Tashfeen PROCESSES 22i-2041 NUCES, ISB

Chapter no. 3 (3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7.4)

- The Processes →
  - Process is a program in execution.
  - Tracked by Program Counter, contents of processor registers.

- Memory Layout Sections →
  - Executable code
  - global variable
  - DMA
  - Temporary data storage. (holds parameters, local variables and addresses)

- Fixed vs Dynamic sizes
  - Fixed ↓ Text (code) data (global variables)
  - Dynamic ↓ stack or heap.

- Stack operations / heap operation.

- Program / Process (Program becomes process when loaded into memory)
  - Program ↓ passive entry stored on disk
  - Process ↓ is active with its own resources

- Java programs run in JVM which is process.
- Command :- Java Program.

- Process Control Block (PCB) → Each process is represented in OS by PCB / task control block.
  - Process state (new, ready, running, waiting)
  - Program Counter (Counter indicates that address of next instruction to be executed for this process)
  - CPU registers (small storage areas. When an interrupt happens, program counter & registers need to be saved so process can continue correctly).
  - CPU sheduling information (priority level of the process)
  - Accounting information (how much CPU time & real time process has used)
  - I/O status information (list of I/O devices that process is using).

Tashi!!

3.2

Process Scheduling

- **Process Scheduling** → • Objectives →
  - • Multiprogramming (Maximize CPU Utilization by having a process running at all times)
  - • Time Sharing (Allow users to interact with programs while they are running by switching CPU cores)
- • Process Execution →
  - • System selects available process for execution on CPU core.
  - • Each CPU core can run one process at a time
  - • ~~In single-core system one process runs at a time~~
- • Process Management →
  - • more processes than available cores, excess processes must wait for a free core.
  - • ~~number of processes currently in memory is called degree of multiprogramming.~~
- • Process Behaviour →
  - • I/O bound (~~spend more time on~~ I/O operation than computation)
  - • CPU-bound (generate I/O request infrequently and focus on computation.

Scheduling queues

① Ready queue → • When processes enter system, they are placed in ready queue, where they wait to execute on CPU core.

↓

implemented as linked list with header pointing to First PCB.

② Waiting queue →
- • When process needs to wait for an event
- • multiple waiting queues can exist

③ Process Flow → new process starts in ready queue → Once allocated a CPU core than →
- • Make I/O request
- • Create child process
- • forcly removed.

④ Cycle →
- • Process can switch b/w states until they terminate.
- • Upon termination they are removed from all queues and their PCB & resources are deallocated.

- CPU Scheduling

Process migration → process moves b/w ready queue and various waiting queue ~~throughout its lifetime.~~

Role of CPU Scheduler →
- Select process from ready queue to allocate CPU core.
- It must select a new process for the CPU.

I/o bound process → It occurs for short period (few milliseconds) before waiting for an I/o request.

CPU-bound process →
- needs CPU for longer duration
- Scheduler does not allow CPU to remain allocated to single process for longer time.

Forced CPU removal →
- Scheduler forcely removes CPU from one process to allow another process to run
- This happens atleast once in every 100 milliseconds

Swapping →
- intermediate form of scheduling where process is removed from memory to reduce multiprogramming
- Later reintroduced into memory to continue its execution
- Swapping out (moving process from memory to disk)
- Swapping in (moving back into memory from disk).

↓

used when memory is overcommited and needs to be free.

- Context Switching (when an interrupt occurs, system must save state of running process and restore state of another process)

↓

System saves CPU resister value

↓

process state, memory for current process and loads for new processes.

•– overhead → context switches are pure overhead b/c system doesn't perform useful work during switch.

↓speed depends upon
hardware architecture,
memory speed,
no. of registers.

3.3 Operations On Processes

•– Process Creation → • Parent & child process → • parent process can create multiple child process
• forms tree like structure
• In UNIX/Linux process is identified by pid.

• Resource Sharing → • Child processes inherit resources from their parent's resource.

• Initilization data → Parents can pass initilization data to their children (e.g files names / resource needed for task completion)

•– Execution Flow → • Concurrent execution → After creating child, parent can either continue executing or wait for child's termination.

• Address Space → Child may duplicate parent's address space or load new program

•– UNIX/Linux Process Creation → • fork() system call creates new process by duplicating parent's address space.
• Parent, child executes same program
• Child process receives return code of 0 while parent gets child's pid.
• exec() system call replaces a processes memory space with new program. allowing child to run different program than parent.

- Windows Process Creation → CreateProcess() function is used for new creating process
  → Immediately loads a program into child's address space.

- Process termination (Process terminates when it finishes its final statement & request deletion via exit() system call)
  - It can return a status value to its parent process using wait() system call.
  - All resources, memory & I/O are reclaimed by OS.

- Termination of other processes → process can terminate another via system calls (TerminateProcess()) usually limited to parent processes.

  ↓ Why termination?

  resource overuse,
  unnecessary tasks,
  parent process termination

- Zombie process → After termination, process may remain in process table as a zombie until its parent calls wait().
  - If parent terminates without calling wait(), init process adopt orphaned child processes

- Resource Management in Mobile OS → Android may terminate processes based
  Foreground (Current process visible on screen)
  Visible background (not visible directly on forehead but performing)
  Service (same to background but is apparent to user).
  Background (Processes not apparent to user.)
  Empty (holds no active components associated with any application).

3.4 •– Interprocess Communication (refers to methods that allow different processes to communicate and share data with each other)

- When multiple processes run at same time on a computer they can be
  - Independent Processes: Don't share any data with other processes
  - Cooperating Processes: Share data and can effect each other.

- Why Cooperation is important?
  - i) Shared information (different applications needs to access same data)
  - ii) Faster tasks (Breaking big tasks into smaller ones allows them to run at same time, which is faster.
  - iii) Modular design (splitting functions into separate processes makes system easier to manage)

Cooperating Processes share data, IPC needed.

•– Shared-Memory Model →
- Processes read/write data to a common memory space.
- This is faster since they don't need to go through OS for every operation.

•– Message-Passing Model →
- Processes sends messages to each other to share information.
- Best for smaller data exchanges and easier to use in distributed system.

3.5

IPC in shared memory system:-

- Producer-Consumer Problem (producer generates data, consumer consumes it)
  e.g., Client-server model (server (producer) resources for clients (consumer))

- Buffering Solutions:
  - Unbounded Buffer (no practical size limit)
    → Consumer may wait for new items, but producer can produce without waiting.
  - Bounded Buffer (fixed size)
    → Consumer waits if buffer is empty
    producer waits if buffer is full.
    → Implementation:- (Implemented as circular array with two logical pointers: in/out.)
    in: points to next free position in buffer.
    out: points to first full position in buffer.
    → Buffer Status:-
    Empty: in == out
    Full: ((in+1) % BUFFER_SIZE) == out.

- Producer & Consumer Processes:-
  1. Producer stores next item to be produced in local variable
  2. Consumer stores next item to be consumed in another local variable.
  3. Maximum of BUFFER_SIZE − 1 items can be in the buffer.

- Synchronization Challenges.
  Both processes have concurrent access to shared buffer must be synchronized.

Beneficial in distributed environments, where processes may be located on different machine connected by a network.

3.6

• – IPC in Message-Passing Systems (process can communicate their action without sharing same memory space)

• Key operations → send / receive.

• Messages type → • Fixed-size messages (• simple system-level implementation, • complecates programming)

• Variable-size messages (• complex implementation, • simplifies programming)

→ Communication links :-

direct / indirect

synchronous / Asynchronous

synchronous → sender waits for receivers

Asynchronous → sender can send message

Automatic / Explicit

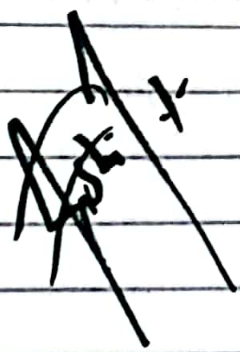
Automatic → System manages buffers for message storage

Explicit → programmer manually handles message buffering.

• – Naming → • Direct Communication → • Both sender & receiver explicitly name each other (symmetric addressing)

e.g., send (P, message)

receive (Q, message)



• Only sender names the recipient (Asymmetric addressing)

e.g., send (P, message)

receive (id, message)

• Indirect Communication → • Utilizes mailboxes or ports for message sending or receiving.

• – properties → • links established only if processes share a mailbox.

• Link may connect more than 2 processes.

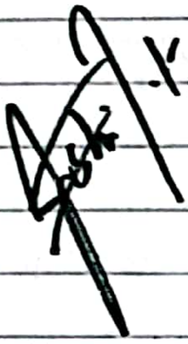
• Message reception scenarios (when multiple processes receive from shared mailbox)

• Mailbox ownership → • Process-Owned mailboxes → only owner can receive messages

• OS-owned mailboxes → independent of any processes.

- Synchronization → • When both send() & receive() are blocking, there's a synchronization b/w sender & receiver.

- Buffering →



- Zero Capacity buffering (no buffering) → • Queue can not hold any messages
  • Sender must wait until recepient is ready to recieve messages.
  • Can lead to blocking due to

Automatic buffering system:

- Finite-Length buffering → • Queue hold limited number of messages
  • If queue not full, sender can place message in queue & continue execution.
  • If queue full, sender must wait until space becomes avalible resulting in blocking

- Infinite-length buffering → • Queue hold unlimited no. of messages.
  • sender can always place messages.

3.7.4

- Pipes (facilitate communication b/w processes)
- Types → Ordinary (unidirectional)
  Named pipes (bidirectional)
- Communication Modes → • Unidirectional → one way communication (e.g Producer & Consumer)
  • Bidirectional → Named pipes allow two-way communication without parent-child relationship.

Window → support both byte & message-oriented data

UNIX/FIFO → support byte-oriented data.

- Ordinary Pipes → • UNIX → Child processes inherit pipe from their parent.
  (temporary & exist only during Communication)
  • Windows → Created with CreatePipe().
- Named Pipe → • UNIX → known as FIFO / Created by mkfifo()
  (half-duplex communication) exist until explicitly deleted.
  • Windows → used for inter-machine Communication
  (full-duplex communication)

Limitation (Ordinary & named pipes requires processes to be on same machine

Chapter 3 finished.

Init process → • first process that starts on a system.
PID = 1

- All other processes running on system will be direct/indirect ~~children of the process.~~
- ~~Drawback → starts fork serially~~
  1-
  2- ~~long delay in~~
  3- boot process.
- Alternatives → ① systemd] do parallelly.
  ② upstart]
- fork() , CreateProcess()
  ↓ Linux ↓ calls ↓ Windows

clone() → system call

Execution:-

1) Parent continues to executes concurrently.
2) Parent waits until its child has completed.

Memory:-

1) Child process is duplicate of parent
2) new program loaded in child.

pid > 0 (parent)
pid < 0 (error)
pid == 0 (child)