你上传的这些 PDF 是 RWTH Aachen（亚琛工大）《嵌入式系统导论》课程的不同章节讲义。我给你逐章总结一下：

第 1 章 – 微控制器 (Microcontrollers)

- 微控制器 vs 微处理器：微控制器是低成本、带存储器和外设的独立设备，专门用于嵌入式应用。
- 基本组成：处理器核心、SRAM、Flash/EEPROM、定时器/计数器、看门狗、数字/模拟 I/O、总线控制器。
- 数字 I/O：通过寄存器 (DDR, PORT, PIN) 控制 LED、按钮等硬件输入输出。
- 中断：代替轮询，响应外部或内部事件，使用中断向量表和 ISR。
- 定时器与计数器：实现延时、频率测量、PWM。
- 模拟 I/O：A/D 转换器与 D/A 转换器。

第 2 章 – 数据总线 (Data Buses)

- 通信原理：嵌入式系统组件之间通过总线互连。
- 拓扑结构：总线型、星型、环型（成本/可靠性/实时性不同）。
- ISO/OSI 模型：重点是物理层和数据链路层。
- 物理层：介质（铜线、光纤、无线）、电气特性、比特编码方式（NRZ、曼彻斯特、4B/5B）。
- 数据链路层：帧结构、错误检测（奇偶校验、CRC、海明码）、自动重传 (ARQ)、介质访问控制 (MAC)。
- 典型总线协议：I²C、SPI、CAN、FlexRay、Ethernet。

第 3 章 – 可编程逻辑控制器 PLC (Programmable Logic Controllers)

- 控制类型：
    - 逻辑控制（急停、互锁、顺序控制）。
    - 连续控制（PID 控制）。

- PLC 技术：结构、循环扫描执行方式（读输入→执行逻辑→写输出），反应时间约等于两倍扫描周期。
- IEC 61131 标准化语言：
    - 功能块图 (FBD)、梯形图 (LD)、指令表 (IL)、结构化文本 (ST)、顺序功能图 (SFC)。
- 应用：工业自动化、分布式控制系统 (DCS)、软 PLC。

第 4 章 – 实时系统 (Real-Time Systems)

- 实时要求：不仅正确，还要在截止时间内完成。
    - 硬实时：必须在截止时间前完成。
    - 软实时：大部分情况下按时完成即可。
- 实时操作系统 (RTOS) – 示例 OSEK：任务模型、调度（抢占/非抢占）、优先级队列。
- 常见问题：
    - 死锁。
    - 优先级反转（例如 NASA 火星探路者号曾经遇到过）。
- 解决方案：优先级继承、优先级天花板协议。

第 5 章 – 嵌入式软件开发与设计 (Embedded Software Development & Design)

- 开发过程模型：瀑布模型、螺旋模型、V 模型（最常见）。
- 需求分析：功能需求 vs 质量需求（如安全性、可靠性、可维护性）。
- 架构设计：如 AUTOSAR、CAN/FlexRay 总线架构。
- 案例：自适应巡航控制 (ACC) 系统。
- 要点：
    - 区分需求和约束。
    - 需求必须可验证、可理解。
    - 功能需求（做什么） vs 非功能需求（做得多好）。

第 6 章 – 模型驱动开发与 Simulink (Model-Based Development & Simulink)

- MATLAB & Simulink：用图形化方式建模动态系统。
- 信号建模：采样、数据类型、维度等。
- 层次化建模：子系统、信号总线。
- 高级扩展：M 函数、S 函数、Stateflow（状态机）。
- 模型驱动软件开发流程：
    - 需求分析 → 架构设计 → 建模 → 自动生成代码。
- 自动代码生成工具：
    - Embedded Coder（C/C++）。
    - PLC Coder（梯形图、功能块）。
    - TargetLink（C，需特定库）。

📌 **总结**：
这些文档构成了一个系统化的《嵌入式系统导论》课程，内容覆盖 **硬件（微控制器、总线、PLC）、操作系统（实时系统）、软件工程（开发流程、需求分析）、模型驱动开发（Simulink）** 等。对学习嵌入式系统开发的整个链条（从硬件到软件方法论）都有完整覆盖。

要不要我帮你把这些章节之间的联系画成一张 **知识结构图/思维导图**，方便你快速掌握整体框架？

你这份 PDF 是 **亚琛工大（RWTH Aachen）“Fundamentals of Computer Science 3: Operating Systems and System Security”** 课程的讲义。我帮你总结一下主要内容：

课程背景

- 课程由 **Stefan Lankes 博士** 和 ACS 团队授课。
- 重点在 **操作系统原理** 和 **系统安全**，并结合现代编程语言（C、Rust、汇编）。

核心内容

1. 编程语言与系统编程

- 讨论了 **低级语言**（C、Rust、Assembly）的特点。
- 为什么要引入 **Rust**：内存安全（避免空指针、悬垂指针），但依然有零开销抽象。
- 汇编的重要性：直接访问硬件、利用新指令扩展。

2. 操作系统基础

- 操作系统的角色：
  - **裁判**：资源管理（公平性、效率、实时性）。
  - **魔术师**：提供虚拟化和抽象（文件系统、虚拟机）。
  - **胶水**：连接应用和硬件（API、ABI、系统库）。
- 历史：从 **Unix、C语言、GNU/Linux** 的发展讲起。
- 不同操作系统在 **服务器、物联网、实时计算、高性能计算、云计算** 等领域的应用。

3. UNIX 与 Shell

- Unix 哲学："一切皆文件"。
- Shell 基础：CLI 的优势、常用命令（ls, cd, cp, rm, man, echo 等）。
- I/O 重定向与管道、退出码、环境变量、循环与条件语句、Shell 脚本编写。

4. 软件开发与版本控制

- Makefile 与依赖管理。
- Git：版本控制原理、本地仓库与远程仓库（GitHub/GitLab/RWTH Git）。
- 大型项目协作的复杂性（Linux 内核的例子：每天上万行代码变更）。

5. 计算机体系结构

- 冯·诺依曼结构：CPU、内存、I/O 的交互。
- 缓存与局部性原理（时间局部性、空间局部性）。
- 流水线、分支预测、流水线冲突（数据、控制、结构性 hazard）。
- 单核 vs 多核，RISC vs CISC。

6. 汇编编程

- 内存布局（代码段、数据段、栈、堆）。
- 基本语法（寄存器、数据类型、指令示例）。
- 栈帧与函数调用约定（C calling convention，参数传递、寄存器保存/恢复）。

总结

这份讲义是 **操作系统+计算机体系结构+低级编程（C/Rust/汇编）+软件工程工具（Shell、Make、Git）** 的综合课程材料。
它的目标是：**带你从底层理解操作系统，并最终能在自己写的内核上运行 Linux 应用。**

要不要我帮你把这份 PDF 的 **知识点体系图**（类似思维导图）画出来？这样你能快速看到“编程语言 → 操作系统角色 → UNIX & Shell → 开发工具 → 体系结构 → 汇编”之间的关系。