* 1. The output is interleaved because the order of the threads does not matter and is not set by the program, so the Operating System performs the *run()* method of each thread with a random priority set for each thread. At least on my system, there are four processors available to use. So each thread (A, B, and C) gets their own processor, and the output is determined by whichever processor finishes it’s task (printing out the name of the thread + a number) first).
  2. The java.lang.Runnable class works almost the same way as the java.lang.Thread class, except it starts all of the threads created by the program before performing each thread’s task. Except for that one difference, programs 1 and 2 are functionally equivalent.
  3. The *yield()* command tells the currently running thread to pass its priority onto the next available thread. So, the interleaving is (mostly) based on how often you run the yield command. In programs 1 and 2 there is no *yield()* command so the priority is just determined by which core completes its task first and by what the Operating System determines should go next. The *yield()* command **does** affect the program execution, it allows the user to tell the Operating System what the priority of the threads should be.
  4. Multiprogramming allows the system to store multiple functions and run them mostly concurrently off of one CPU. This allows a user to run multiple programs at once and change the priorities to allow them to get multiple tasks done at once. This came about before computers had multiple processors and the ability to run multiple programs fully concurrently. Multiprogramming allows a user to run a program that takes, for example, five hours to run and then while waiting for that program to finish the user can quickly switch the processor over to another program that is loaded up and run a, for example, 5 minutes program and then resume the 5 hour program once the 5 minute program is done. Multiprocessing appeared once a single computer could have multiple CPUs. Multiprocessing allows each CPU or processor to run its own program allowing for full concurrency in a system. A system with 4 processors can run 4 completely independent programs in the same instant.
  5. The Principle of Least Privilege (PLOP) is the paradigm of allowing the user to access only what is completely essential to run their program. This prevents users from being able to affect essential systems that should not be accessed by any random user. This is similar to having a dual mode of operations where the user is only able to perform functions while the system is in “user mode” and the operating system runs itself when in “monitoring mode”. The “operating mode” allows the operating system to operate without user interference to prevent unwanted changes to systems by the average user.
  6. DMA and “cycle stealing” allow parts of the computer system to have Direct Memory Access (DMA) and Random Access Memory (RAM) without going through the CPU. This allows the CPU to run continuously without having to stop for other processes. A single CPU can only run one process at a time, so “cycle stealing” and giving Input/Output (I/O) processes DMA allows these other processes to have access to memory without taking up the CPU’s precious processing space.
  7. A distributed operating system controlling a set of loosely coupled computers is the base idea that allows supercomputers to work. The advantages to this are *greatly* increased processing power and concurrency abilities. Each separate computer in the system can perform its own processes and the distributed operating system sets the priorities for each task and balances the processing requirements for each input program across the entire system. So, a program that takes 5 hours to run on a “regular” computer can take 5 minutes on a distributed operating system on a set of loosely coupled computers. Another advantage to a system like this is each computer can run completely independently of each other allowing multiple users to run their own programs off of one central system, with each user getting their own dedicated set of loosely coupled computers. The difficulty in setting up a computer like this is it is very difficult to manage possibly hundreds of thousands of “nodes” (processing systems) and have them interact with each other and the operating system with little or no delays. A “regular” computer only has maybe four or eight processors while a supercomputer like the one mentioned above can have hundreds of processors.
  8. Time-sharing in a system allows the user to directly communicate with the system through the use of hardware like a keyboard. This allowed the elimination of the “card-based” programming system that required the user to create and print a physical card in order to give a system programming instructions. The evolution of time-shared systems allowed for the digital storing of computer programs permitting multiple users to edit, manage, and run the same digital code. The elimination of cards allowed for increased productivity for single users that no longer had to use a “middle-man” (the cards) to interact with the computer. The advent of digitally stored code increased productivity for programming teams by allowing everyone on the team to access and test the same code.