Tyler Higashihara
CSCI 475
5/2/24
Prof Chiu

Linux OS

Operating systems play a major role in managing the resources of a computer system. There exist many operating systems that anyone can choose to use such as Microsoft Windows or macOS. This paper however will be discussing Linux. Linux was created by Linus Torvald and was inspired by the operating system known as Unix. It is an open-source operating system, allowing its source code to be used freely for everyone to use, modify, and distribute. What this allows is for a widespread adoption of the OS across a large range of computing environments, from small devices to enterprise servers. Because Linux is easily customizable, any user could easily configure it to serve a different purpose. As it is free to download, this was another incentive for many users to use Linux. Linux is among the more famous OS choices that people can make and is one of the more versatile options. It already powers a significant portion of the world's computer systems. Therefore, in this paper, we will dive into the internal workings of Linux, focusing on the aspects of process and thread management, CPU scheduling policies, and memory management.

In Linux, threads play a significant role in enabling concurrent and efficient task execution. As stated by Verma[1], "Threads are lightweight execution units within a process that enable efficient and concurrent task execution," By allowing parallel processing each process will be able to execute tasks independently, we will obtain faster execution, better resource utilization, etc. Additionally, “Threads share the same memory space and resources with other threads in the process,” allowing data sharing and communication between threads to be effortless.

A Linux thread is made up of three main parts. First, the stack, a region of memory used to enable the thread’s execution. This part holds the local variables, function calls, and return addresses. Since each thread has its own stack, this allows each thread to execute independently. Second, the register set holds the thread’s execution context. This is where information about CPU registers is held. These registers store important values such as “program counters, stack pointers, and other processor-specific registers.” allowing easier manipulation of data to execute instructions. The third and final part is thread-specific data, which allows each thread to operate in its unique environment. What can be found here are variables specific to the thread itself, such as thread identifiers or thread-specific storage which help enable thread management and synchronization.

System calls are used to facilitate the act of creating and managing processes in Linux. By using system calls, programs can interact with the kernel and thereby access the OS services. The five main and system calls used are fork(), exec(), exit(), wait(), and kill(). Fork creates a duplicate child process and inherits the memory space and resources of

[1] Verma, S. (2023, August 8). Threads vs processes in linux. Scaler Topics.

the parent process. Exec replaces the current process with an entirely different program and loads it into the memory space of the currently ongoing process replacing everything. Exit is used to terminate a process and frees allocated resources. Wait makes the parent process wait for the termination of the child process. Upon termination, the parent obtains the exit status of the child and performs actions based upon the status. Kill sends a signal to a single process or a group, in which it will signal a termination, handle a specific event, or request the process actions.

A process in Linux is managed through the Linux Kernel using a data structure called task_struct. This is managed through a tasklist, which holds information on all processes currently running. The task_struct holds a variety of fields categorizing the process's attributes, and states. Every time a new process in Linux is created, "the Linux kernel allocates memory for a new task_struct in the kernel memory space." The task_struct represents the process and is used to manage and control it's execution.

The internal structure of a process is made up of 5 main parts. First the Process ID (PID), each process has an assigned identifier known as the PID. These serve as a reference to distinguish between processes. Second is the memory space, each process possesses its own memory space that is divided into different segments. There is the code segment, data segment, and stack segment. The code contains all the executable instructions, the data stores global and static variables and the stack holds function call stack and local variables. Third is the process state, this part is keeping track of the process's current condition, whether it be running, waiting, sleeping, or terminated. Fourth is process priority, which is used to decide upon its own relative importance to scheduling. Processes of higher priority are given more CPU time, lower priority are given less. The fifth and final part is the file descriptors. This part will allow a process to access files and resources.

Linux also has a process tree, which is a hierarchical structure representing the relationships between parent and child processes. The root process is generally the first process created during the system boot. Every other process outside the root process possesses a parent process from which it was created. There exist process groups where processes with a common ancestry form a group.

Moving on to CPU scheduling in Linux. According to Rajam[2], since Linux supports the concurrent execution of many tasks it also means that there will be many tasks waiting to use the CPU. This necessitates a CPU scheduling policy to enforce fair allocation of resources. Linux distinguishes between scheduling user processes and kernel tasks, both requiring its own scheduling approaches. Linux uses two different scheduling algorithms for user-defined processes. A time-sharing algorithm, which is used for preemptive scheduling of multiple processes. This is for when non-real time tasks are executed and prioritizes fairness. The second scheduling algorithm is where absolute priorities are enforced rather than fairness and is used for real-time tasks.

[2] Rajam, M. A. (n.d.). Operating system. Scheduling in Linux – Operating System.

First, the Completely Fair Scheduler (CFS). Here, Linux employs a time-sharing algorithm for preemptive scheduling of non-real-time tasks. CFS prioritizes fairness in CPU allocation among multiple processes. It operates using two priority ranges: one for real-time processes (0 to 99) and another based on the process's "nice" value (-20 to 19), where lower "nice" values are given higher priorities. CFS treats all processes fairly by allocating CPU time proportionally to each runnable process. This is achieved by calculating the runtime of each process as a function of the total number of runnable processes. Processes with lower "nice" values are assigned higher priorities and receive more CPU time, while those with higher "nice" values receive less.

For CFS to manage CPU time effectively, CFS utilizes a concept known as target latency, which makes sure that every runnable task will run at least once within a certain time interval. In addition, a minimum limit setting will make certain that processes will not be preempted excessively thereby minimizing switching costs. While CFS aims to be fair, there do exist certain scenarios where all processes cannot receive CPU time.

When real-time processes are present, Linux will use two scheduling classes, FCFS and round robin. Here, each process has a priority, the highest priority process is executed first. If there exist multiple processes with the same priority, the one with the longest wait time is scheduled. FCFS processes continue running until they exit or block, even if a higher priority process arrives. Round robin processes are preempted by higher priority processes, with lower priority processes moved to the end of the queue. Processes of equal priority in round robin will automatically time-share.

Kernel tasks, executed in kernel mode, use specialized scheduling and synchronization policies. Unlike user processes, kernel tasks are subject to being preempted within the kernel. Linux employs spinlocks and semaphores for kernel-level locking, ensuring data integrity amidst concurrent access. Furthermore, it uses synchronization architecture that allows long critical sections to execute without being disabled by interrupts, allowing greater system responsiveness.

For memory management, Linux utilizes demand paging. According to Rusling[3], Demand paging conserves physical memory by loading virtual pages into memory only when they are needed. When a process attempts to access a virtual address not in memory, a page fault occurs. If the address is valid but not in memory, the OS fetches the page from disk, which takes time. When a process needs to bring a virtual page into physical memory but there are no free physical pages available, the operating system makes room by discarding another page. If the discarded page hasn't been modified (dirty), it can easily be discarded and fetched from the original source if needed again. However, if the page has been modified, its contents need to be written back. The dirty pages are saved in a swap file, but because accessing this file is slower compared to accessing physical memory, the OS balances writing pages to disk by retaining them in memory.

[3] Rusling, D. A. (n.d.). Chapter 3 memory management. The Linux Documentation Project.

If the algorithm that chooses which pages to swap in and out isn't efficient, it can lead to thrashing, where pages are constantly swapped in and out. Therefore Linux uses a Least Recently Used (LRU) policy to choose which pages to discard. Each page in the system has an age that is updated. Pages frequently accessed are considered younger and less likely to be swapped, meanwhile, less accessed pages are older and more likely to be swapped out.

Linux utilizes memory management caches as well. The Buffer Cache contains fixed-size data buffers to block device drivers, allowing faster access to data read from or written to block devices. The Page Cache speeds up disk access by caching file contents on a page-by-page basis, which is accessed via file and offset within the file. The Swap Cache saves modified (dirty) pages in the swap file, thereby avoiding unnecessary disk operations if there exist pages that remain unmodified after being written to the swap file. The Hardware Caches may cache Page Table Entries (PTE) in Translation Look-aside Buffers (TLBs), which allows the direct translation of virtual addresses into physical ones. Whenever the TLB misses, it prompts the OS to intervene, generating new TLB entries.

In page tables, Linux assumes that there exist three levels of page tables where each level accessed contains the page frame number of the next level of Page Table. A virtual address can be broken up into different segments wherein each segment provides an offset for a Page Table. Therefore, for it to translate a virtual address into a physical one, the processor has to take the contents of each level segment and convert it into an offset into the physical page containing the page table. This is then read into the page frame number of the page table to the next level. This repeats up to three times until it finds the page frame number of the physical page with the virtual address.

In conclusion, Linux is a very versatile, open-source operating system that offers widespread adoption across a spectrum of computing environments. Its flexibility and customizability is what makes it valuable and is further prized for its open-source nature, which encourages collaboration, innovation, and widespread usage. Linux uses various mechanisms, each for its own purposes, such as system calls, task_struct, and scheduling algorithms like the Completely Fair Scheduler (CFS) for user processes, and First-Come, First-Served (FCFS) or round robin for real-time tasks, to ensure fair resource allocation and timely task execution. Furthermore, Linux's memory management strategies, including demand paging and memory management caches like Buffer Cache, Page Cache, and Swap Cache, each help to conserve physical memory and improve system responsiveness. The integration of hardware caches, such as Translation Look-aside Buffers (TLBs), helps in virtual-to-physical address translation, which contributes to overall system efficiency.

To summarize in three words, Linux is great.

References:

1. Encyclopædia Britannica, inc. (2024, April 9). *Linux*. Encyclopædia Britannica. https://www.britannica.com/technology/Linux

2. Zhi-hua, Huang. “Research and Implantation of a Kernel Thread Library Based on Embedded Linux Operation System.” *Journal of Chinese Computer Systems* (2009): n. Pag.
3. Verma, S. (2023, August 8). *Threads vs processes in linux*. Scaler Topics. https://www.scaler.com/topics/linux-thread/
4. Rajam, M. A. (n.d.). *Operating system*. Scheduling in Linux – Operating System. https://ebooks.inflibnet.ac.in/csp3/chapter/scheduling-in-linux/#:~:text=The%20job%20of%20allocating%20CPU,waiting%20to%20use%20the%20CPU.
5. Rusling, D. A. (n.d.). *Chapter 3 memory management*. The Linux Documentation Project. https://tldp.org/LDP/tlk/mm/memory.html