Lecture 20: Operating System Objectives, History, and Scheduling

I. Operating System Objectives
   A. Make the computer easy to use.
      1. Provide services for program development.
         a. Support for debugging like suspend and continue signals.
         b. Provide an instruction set architecture (ISA) that makes it easy for compilers to know which instructions are for
            users (user ISA), and which for use only by the OS (system ISA).
         c. Provide application programming interface (API) which are HLL libraries for system calls.
            1) Simplify access to I/O by hiding low-level routines, e.g. printf and scanf.
            2) Make access to files easy and secure, e.g. fopen() that automatically finds file and checks permissions.
         d. Provide an application binary interface (ABI) which is a standard for binary portability between programs, e.g. cut
            and paste.
      2. Make the process starting and running a program simple, e.g. click an icon.
      3. Provide security by using password protection.
      4. Handle error detection and correction as transparently as possible, e.g. error correction on I/O and files.
      5. Adjust resource assignments to provide quick response time to the actual user, e.g. prioritize foreground processes.
   B. Manage the computer’s resources so that they are used efficiently.
      1. OS as just another program
         a. It is a program the relies on the processor and system just like any other.
         b. It must relinquish control of the CPU, and rely on the CPU to return control to it.
      2. Kernel = frequently used routines in RAM at all times, e.g. device drivers, schedulers, memory manager.

II. History of Operating Systems
   A. Two dimensions determine the jobs for the operating system
      1. Single program vs batch vs interactive
      2. Uniprogrammed vs multiprogrammed.
   B. Earliest computers ran one program at a time with no user interactions.
      1. No operating system necessary with each program using wired patchboards and technicians flipping switches and
         mounting reels of tape.
      2. One program at a time would be run manually, so scheduling was a sign-up sheet.
   C. With more RAM came the development of standard rudimentary routines for low level device interactions--the first OS.
      1. A “job” would entail
         a. Loading a compiler and the source code, compiling, while saving the compiled program as it is compiled.
         b. Loading the linker and the compiled program, link the system routines with the compiled program to create an
            executable, and while saving the linked program as it is linked.
         c. Loading the executable, and trying to run it, with any errors printed out on paper, and returned to the user.
   D. With even more RAM, could add a monitor that was responsible automatic job sequencing—batching
      1. Monitor is responsible for loading a program, running it, printing out the results, and then loading the next program.
      2. Resident monitor = that part of the monitor that is always in RAM, i.e. scheduler, interrupt processor, and control
         language interpreter that responds to the job control language (JCL).
      3. Non-resident parts of the monitor were device drivers added to user programs as needed.
      4. With two programs now in residence in RAM we need
         a. Memory protection for the monitor
         b. Timer to return control to the monitor
         c. Privileged instructions that can be executed only by the monitor, e.g. I/O instructions.
         d. Possibly interrupts.
      5. Better than before, but CPU spent a lot of time waiting on I/O.
   E. With a little more RAM, could have multiple programs available to run at once, called multiprogramming.
      1. Memory protection is more complex.
      2. Memory management can be more complex to facilitate having only part of program in memory at any given time.
      3. Need a scheduler.

III. Scheduling
   A. Process = the entity to which a program is assigned that contains the information necessary for proper scheduling.
   B. Types of scheduling
      1. Long-Term Scheduling = decision to add to the pool of processes to be executed.
         a. Can the OS take on another process?
         b. Are there resources to accommodate another process?
      2. Medium-Term Scheduling = decision to add to the number of processes that are partially or fully in RAM.
         a. What degree of multi-programming is acceptable?
b. Is there room in RAM for the new process.
3. I/O Scheduling= decision as to which process’s pending I/O request shall be handled by an available I/O device.
4. Short-Term Scheduling = decision by dispatcher as to which of available processes will be executed by the CPU.
   a. Each process is placed in a process state.
      1) New = waiting to have the process initialized, then moved to ready state.
      2) Ready = process is ready in all ways, and waiting to be executed by the processor.
      3) Running = being executed by the processor
      4) Waiting = the process is suspended form execution because it is waiting for some system resource, e.g. I/O or file.
      5) Halted = the process has terminated, and is waiting for the OS to remove it from RAM, and its process control block from the process table.
5. Process Control Block
   a. Identifier = Unique identifier for each process, process ID = PID
   b. State = current state of the process (new, ready, running, waiting, halted)
   c. Priority
   d. Program counter = address of the next instruction in the program to be executed.
   e. Memory pointers = starting and ending locations of the RAM assigned to the process.
   f. Context data = content of registers when a context switch occurred.
   g. I/O status information = outstanding I/O requests, I/O devices assigned, files assigned, etc.
   h. Accounting information = amount of processor time and clock time used, time limits, account number, etc.
C. Scheduling Techniques
   1. A running process ceases execution for one of three reasons.
      a. It makes a service call to the OS, e.g. a read from the keyboard.
      b. It causes an interrupt. This could be caused by an error, or simply the system timer firing indicating that the process has used up all its allotted time.
      c. An outside event causes an interrupt, e.g. completion of a high-priority I/O operation.
   2. When a running process ceases the processor saves its current context and PC. The OS will then copy this information into its process control block, change its state in the PCB, and assign it to the appropriate queue.
   3. Relies on queues.
      a. Long-term priority queue of processes waiting to use the system.
      b. Short-term (Ready) priority queue of processes waiting for the CPU.
      c. I/O queues waiting for a specific I/O device.