**Final Operating Systems Syllabus for Aspiring Computer Scientists**

**Course Overview**

This 12-week syllabus is designed for learners with a background in Data Structures and Algorithms (DSA), Software Engineering, and Design Patterns, aiming to master Operating Systems (OS) as a foundation for theoretical computer science. The course integrates core OS concepts, formal methods, distributed systems, and practical implementations, preparing learners for advanced study and research.

**Prerequisites**: Knowledge of DSA, Software Engineering, Design Patterns, and programming in C/C++ or Python. Basic familiarity with computer architecture is recommended (a 1-2 hour primer is included).

**Learning Outcomes**:

* Master theoretical and practical aspects of OS design, including processes, memory, file systems, and security.
* Apply formal methods (e.g., state machines, TLA+) to analyze OS behavior.
* Understand distributed OS concepts and their theoretical challenges.
* Develop skills to implement and verify OS components, preparing for research in theoretical CS.

**Resources**:

* **Textbooks**:
  + *Operating System Concepts* by Silberschatz, Galvin, and Gagne (10th Edition)
  + *Modern Operating Systems* by Andrew S. Tanenbaum (4th Edition)
  + *Operating Systems: Three Easy Pieces* by Arpaci-Dusseau (free online)
* **Tools**: C/C++, Python, Linux (Ubuntu), VirtualBox/VMware, xv6 OS, QEMU, TLA+.
* **Supplementary**: MIT 6.828 (OS Engineering), CMU 15-410 (OS Design), OSDev.org, TLA+ tutorials, SOSP/OSDI papers.

**Pre-Course Primer**: Spend 1-2 hours reviewing CPU, memory hierarchy, and interrupts (*Computer Organization and Design* by Patterson & Hennessy, Chapters 1-2) before Week 1.

**Weekly Syllabus**

**Week 1: Introduction to Operating Systems**

* **Topics**:
  + Role of OS: Abstraction, resource management, virtualization.
  + OS architectures: Monolithic, microkernel, layered, modular.
  + Evolution of OS (UNIX, Windows, Linux).
  + Formal modeling: State machines and process algebras for OS behavior.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 1.
  + Lab: Set up Ubuntu in VirtualBox; explore ps, top, ls.
  + Exercise: Model a resource allocator as a finite state machine.
* **Objective**: Understand OS roles and formal modeling for theoretical analysis.

**Week 2: Processes and Threads**

* **Topics**:
  + Process concept: States, PCB, context switching.
  + Threads: Single vs. multi-threaded, thread models.
  + Inter-process communication: Shared memory, message passing.
  + Theoretical aspects: Process scheduling as graph problems.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 3.
  + Lab: Write a C program using fork() and exec() on Linux.
  + Exercise: Model process state transitions as a state machine.
* **Objective**: Master process/thread management, linking to DSA concepts.

**Week 3: CPU Scheduling**

* **Topics**:
  + Scheduling algorithms: FCFS, SJF, Round-Robin, Priority, Multilevel Queue.
  + Real-time scheduling: Rate Monotonic, Earliest Deadline First (EDF).
  + Theoretical metrics: Throughput, turnaround time, fairness, schedulability.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 5.
  + Lab: Simulate Round-Robin and EDF in Python; analyze metrics.
  + Exercise: Prove schedulability of a task set using Rate Monotonic.
* **Objective**: Master scheduling, including real-time systems, with formal analysis.

**Week 4: Process Synchronization**

* **Topics**:
  + Critical section problem: Mutual exclusion, progress, bounded waiting.
  + Synchronization primitives: Semaphores, monitors, locks.
  + Classical problems: Producer-Consumer, Dining Philosophers.
  + Theoretical aspects: Deadlock, starvation analysis.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 6.
  + Lab: Implement Producer-Consumer using semaphores in C.
  + Exercise: Model Dining Philosophers to analyze deadlock.
* **Objective**: Apply concurrency concepts to solve synchronization problems.

**Week 5: Deadlocks**

* **Topics**:
  + Deadlock conditions: Mutual exclusion, hold-and-wait, no preemption, circular wait.
  + Deadlock prevention, avoidance, detection, recovery.
  + Resource allocation graphs; formal verification with TLA+.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 7.
  + Lab: Implement Banker’s algorithm in Python.
  + Exercise: Model a deadlock scenario in TLA+ for verification.
* **Objective**: Analyze deadlocks using graph theory and formal verification.

**Week 6: Memory Management**

* **Topics**:
  + Memory hierarchy, logical vs. physical addresses.
  + Contiguous allocation: Fragmentation, first-fit, best-fit.
  + Paging: Page tables, TLB, multi-level paging.
  + Theoretical aspects: Memory allocation as optimization.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 8.
  + Lab: Simulate paging and page tables in C.
  + Exercise: Calculate fragmentation for allocation strategies.
* **Objective**: Master memory management and optimization challenges.

**Week 7: Virtual Memory**

* **Topics**:
  + Virtual memory: Demand paging, page faults, copy-on-write.
  + Page replacement: FIFO, LRU, Optimal.
  + Theoretical analysis: Belady’s anomaly, working set model.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 9.
  + Lab: Simulate page replacement algorithms in Python.
  + Exercise: Prove Belady’s anomaly for FIFO.
* **Objective**: Understand virtual memory and its performance implications.

**Week 8: File Systems**

* **Topics**:
  + File system structure: Attributes, operations, organization.
  + File allocation: Contiguous, linked, indexed.
  + Directory structures; journaling and crash consistency.
  + Theoretical aspects: Consistency models.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 11.
  + Lab: Simulate file allocation in C.
  + Exercise: Design a journaling protocol for consistency.
* **Objective**: Understand file system design and theoretical consistency.

**Week 9: I/O Systems and Storage Management**

* **Topics**:
  + I/O hardware: Device controllers, interrupts, DMA.
  + I/O scheduling: FCFS, SSTF, SCAN, C-SCAN.
  + Storage: RAID levels, disk partitioning.
  + Theoretical aspects: I/O optimization, queuing theory.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 12.
  + Lab: Simulate disk scheduling in Python.
  + Exercise: Analyze I/O performance using queuing models.
* **Objective**: Master I/O and storage optimization.

**Week 10: Operating System Security**

* **Topics**:
  + Security principles: Authentication, authorization, access control.
  + Protection mechanisms: Domains, access matrix, capabilities.
  + Formal security models: Bell-LaPadula, Biba.
* **Activities**:
  + Read *Operating System Concepts*, Chapter 14.
  + Lab: Implement a basic access control list in C.
  + Exercise: Model an access control policy using Bell-LaPadula.
* **Objective**: Understand OS security and formal verification.

**Week 11: Case Studies and Modern OS**

* **Topics**:
  + Case studies: Linux, Windows, xv6.
  + Distributed OS: Process migration, distributed scheduling, consensus.
  + Modern trends: Containers, virtualization, microkernel trade-offs.
* **Activities**:
  + Read *Modern Operating Systems*, Chapter 10.
  + Lab: Modify an xv6 system call; simulate distributed scheduling in Python.
  + Exercise: Summarize an OS research paper (SOSP/OSDI).
* **Objective**: Analyze modern and distributed OS, linking to theoretical challenges.

**Week 12: Capstone Project and Review**

* **Topics**:
  + Review: Processes, memory, file systems, scheduling, security.
  + Project: Design an OS component (e.g., scheduler, memory allocator) with formal analysis.
  + Advanced topics: Real-time OS, embedded systems.
* **Activities**:
  + Project: Implement a scheduler/memory allocator in C; verify using TLA+.
  + Document: Include theoretical analysis (e.g., correctness proof).
  + Review: Solve theoretical problems (e.g., scheduling optimization).
* **Objective**: Synthesize OS knowledge with formal theoretical analysis.

**Assessment**

* **Weekly Labs (40%)**: Programming assignments to implement OS concepts.
* **Exercises (30%)**: Theoretical problems linking OS to DSA and formal methods.
* **Capstone Project (20%)**: Design, implement, and verify an OS component with documentation.
* **Participation (10%)**: Engagement in exercises and paper summaries.

**Additional Notes**

* **Design Patterns**: Apply patterns like Singleton (resource management), Observer (I/O events), and Strategy (scheduling) throughout labs.
* **Theoretical CS**: Use formal methods (state machines, TLA+) and graph theory (deadlocks, scheduling) to align with theoretical CS.
* **Practical Tips**: Experiment with Linux system calls, xv6, and QEMU regularly.
* **Research Preparation**: Summarize one OS research paper in Week 11 to build research skills.
* **Further Study**: Explore distributed systems (e.g., Paxos), formal verification (Coq), or real-time OS for advanced research.