第一部分：

userinit/proc.c -> allocproc/proc.c

kalloc/proc.c

mpmain/main.c->scheduler/proc.c->swtch.S->forkret/proc.c-> trapret/trapasm.S

重点关注问题：

- 进程相关的数据结构，如 struct proc，其中重要字段的作用，如 pid, state, tf 等。
- 创建第一个进程时，内核栈的布局—trapframe 和 context 的结构
- c 代码在调用汇编代码时，汇编代码是如何获取到相应参数的？提示：calling convention。
- swtch 将控制转移到了什么地方？是如何实现的？
- forkret 又将控制转移到了什么地方？是如何实现的？
- trapret 中 iret 指令的作用？为什么是 iret 而不是 ret？返回地址是什么？

进阶题：

Linux 中进程相关的数据结构在哪里？相比 xv6 的 PCB 增加了哪些结构？请选取其中 1 ~ 2 项结合代码详细分析。

第二部分：

initcode.S->alltraps/trapasm.S->vector64/vectors.S(注意！该文件源码中可能没有，它是通过 vectors.pl 文件生成的)->trap/trap.c->syscall/syscall.c->trapret/trapasm.S

重点关注的问题：

- initcode.S 这段用户态代码是如何向内核传递参数的？
- initcode.S 代码里，int 指令的作用？
- int 指令执行完后，栈指针 esp 指向的用户栈还是内核栈？内核栈发生了什么变化？int 指令执行完后，控制转移到了什么地方？为什么？
- 执行系统调用时，trapframe 中的 trapno 的值时多少？在什么地方压栈的？
- 汇编代码在调用 c 代码时，是如何向 c 传递参数的？以 trap 函数的 tf 参数为例子。
- 系统调用的返回值是如何从内核态返回到用户态的？
- initcode.S 执行的是哪个 syscall？是通过什么方式告知内核的？
- trap 函数执行完 syscall 后，return 到哪里？

进阶题：

Linux 中内核态和用户态之间数据传递的有哪些方式？请结合具体代码分析。

第三部分：

exec/usys.S

sys_exec/sysfiles.c->argstr, fetchstr, argint, fetchint/syscall.c

->exec/exec.c

重点关注的问题：

- 用户态的 exec 是在哪里定义的？和一般的函数定义有什么区别。
- 第一次进行 exec 系统调用的位置在哪里？加载的程序是哪个？参数是什么？
- sys_exec 是如何获取到从用户态传递过来的参数的？
- argint 的功能是什么？它在调用 fetchint 时第一个参数是什么含义？其中 proc->tf->esp 指向的栈是用户栈还是内核栈？proc->tf->exp+4+4*n 中为什么要加 4？
- exec/exec.c 加载用户程序用到的 struct elfhdr 数据结构，其中 magic, phnum, phoff 等字段的作用是什么？以及 struct proghd 的数据结构, 其中 vaddr, memsz, filesz 等字段的作用是什么？
- exec/exec/c 执行完以后，返回的地址是什么？为什么？

进阶题：

请结合代码详细分析 Linux 中 elf 文件格式(利用 readelf 命令)，以及链接和加载的机制。

第四部分：

main/init.c->fork/usys.S->alltraps/trapasm.S->trap/trap.c->syscall/syscall.c->sys_fork/sysproc.c->for, wait, sleep/proc.c ->sched/proc.c->swtch/swtch.S

其中 alltraps, trap, syscall 简要介绍下执行流程即可。

重点关注的问题：

- 用户态程序 main/init.c 调用的 fork 的定义在哪里？和一般的函数定义有什么不同？
- 为什么子进程和父进程一样都会返回 main/init.c？
- 子进程是如何从 RUNNABLE 转换到 RUNNING 状态的？
- main/init.c 调用 fork 后，是父进程先返回还是子进程先返回？
- 对于父进程和子进程，fork 返回的 pid 相同么？为什么？
- 子进程返回后，加载的程序是什么程序？
- wait 系统调用的功能？

进阶题：

xv6 和 Linux 中调度器如何选择下一个要执行的进程？可选取一个 Linux 调度算法针对代码详细分析。