Comprehensive Linux Systems Programming Course Outline

This course offers a thorough exploration into the world of Linux systems programming and kernel architecture. Each module is designed to provide detailed insights into key areas, ensuring a structured learning experience for advanced learners. Below is a breakdown of each section along with their objectives and key topics:

1. **Introduction to Assembly Language**
   - **Objective**: Demonstrate the portability of the UNIX/Linux interface through the use of assembly system calls in Glibc.
   - **Key Topics**:
     - Basics of assembly language and its importance in low-level programming.
     - How system calls are implemented in assembly within the Glibc environment.
     - Calling conventions across different processors (e.g., x86, ARM).
2. **Tutorial on POSIX Threads**
   - **Objective**: Explore the Native POSIX Thread Library (NPTL) and relevant kernel implementations.
   - **Key Topics**:
     - Overview of POSIX threads (pthreads) and thread management.
     - Detailed walkthrough of thread creation, synchronization, and management in Linux.
     - Examination of kernel-level support for threading.
3. **Signals as a Means of Interprocess Communications**
   - **Objective**: Discuss the role of signals in IPC and related system concepts.
   - **Key Topics**:
     - Fundamentals of UNIX signals and their role in controlling processes.
     - Process states, Job Control, and pseudo terminals (PTYs).
     - Use cases and limitations of signals in modern IPC.
4. **Introduction to Kernel Modules**
   - **Objective**: Explore kernel modules and pseudo file systems crucial for system monitoring.
   - **Key Topics**:
     - Concepts and creation of Linux kernel modules.
     - Deep dive into /proc, /sys filesystems and their role in the Linux ecosystem.
     - Understanding Cgroup filesystem and its usage in resource management.
5. **Building Embedded Linux File Systems and Kernel using Yocto Build System**

- **Objective**: Learn to build customized Linux environments using different build systems.
- **Key Topics**:
  - Introduction to the Yocto project and its architecture.
  - Building a minimal Linux system for embedded devices.
  - Comparison with Debian-like file systems built using Bootstrap and Multistrap.

6. **Kernel Namespaces and Container Software**
   - **Objective**: Examine how Linux namespaces support containerization technologies.
   - **Key Topics**:
     - Overview of Linux namespaces (PID, net, IPC, mount, etc.).
     - Practical demonstrations using Docker, Podman, and Systemd-nspawn.
     - System calls associated with namespaces management.
7. **Fundamentals of TCP/IP and Socket Programming**
   - **Objective**: Detailed study of network programming and the Linux network stack.
   - **Key Topics**:
     - TCP/IP fundamentals and socket programming concepts.
     - Exploring the Linux kernel network stack.
     - Hands-on coding examples in creating network applications.

Fridays

- Special sessions dedicated to fundamental computer science concepts necessary for understanding deeper system-level functionalities.
- Topics include implementation of classes and objects in non-object-oriented languages like C, with parallels drawn in C++, Python, and JavaScript.
- Additional fundamental topics necessary for a rounded understanding of system programming.

Each topic is paired with practical labs and real-world examples to enhance learning and retention, making it ideal for those aiming to deepen their understanding of Linux internals and system programming.