Based on my research, here's a comprehensive **7-day study plan** with direct links to all the resources you need:

🐧 Complete 7-Day Linux Kernel Interview Preparation Plan

Overview

Day	Focus Area	Hours
1	Kernel Architecture & Setup	4-6 hrs
2	Process Management & Scheduling	4-6 hrs
3	Memory Management	4-6 hrs
4	System Calls	4-5 hrs
5	Synchronization & Concurrency	4-6 hrs
6	Kernel Modules & Device Drivers	4-6 hrs
7	Review & Mock Interview Practice	4-6 hrs

📅 Day 1: Kernel Architecture & Development Setup

🎯 Learning Objectives

- Understand what the Linux kernel is and its architecture
- Set up a development environment
- Learn the difference between user space and kernel space
- Build your first kernel from source

📖 Reading Materials

Resource	Link	Time
HOWTO do Linux kernel development (Official)	kernel.org/doc/html/latest/process/howto.html	45 min

Resource	Link	Time
Linux Kernel Architecture Overview	kernel.org/doc/html/latest/	1 hr
A Beginner's Guide to Linux Kernel Development (LFD103)	training.linuxfoundation.org/training/a-beginners-guide-to-linux-kernel-development-lfd103/	2 hrs
Kernel Newbies - Getting Started	kernelnewbies.org/KernelBuild	30 min

🛠️ Hands-On Tasks

1. **Set up your development environment:**

```bash
# Install required packages (Ubuntu/Debian)
sudo apt update
sudo apt install build-essential libncurses-dev bison flex libssl-dev libel
f-dev

# Clone the kernel source
git clone --depth=1 https://git.kernel.org/pub/scm/linux/kernel/git/torvald
s/linux.git
cd linux
```

2. **Configure and build the kernel:**

```bash
# Copy current config
cp /boot/config-$(uname -r) .config
make olddefconfig

# Build (use -j$(nproc) for parallel compilation)
make -j$(nproc)
```

📝 Key Concepts to Master

- **Monolithic kernel** vs Microkernel
- **User space** vs **Kernel space**
- **Kernel rings** (Ring 0 vs Ring 3)
- Role of the **bootloader** (GRUB)

📅 Day 2: Process Management & Scheduling

🎯 Learning Objectives

- Understand `task_struct` and process descriptors
- Learn process states and lifecycle
- Master the CFS and EEVDF schedulers
- Understand context switching

📖 Reading Materials

Resource	**Link**	**Time**
Process Scheduling in Linux	scaler.com/topics/operating-system/process-scheduling-in-linux/	45 min
EEVDF Scheduler Documentation (Official)	kernel.org/doc/html/latest/scheduler/sched-eevdf.html	30 min
CFS Scheduler Documentation	kernel.org/doc/html/latest/scheduler/sched-design-CFS.html	45 min
Linux Inside: Process Scheduler	0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-4.html	1 hr
Kernel Source: task_struct	elixir.bootlin.com/linux/latest/source/include/linux/sched.h	30 min

📝 Key Concepts to Master

Process States

State	Code	Description
Running	R	Currently executing or in run queue
Interruptible Sleep	S	Waiting for event, can be interrupted
Uninterruptible Sleep	D	Waiting for I/O, cannot be interrupted
Stopped	T	Stopped by signal (SIGSTOP)
Zombie	Z	Terminated but not reaped by parent

Scheduler Evolution

```
O(1) Scheduler → CFS (2.6.23) → EEVDF (6.6+)
```

🛠️ Hands-On Tasks

```bash
# Observe process states
ps aux | head -20

# Watch scheduler in action
watch -n 1 'cat /proc/loadavg'

# View scheduling info for a process
cat /proc/self/sched

# Check nice values
nice -n 10 sleep 100 &
ps -l
```

📅 Day 3: Memory Management

🎯 Learning Objectives

- Understand virtual memory and paging
- Learn about page tables (multi-level)
- Master the Buddy Allocator and SLUB allocator
- Understand `kmalloc()` vs `vmalloc()`

📖 Reading Materials

Resource	Link	Time
Memory Management Documentation (Official)	kernel.org/doc/html/latest/admin-guide/mm/index.html	1 hr
Linux Inside: Memory Management	0xax.gitbooks.io/linux-insides/content/MM/	1.5 hrs
SLUB Allocator	kernel.org/doc/html/latest/mm/slub.html	30 min
Understanding Linux Virtual Memory Manager	kernel.org/doc/gorman/	1 hr
Multi-Gen LRU (MGLRU)	kernel.org/doc/html/latest/mm/multigen_lru.html	30 min

📝 Key Concepts to Master

Memory Hierarchy

```
┌─────────────────────────────────────────┐
│              Virtual Address Space              │
├─────────────────────────────────────────┤
│  User Space (Lower addresses)                  │
├─────────────────────────────────────────┤
│  Kernel Space (Higher addresses)               │
│    ├── Direct Mapping (Linear Map)             │
│    ├── vmalloc Area                            │
│     └── Kernel Modules                               │
└──────────────────────────────────────────────────────┘
```

Memory Zones

Zone	Purpose
`ZONE_DMA`	Legacy devices (first 16MB)
`ZONE_NORMAL`	Directly mapped kernel memory
`ZONE_HIGHMEM`	Memory above 896MB (32-bit only)
`ZONE_MOVABLE`	Migratable pages for hot-plug

kmalloc vs vmalloc

Feature	`kmalloc()`	`vmalloc()`
Physical Memory	Contiguous	Non-contiguous
Virtual Memory	Contiguous	Contiguous
Speed	Faster	Slower
Use Case	Small allocations	Large allocations

🛠️ Hands-On Tasks

```bash
# View memory info
cat /proc/meminfo

# Monitor memory in real-time
vmstat 1

# View slab allocator statistics
sudo slabtop

# Check page size
```

```
getconf PAGESIZE

# View memory zones
cat /proc/buddyinfo
```

🗓️ Day 4: System Calls

🎯 Learning Objectives

- Understand how system calls work
- Learn the syscall entry/exit mechanism
- Trace system calls with `strace`
- Understand the syscall table

📖 Reading Materials

Resource	Link	Time
System Calls Documentation (Official)	kernel.org/doc/html/latest/process/adding-syscalls.html	45 min
Linux Inside: System Calls	0xax.gitbooks.io/linux-insides/content/SysCall/	1.5 hrs
The Definitive Guide to Linux System Calls	packagecloud.io/blog/the-definitive-guide-to-linux-system-calls/	1 hr
Searchable Linux Syscall Table (x86_64)	filippo.io/linux-syscall-table/	Reference
Syscall Man Pages	man7.org/linux/man-pages/man2/syscalls.2.html	Reference

📝 Key Concepts to Master

System Call Flow (x86_64)

```
User Space                    Kernel Space
    │                              │
    ├── syscall instruction ──────►│
```

```
        │   (saves RIP, loads          │
        │    kernel entry point)       │
        │                              ├── entry_SYSCALL_64
        │                              │
        │                              ├── Look up syscall table
        │                              │
        │                              ├── Execute sys_xxx()
        │                              │
        │◄── sysret instruction ───────┤
        │   (restore user context)     │
```

Important System Calls to Know

Syscall	Purpose
`fork()` / `clone()`	Create new process/thread
`exec()`	Replace process image
`open()` / `read()` / `write()`	File I/O
`mmap()`	Memory mapping
`ioctl()`	Device control
`socket()`	Network communication

🛠️ Hands-On Tasks

```bash
# Trace system calls of a command
strace ls -la

# Count syscalls
strace -c ls -la

# Trace specific syscalls
strace -e open,read,write cat /etc/passwd
```

```
# View syscall numbers
ausyscall --dump
```

🗓️ Day 5: Synchronization & Concurrency

🎯 Learning Objectives

- Master spinlocks, mutexes, and semaphores
- Understand when to use each primitive
- Learn about RCU (Read-Copy-Update)
- Understand deadlock prevention

📖 Reading Materials

Resource	Link	Time
Unreliable Guide to Locking (Official)	kernel.org/doc/html/latest/kernel-hacking/locking.html	1 hr
Linux Inside: Synchronization	0xax.gitbooks.io/linux-insides/content/SyncPrim/	1.5 hrs
RCU Documentation	kernel.org/doc/html/latest/RCU/whatisRCU.html	45 min
Mutex API Reference	kernel.org/doc/html/latest/locking/mutex-design.html	30 min
Completions API	kernel.org/doc/html/latest/scheduler/completion.html	20 min

📝 Key Concepts to Master

Comparison Table

Feature	Spinlock	Mutex	Semaphore
Sleeping	❌ No	✅ Yes	✅ Yes
Interrupt Context	✅ Yes	❌ No	❌ No

Feature	Spinlock	Mutex	Semaphore
Ownership	No	Yes	No
Count	Binary	Binary	Counting
Overhead	Low	Medium	Medium

When to Use What

```
┌─────────────────────────────────────────┐
│ Can I sleep?                              │
│   ├── NO  → Use Spinlock                 │
│   └── YES → Is critical section short?   │
│              ├── YES → Use Spinlock (maybe)  │
│              └── NO  → Use Mutex         │
└─────────────────────────────────────────┘
```

Spinlock Variants

```c
spin_lock()              // Basic, disables preemption
spin_lock_irq()          // Also disables local interrupts
spin_lock_irqsave()      // Saves and restores IRQ state (SAFEST)
spin_lock_bh()           // Disables bottom halves
```

🛠️ Hands-On Tasks

```bash
# Check for lock contention
sudo perf lock record -a sleep 5
sudo perf lock report
```

```
# View lock statistics (if enabled)
cat /proc/lock_stat

# Monitor for deadlocks
echo 1 | sudo tee /proc/sys/kernel/hung_task_warnings
```

📅 Day 6: Kernel Modules & Device Drivers

🎯 Learning Objectives

- Write and load kernel modules
- Understand the device model
- Learn about character and block devices
- Understand `/dev` and `sysfs`

📖 Reading Materials

Resource	Link	Time
Linux Kernel Module Programming Guide (LKMPG) ⭐	sysprog21.github.io/lkmpg/	2 hrs
LKMPG GitHub (Latest Source)	github.com/sysprog21/lkmpg	Reference
Linux Device Drivers, 3rd Edition (Free)	lwn.net/Kernel/LDD3/	1 hr (Ch 1-2)
Driver Model Documentation	kernel.org/doc/html/latest/driver-api/driver-model/	30 min
Kernel Newbies: First Patch Tutorial	kernelnewbies.org/FirstKernelPatch	30 min

🛠️ Hands-On: Write Your First Kernel Module

1. Create `hello.c`:

```
c
```

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("Hello World Kernel Module");
MODULE_VERSION("1.0");

static int __init hello_init(void)
{
    printk(KERN_INFO "Hello, Kernel World!\n");
    return 0;  // 0 = success
}

static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye, Kernel World!\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

2. Create `Makefile`:

makefile

```makefile
obj-m += hello.o

KDIR := /lib/modules/$(shell uname -r)/build

all:
	make -C $(KDIR) M=$(PWD) modules

clean:
	make -C $(KDIR) M=$(PWD) clean
```

3. Build and test:

bash

```bash
# Install headers
sudo apt install linux-headers-$(uname -r)

# Build
make

# Load module
sudo insmod hello.ko

# Check kernel log
dmesg | tail -5

# List loaded modules
lsmod | grep hello

# Unload module
sudo rmmod hello

# Check log again
dmesg | tail -5
```

📝 Key Concepts to Master

- `module_init()` and `module_exit()` macros
- `printk()` log levels (KERN_INFO, KERN_ERR, etc.)
- Module parameters with `module_param()`
- Character device registration with `register_chrdev()`

📅 Day 7: Review & Mock Interview Practice

🎯 Learning Objectives

- Review all key concepts

- Practice explaining concepts out loud
- Work through common interview questions
- Identify and fill knowledge gaps

📖 Review Resources

Resource	Link	Time
Top Linux Kernel Interview Questions	adaface.com/blog/kernel-interview-questions/	1 hr
Linux Kernel Coding Style	kernel.org/doc/html/latest/process/coding-style.html	30 min
Kernel Source Browser (Elixir)	elixir.bootlin.com/linux/latest/source	Reference

💡 Top 20 Interview Questions to Practice

Process Management

1. **What is `task_struct` ?**
2. **Explain fork(), exec(), and wait() system calls.**
3. **What is a zombie process? How do you prevent them?**
4. **Explain the difference between CFS and EEVDF schedulers.**
5. **What triggers a context switch?**

Memory Management

6. **How does virtual memory work?**
7. **What is a page fault? Types of page faults?**
8. **Difference between `kmalloc()` and `vmalloc()` ?**
9. **What is the Buddy Allocator?**
10. **Explain the SLUB allocator.**

Synchronization

11. **When do you use spinlock vs mutex?**
12. **Can you sleep while holding a spinlock? Why not?**
13. **What is RCU and when is it used?**

14. **Explain the difference between `spin_lock()` and `spin_lock_irqsave()`.**
15. **What is a deadlock? How do you prevent it?**

System Calls & Modules

16. **What happens when you call `read()` from user space?**
17. **How do you add a new system call to the kernel?**
18. **What is the difference between `insmod` and `modprobe`?**
19. **What does `MODULE_LICENSE("GPL")` do?**
20. **How do interrupts work? Top-half vs bottom-half?**

🎤 Mock Interview Exercise

Practice answering these scenario-based questions:

1. **"Walk me through what happens when you type `ls`** in a terminal."
   - *Cover: shell parsing, fork(), exec(), syscalls, file system, output*
2. **"A process is stuck in 'D' state. How do you debug it?"**
   - *Cover: uninterruptible sleep, I/O wait,* `cat /proc/<pid>/stack`
3. **"How would you debug a kernel panic?"**
   - *Cover: dmesg, crash dump, addr2line, stack trace analysis*

✅ Final Checklist

Before your interview, make sure you can:

- [ ] Explain kernel architecture (monolithic, user/kernel space)
- [ ] Describe process states and transitions
- [ ] Explain virtual memory and paging
- [ ] Compare spinlock, mutex, and semaphore
- [ ] Write and explain a basic kernel module
- [ ] Trace a system call from user space to kernel
- [ ] Use debugging tools: `dmesg`, `strace`, `ftrace`, `perf`

📚 Quick Reference: All Links in One Place

Official Documentation

Topic	URL
Kernel Documentation Portal	https://www.kernel.org/doc/html/latest/
HOWTO Kernel Development	https://www.kernel.org/doc/html/latest/process/howto.html
Scheduler Documentation	https://www.kernel.org/doc/html/latest/scheduler/
Memory Management	https://www.kernel.org/doc/html/latest/mm/
Locking Guide	https://www.kernel.org/doc/html/latest/kernel-hacking/locking.html

Learning Resources

Resource	URL
Linux Kernel Module Programming Guide	https://sysprog21.github.io/lkmpg/
Linux Inside (Deep Dive)	https://0xax.gitbooks.io/linux-insides/
Kernel Newbies	https://kernelnewbies.org/
LFD103 (Free Course)	https://training.linuxfoundation.org/training/a-beginners-guide-to-linux-kernel-development-lfd103/
Linux Device Drivers (Book)	https://lwn.net/Kernel/LDD3/

Tools & References

Tool	URL
Kernel Source Browser	https://elixir.bootlin.com/linux/latest/source
Syscall Table	https://filippo.io/linux-syscall-table/
Man Pages	https://man7.org/linux/man-pages/

🚀 Final Tips for Success

1. **Focus on understanding, not memorization** Interviewers want to see you can reason through problems
2. **Practice explaining concepts out loud** Record yourself or explain to a friend
3. **Get hands-on experience** Even a simple module shows initiative
4. **Know your debugging tools** `dmesg`, `strace`, `gdb`, `perf`
5. **Be honest about what you don't know** It's okay to say "I'm not sure, but I would approach it by..."