CMPT 300: Operating Systems
Summary Notes

June, 1999

Segment 1: Overview & History

In this segment, we introduced the general topic of operating systems. We examined several views of operating systems, including their role as:

- A Programmer's Toolkit
- An Abstraction Layer
- A Protection Layer
- A Policy Enforcer
- A Control Program
- A Virtual Machine
- A Resource Manager
- A Product

We also examined how computer systems vary and how operating systems must vary with them. Some of the ways in which computer systems differ are

Special-purpose	—	General-purpose
Single-process	—	Multi-process
Single-user	—	Multi-user
Non–resource-sharing	—	Resource-sharing
Single-processor	—	Multi-processor
Stand-alone	—	Networked
Centralized	—	Distributed
Batch	—	Interactive
Deadline-free	—	Real-time
Insecure	—	Secure
Symmetric	—	Asymmetric
Simple	—	Complex
Small	—	Large
Inexpensive	—	Expensive

We also described the history of operating systems. This history shows us *why* operating systems are what they are by showing *how* they developed.

If you understand the evolution of operating systems you should be able to:

- Explain what an operating system is, contrasting the different roles an operating system fulfills.
- Classify computer systems according to a taxonomy.
- Describe how computer systems evolved, explaining what motivated the changes at each step.
- Contrast time-sharing and batch systems.
- Explain why personal computer systems did not start out with "state of the art operating systems".

Segment 2: Computer System Architecture and Operation

This segment covered the general structure of computer systems. We reviewed the basic concepts of machine organization and touched on how code looks at the assembly language level. We briefly examined the concepts of memory, CPU, registers, I/O, interrupts, instructions, and the instruction-execution cycle. Since the operating system is the interface between the hardware and user programs, a thorough understanding of operating systems requires an understanding of both hardware and programs.

We also examined the programmer's view of the operating system, covering system calls and system programs.

If you understand this segment you should be able to:

- Determine which aspects of a computer system operate concurrently.
- Compare and contrast programmed and interrupt-

driven I/O.
- Draw and explain a storage hierarchy.
- Explain hardware protection and design hardware protection schemes.
- Explain the trade-offs associated with caching.
- Explain the trade-offs associated with protection.
- Differentiate between system calls and library calls.
- Identify system programs and shells.
- Differentiate between mechanisms and policies.
- Compare and contrast techniques for structuring operating systems.

Segment 3: Processes & Threads

In this segment we introduced the concept of a process and the notion of concurrent execution. These concepts are at the very heart of modern operating systems. A process is a program in execution and is the unit of work in a modern time-sharing system. Such a system consists of a collection of processes: operating-system processes executing system code, and user processes executing user code. All these processes can potentially execute concurrently, with the processor(s) multiplexed among them. By switching the processor between processes, the operating system can make the computer more productive. We also discussed the notion of a thread (lightweight process) and interprocess communication using message passing and shared memory.

If you understand this segment you should be able to:

- Explain the concept of a process.
- Describe the costs associated with a process switch and a thread switch.
- Explain and contrast the roles of the short-term, medium-term, and long-term schedulers.
- Explain and contrast the terms *time-sharing*, *preemptive multitasking*, *multiprocessing*, and *multiprogramming*.
- Determine the necessary fields for a process-control block.
- Contrast processes and threads.
- Describe the possible scheduling states for a process in both two-state and five-state process models.
- Categorize message-passing systems.
- Explain how to implement one interprocess communications mechanism in terms of another.

Segment 4: Processor Scheduling

Processor scheduling is the basis of multiprogrammed operating systems. By switching the processor among processes, the operating system can make the computer more productive. In this segment, we introduced the basic scheduling concepts and several well-known scheduling algorithms, including FCFS, SBF, Round-Robin and Multilevel Feedback.

If you understand this segment you should be able to:

- Draw Gantt charts to describe scheduling algorithms.
- Describe and contrast a variety of scheduling algorithms.
- State and contrast optimization criteria for processor scheduling.
- Calculate response time (aka turnaround time), missed time (aka waiting time), and penalty ratio for a scheduled burst.

Segment 5: Synchronization

This segment discussed process synchronization among concurrently executing tasks. We discussed the critical-section problem and covered a number of solutions. We also examined a number of "classic" problems in synchronization, including the bounded-buffer problem and the dining-philosophers problem. Synchronization is a difficult topic, and so I strongly recommend students do practice problems beyond those done in class and assignments.

If you understand this segment you should be able to:

- Describe the critical section problem.
- Write parallel programs that do not contain race conditions.
- Determine whether software-based synchronization algorithms operate correctly (or contain race conditions).

- Describe and implement mutexes, semaphores, and monitors.
- Design algorithms using mutexes, semaphores, or monitors for synchronization.
- Describe and implement solutions to several classical synchronization problems.
- Explain and prevent priority inversion.