is an extension of round robin, instead of using a single queue, multiple are used with different priority levels and time quantum. Tasks at a higher priority level preempt lower priority tasks, while tasks at the same level are scheduled in a round robin fashion. Higher priority levels have shorter time quantum than lower levels. Tasks are moved between priority levels to favor short tasks over long ones. New task enters at the top priority level. Every time the task uses up its time quantum, it drops a level: every time the task yields the process because it is waiting on I/O, it stays at the same level (or is bumped up a level; and if the task completes it leaves the system
Little's Law -

CHAPTER 8
Segmentation fault - An error caused when a process attempts to access memory outside of one of its valid memory regions
Zero-on-reference - Only want to pay overhead of zeroing memory if it will be used, issue for dynamically allocated heap and stack, not clean when program starts how much memory it will use; heap could be anywhere from a few KB to several GB, operating system allocates a memory region for the heap, but only zeroes the first few kilobytes. Only allows program to access zeroed memory, if program expands its heap, it will take an exception, and operating system can zero out additional memory before resuming program execution
Internal fragmentation - reducing space taken up by page table by choosing larger page from, larger page from can waste space if process does not use all memory inside the frame, this is internal fragmentation. Fixed-size chunks are easier to allocate but waste space if entire chunk is not used. Either pages or page tables are very large (wasting space due to internal fragmentation or wasting space with large page tables)
External fragmentation - new segments are placed as more memory becomes allocated, operating system may reach point where there is enough free space for a new segment, but the free space is not contiguous, this is called external fragmentation. Operating system is free to compact memory to make room without affecting applications, because virtual addresses are unchanged we relocate a segment in physical memory, this compaction can be costly in terms of processor overhead, typical server configuration would take roughly a second to compact its memory
Multi-level translation - the use of multiple different translation systems which make up the final location of the program in memory, for instance in paged segmentation, memory is segmented, but instead of each table entry pointing to a contiguous region of physical memory, each segment table entry points to a page table, which in turn points to the memory backing that segment, the segment table entry "bound" describes the page table length, that is, the length of the segment in pages, paging is used at the lowest level leading to all segment lengths being some multiple of the page size
Translation lookaside buffer (TLB) - small hardware table containing results of recent address translations, each entry in TLB maps a virtual page to a physical page, this is done by checking all entries of the TLB simultaneously against the virtual page, of there is a match process uses that entry to form the physical address, skipping the rest of the steps of address translation (TLB hit). On TLB hit, hardware still needs to check permissions in case program attempts to write to a code-only page or the operating system needs to trap on a store instruction to a copy-on-write page. TLB miss occurs if none of the entries are in the TLB match, hardware does the full address translation as normal, when completed, the physical page is used to form the physical address, and the translation is installed in an entry in the TLB, replacing one of the existing entries, replaced entry will typically be one that is not recently used