Embedded Operating Systems

OS Concepts

- Introduction
- File & IO management
- Process management
- Memory management
- CPU scheduling

Linux Programming

- Commands
- Shell scripts
- System call programming

Embedded Linux

- Linux porting

Evaluation

- Theory -- 40 marks -- CCEE
- Lab -- 40 marks -- Linux programming
- Internals -- 20 marks

Books

- OS Concepts -- Galvin (Chapter 1 & 2)
- OS Design -- Crowley
- Design of UNIX OS -- Bach (File IO, IPC)
- Professional Linux Kernel Architecture (Linux internals -- VFS, Task management, ...)
- Linux kernel development -- Love (Linux internals -- Memory management)
- Beginning Linux Programming -- Neil (Commands and Shell Scripts, SysCall Basics)
- Linux Programming Interface -- Kerrisk (Linux SysCall Programming)

Learning OS

- step 1: End user
  - Linux commands
- step 2: Administrator
  - Install OS (Linux)
  - Configuration - Users, Networking, Storage, ...
  - Shell scripts

- step 3: Programmer
    - Linux System call programming
- step 4: Designer/Internals
    - UNIX & Linux internals

What is OS?

- Interface between end user and computer hardware.
- Interface between Programs and computer hardware.
- Control program that controls execution of all other programs.
- Resource manager/allocator that manage all hardware resources.
- Bootable CD/DVD = Core OS + Applications + Utilities
- Core OS = Kernel -- Performs all basic functions of OS.

OS Functions

- CPU scheduling
- Process Management
- Memory Management
- File & IO Management
- Hardware abstraction
- User interfacing
- Security & Protection
- Networking

Process Management

Program

- Set of instructions given to the computer --> Executable file.
- Program --> Sectioned binary --> "objdump" & "readelf".
    - Exe header --> Magic number, Address of entry-point function, Information about all sections. (objdump -h program.out)
    - Text --> Machine level code (objdump -S program.out)
    - Data --> Global and Static variables (Initialized)
    - BSS --> Global and Static variables (Uninitialized)
    - RoData --> String constants
    - Symbol Table --> Information about the symbols (Name, Size, section, Flags, Address) (objdump -t program.out)
- Program (Executable File) Format
    - Windows -- PE
    - Linux -- ELF
- Program are stored on disk (storage).

Process

- Progam under execution
- Process execute in RAM.
- Process control block contains information about the process (required for the execution of process).
    - Process id
    - Exit status
        - 0 - Indicate successful execution
        - Non-zero - Indicate failure
    - Scheduling information (State, Priority, Sched algorithm, Time, ...)
    - Memory information (Base & Limit, Segment table, or Page table)
    - File information (Open files, Current directory, ...)
    - IPC information (Signals, ...)
    - Execution context (Values of CPU registers)
    - Kernel stack
- PCB is also called as process descriptor (PD), uarea (UNIX), or task_struct (Linux).
- In Linux size of task_struct is approx 4KB

Process Life Cycle

Process States

- New
    - New process PCB is created and added into job queue. PCB is initialized and process get ready for execution.
- Ready
    - The ready process is added into the ready queue. Scheduler pick a process for scheduling from ready queue and dispatch it on CPU.
- Running
    - The process runs on CPU. If process keeps running on CPU, the timer interrupt is used to forcibly put it into ready state and allocate CPU time to other process.
- Waiting
    - If running process request for IO device, the process waits for completion of the IO. The waiting state is also called as sleeping or blocked state.
- Terminated
    - If running process exits, it is terminated.

OS Data Structures:

- Job queue / Process table: PCBs of all processes in the system are maintained here.
- Ready queue: PCBs of all processes ready for the CPU execution and kept here.
- Waiting queue: Each IO device is associated with its waiting queue and processes waiting for that IO device will be kept in that queue