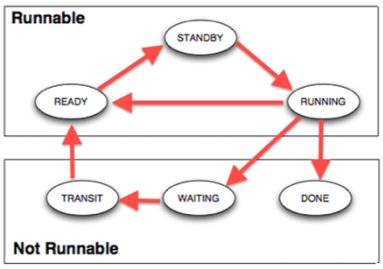
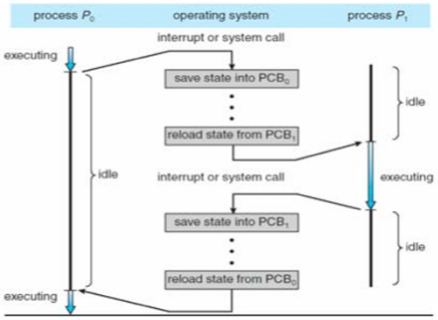
Study Guide for Fall 2025 Operating Systems

Note: This guide (based on Silberschatz) is not a substitute for the lectures and book.

Ch.3 – Processes



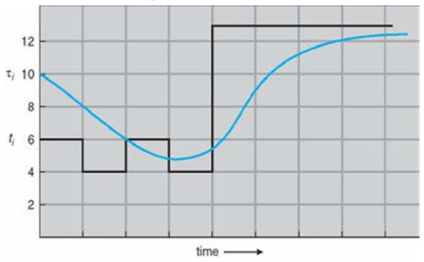
- Process contains a program counter, stack, and data section.
  - Text section: program code itself
  - Stack: temporary data (function parameters, return addresses, local variables)
  - Data section: global variables
  - Heap: contains memory dynamically allocated during run-time
- Process Control Block (PCB): contains information associated with each process: process state, PC, CPU registers, scheduling information, accounting information, I/O status information
- Types of processes:
  - I/O Bound: spends more time doing I/O than computations, many short CPU bursts
  - CPU Bound: spends more time doing computations, few very long CPU bursts
- When CPU switches to another process, the system must save the state of the old process (to PCB) and load the saved state (from PCB) for the new process via a context switch
  - Time of a context switch is dependent on hardware



- Parent processes create children processes (form a tree)
  - PID allows for process management
  - Parents and children can share all/some/none resources
  - Parents can execute concurrently with children or wait until children terminate
  - fork() system call creates new process
    - exec() system call used after a fork to replace the processes' memory space with a new program
- Cooperating processes need interprocess communication (IPC): shared memory or message passing
- Message passing may be blocking or non-blocking
  - Blocking is considered synchronous
    - Blocking send has the sender block until the message is received
    - Blocking receive has the receiver block until a message is available
  - Non-blocking is considered asynchronous
    - Non-blocking send has the sender send the message and continue
    - Non-blocking receive has the receiver receive a valid message or null

Ch.6 – CPU Scheduling

- Process execution consists of a cycle of CPU execution and I/O wait
- CPU scheduling decisions take place when a process:
  - Switches from running to waiting (nonpreemptive)
  - Switches from running to ready (preemptive)
  - Switches from waiting to ready (preemptive)
  - Terminates (nonpreemptive)
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - Dispatch latency- the time it takes for the dispatcher to stop one process and start another
- Scheduling algorithms are chosen based on optimization criteria (ex: throughput, turnaround time, etc.)
  - FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority
- Determining length of next CPU burst: Exponential Averaging:
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. α, $0 \leq \alpha \leq 1$ (commonly α set to 1/2)
  4. Define: $\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n$

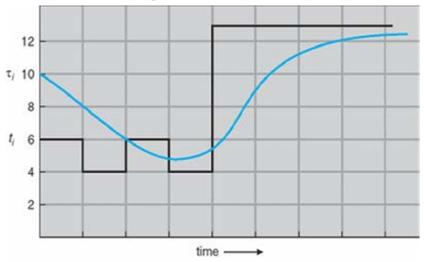


CPU burst (t_i)	6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

- Priority Scheduling can result in starvation, which can be solved by aging a process (as time progresses, increase the priority)
- In Round Robin, small time quantums can result in large amounts of context switches
  - Time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum
- Multilevel Queues and Multilevel Feedback Queues have multiple process queues that have different priority levels
  - In the Feedback queue, priority is not fixed → Processes can be promoted and demoted to different queues
  - Feedback queues can have different scheduling algorithms at different levels

Ch.6 – CPU Scheduling

- Process execution consists of a cycle of CPU execution and I/O wait
- CPU scheduling decisions take place when a process:
  - Switches from running to waiting (nonpreemptive)
  - Switches from running to ready (preemptive)
  - Switches from waiting to ready (preemptive)
  - Terminates (nonpreemptive)
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - Dispatch latency- the time it takes for the dispatcher to stop one process and start another
- Scheduling algorithms are chosen based on optimization criteria (ex: throughput, turnaround time, etc.)
  - FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority
- Determining length of next CPU burst: Exponential Averaging:
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. α, $0 \leq \alpha \leq 1$ (commonly α set to 1/2)
  4. Define: $\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n$

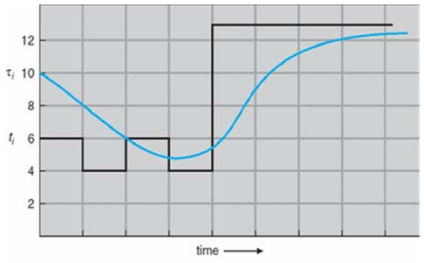


CPU burst (t_i)	6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

- Priority Scheduling can result in starvation, which can be solved by aging a process (as time progresses, increase the priority)
- In Round Robin, small time quantums can result in large amounts of context switches
  - Time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum
- Multilevel Queues and Multilevel Feedback Queues have multiple process queues that have different priority levels
  - In the Feedback queue, priority is not fixed → Processes can be promoted and demoted to different queues
  - Feedback queues can have different scheduling algorithms at different levels

Ch.6 – CPU Scheduling

- Process execution consists of a cycle of CPU execution and I/O wait
- CPU scheduling decisions take place when a process:
  - Switches from running to waiting (nonpreemptive)
  - Switches from running to ready (preemptive)
  - Switches from waiting to ready (preemptive)
  - Terminates (nonpreemptive)
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - Dispatch latency- the time it takes for the dispatcher to stop one process and start another
- Scheduling algorithms are chosen based on optimization criteria (ex: throughput, turnaround time, etc.)
  - FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority
- Determining length of next CPU burst: Exponential Averaging:
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. α, $0 \leq \alpha \leq 1$ (commonly α set to 1/2)
  4. Define: $\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n$

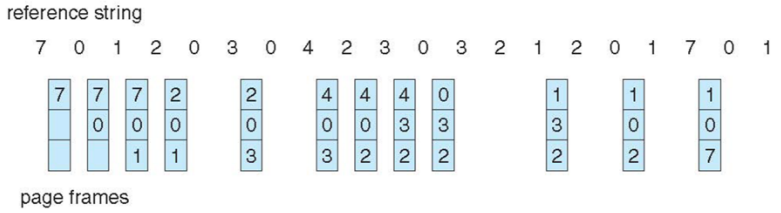


CPU burst (t_i)	6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

- Priority Scheduling can result in starvation, which can be solved by aging a process (as time progresses, increase the priority)
- In Round Robin, small time quantums can result in large amounts of context switches
  - Time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum
- Multilevel Queues and Multilevel Feedback Queues have multiple process queues that have different priority levels
  - In the Feedback queue, priority is not fixed → Processes can be promoted and demoted to different queues
  - Feedback queues can have different scheduling algorithms at different levels

Ch.9 – Virtual Memory

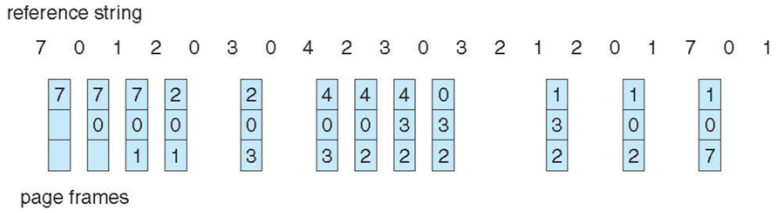
- Virtual memory: separation of user logical memory and physical memory
  - Only part of program needs to be in memory for execution → logical address space > physical address space
  - Allows address spaces to be shared by multiple processes → less swapping
  - Allows pages to be shared during fork(), speeding process creation
- Page fault results from the first time there is a reference to a specific page → traps the OS
  - Must decide to abort if the reference is invalid, or if the desired page is just not in memory yet
    - If the latter: get empty frame, swap page into frame, reset tables to indicate page now in memory, set validation bit, restart instruction that caused the page fault
  - If an instruction accesses multiple pages near each other → less "pain" because of locality of reference
- Demand Paging only brings a page into memory when it is needed → less I/O and memory needed
  - Lazy swapper – never swaps a page into memory unless page will be needed
  - Could result in a lot of page-faults
  - Performance: EAT = [(1-p)*memory access + p*(page fault overhead + swap page out + swap page in + restart overhead)]; where Page Fault Rate 0 ″ p ″ 1
    - if p = 0, no page faults; if p = 1, every reference is a fault
  - Can optimize demand paging by loading entire process image to swap space at process load time
- Pure Demand Paging: process starts with no pages in memory
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- Modify (dirty) bit can be used to reduce overhead of page transfers → only modified pages written to disk
- When a page is replaced, write to disk if it has been marked dirty and swap in desired page
- Pages can be replaced using different algorithms: FIFO, LRU (below)
  - Stack can be used to record the most recent page references (LRU is a "stack" algorithm)



  - Second chance algorithm uses a reference bit
    - If 1, decrement and leave in memory
    - If 0, replace next page
- Fixed page allocation: Proportional allocation – Allocate according to size of process
  - s_i = size of process P_i, $S = \Sigma s_i$, m = total number of frames, a_i – allocation for P_i
  - $a_i = (s_i/S)*m$

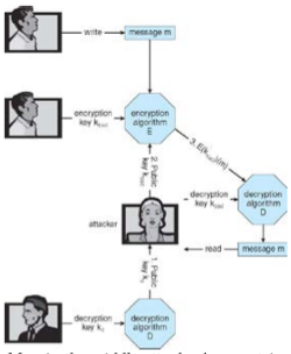
Ch.9 – Virtual Memory

- Virtual memory: separation of user logical memory and physical memory
  - Only part of program needs to be in memory for execution → logical address space > physical address space
  - Allows address spaces to be shared by multiple processes → less swapping
  - Allows pages to be shared during fork(), speeding process creation
- Page fault results from the first time there is a reference to a specific page → traps the OS
  - Must decide to abort if the reference is invalid, or if the desired page is just not in memory yet
    - If the latter: get empty frame, swap page into frame, reset tables to indicate page now in memory, set validation bit, restart instruction that caused the page fault
  - If an instruction accesses multiple pages near each other → less "pain" because of locality of reference
- Demand Paging only brings a page into memory when it is needed → less I/O and memory needed
  - Lazy swapper – never swaps a page into memory unless page will be needed
  - Could result in a lot of page-faults
  - Performance: EAT = [(1-p)*memory access + p*(page fault overhead + swap page out + swap page in + restart overhead)]; where Page Fault Rate 0 ″ p ″ 1
    - if p = 0, no page faults; if p = 1, every reference is a fault
  - Can optimize demand paging by loading entire process image to swap space at process load time
- Pure Demand Paging: process starts with no pages in memory
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- Modify (dirty) bit can be used to reduce overhead of page transfers → only modified pages written to disk
- When a page is replaced, write to disk if it has been marked dirty and swap in desired page
- Pages can be replaced using different algorithms: FIFO, LRU (below)
  - Stack can be used to record the most recent page references (LRU is a "stack" algorithm)



  - Second chance algorithm uses a reference bit
    - If 1, decrement and leave in memory
    - If 0, replace next page
- Fixed page allocation: Proportional allocation – Allocate according to size of process
  - s_i = size of process P_i, $S = \Sigma s_i$, m = total number of frames, a_i – allocation for P_i
  - $a_i = (s_i/S)*m$

Ch.15 – Security



Man-in-the-middle attack - Asymmetric Cryptography

- System secure when resources used and accessed as intended under all circumstances
- Attacks can be accidental or malicious
  - Easier to protect against accidental than malicious misuse
- Security violation categories:
  - Breach of confidentiality – unauthorized reading of data
  - Breach of integrity – unauthorized modification of data
  - Breach of availability – unauthorized destruction of data
  - Theft of service – unauthorized use of resources
  - Denial of service – prevention of legitimate use
- Methods of violation:
  - Masquerading – pretending to be an authorized user
  - Man-in-the-middle – intruder sits in data flow, masquerading as sender to receiver and vice versa
  - Session hijacking – intercept and already established session to bypass authentication
- Effective security must occur at four levels: physical, human, operating system, network
- Program threats: trojan horse (spyware, pop-up, etc.), trap door, logic bomb, stack and buffer overflow
- Viruses: code fragment embedded in legitimate program; self-replicating
  - Specific to CPU architecture, OS, applications
  - Virus dropper: inserts virus onto the system
- Windows is the target for most attacks – most common, everyone is administrator
- Worms: use spawn mechanism – standalone program
- Port scanning: automated attempt to connect to a range of ports on one or a range of IP addresses
  - Frequently launched from zombie systems to decrease traceability
- Denial of service: overload targeted computer preventing it from doing useful work
- Cryptography: means to constrain potential senders and/or receivers – based on keys
  - Allows for confirmation of source, receipt by specified destination, trust relationship
- Encryption: [K of keys], [M of messages], [C of ciphertexts], function E:K to encrypt, function D:K to decrypt
  - Can have symmetric and asymmetric (distributes public encryption key, holds private decipher key) encryption
    - Asymmetric is much more compute intensive – not used for bulk data transaction
    - Keys can be stored on a key ring
- Authentication: constraining a set of potential senders of a message
  - Helps to prove that the message is unmodified
  - Hash functions are basis of authentication
    - Creates small, fixed-size block of data (message digest, hash value)
- Symmetric encryption used in message-authentication code (MAC)
- Authenticators produced from authentication algorithm are digital signatures
- Authentication requires fewer computations than encryption methods
- Digital Certificates: proof of who or what owns a public key
- Defense in depth: most common security theory – multiple layers of security
- Can attempt to detect intrusion:
  - Signature-based: detect “bad patterns”
  - Anomaly detection: spots differences from normal behavior
    - Both can report false positives or false negatives
  - Auditing, accounting, and logging specific system or network activity