|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 武汉大学国家网络安全学院教学实验报告 | | | | | |
| 课程名称 | 操作系统设计与实践 | | 实验日期 | | 2024.10.30 |
| 实验名称 | 基本实验环境搭建 | | 实验周次 | | 第七次 |
| 姓名 | 学号 | | 专业 | | 班级 |
| 王亚鹏 | 2022302181161 | | 信息安全 | | 5班 |
| 杨依磊 | 2022302181159 | | 信息安全 | | 5班 |
| 杜泓波 | 20223021811 | | 信息安全 | | 5班 |
| 侯名扬 | 20223021811 | | 信息安全 | | 5班 |
| 1. 实验目的及实验内容   （要求掌握的知识；实验内容；原理分析） | | | | | |
| **总目标：**  **进程的实现机理与进程管理**  **实验内容：**  **1. 掌握进程相关数据结构的定义方法：**  – 进程控制块(进程表)、进程结构体、进程相关的GDT/LDT、进程相关的TSS，以及数据结构的关系  **2. 掌握构造进程的关键技术**：  – 初始化进程控制块的过程、初始化GDT和TSS、实现进程的启动  **3. 进程的现场保护与切换**  弄清楚需要哪些关键数据结构与步骤  – 时钟中断与进程调度关系，现场保护与恢复机理，从ring0-->ring1的上下文切换方法，中断重入机理  **原理分析：**  **1. 进程控制块（PCB）与进程管理**  PCB（Process Control Block）是操作系统中用于管理进程的数据结构，包含进程的状态、寄存器信息、内存指针等  **2. 进程的初始化与 GDT、TSS**  GDT（全局描述符表）和TSS（任务状态段）是 CPU 的硬件机制，支持进程切换和多任务处理。  GDT 保存了每个段的描述符，包括代码段和数据段的权限控制信息；而 TSS 则保存了当前任务的状态信息，如栈指针、CPU 寄存器等，帮助在进程切换时保存和恢复进程的上下文。  **3. 现场保护与进程切换**  进程切换的本质是保存当前进程的状态（称为“现场保护”），并加载下一个要执行进程的状态（称为“恢复现场”）  在时钟中断发生时，操作系统调用调度程序，根据调度算法决定下一个要执行的进程，并进行上下文切换  Push/Pop 指令用于保存和恢复寄存器状态，确保在切换回该进程时，它能从切换点继续执行  **4. 环级切换（Ring 0 到 Ring 1 或 Ring 3）**  环级是 CPU 的特权级别，0 级（Ring 0）为最高权限，通常用于内核代码，3 级（Ring 3）则用于普通应用程序  实现从 Ring 0 到 Ring 1（或 Ring 3）的切换，主要目的是隔离内核与用户进程，保护内核不受用户进程的影响  切换的方式是通过修改段选择子（Segment Selector）来改变当前的权限级别。  **5. 中断重入问题**  中断重入指的是在处理中断时再次发生中断，可能导致嵌套调用或数据不一致。  通过设置标志位（如 k\_reenter），可以控制中断重入的次数，防止系统进入不可控的嵌套中断状态。  **6. 代码段与数据段的完整性检查**  完整性检查是通过定期检查当前运行进程的代码段和数据段，确保未被恶意修改。这是可信计算的一部分，防止恶意代码篡改程序执行 | | | | | |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体实验步骤） | | | | | |
| 实验环境：  **X86 32位Linux环境**  实验步骤：  **1. 掌握进程相关数据结构的定义方法：– 进程控制块(进程表)、进程结构体、进程相关的GDT/LDT、进程相关的TSS，以及数据结构的关系**  **（1）进程控制块（PCB）**：   * PCB是操作系统管理进程的核心数据结构之一，每一个进程都有一个对应的PCB * PCB包含进程状态、寄存器值、程序计数器、内存管理信息、打开的文件列表等内容   **（2）进程表**：   * 进程表是一个用于记录当前所有进程PCB的数组，操作系统通过进程表对系统中的所有进程进行管理和调度 * 结构体定义如下：     **（3）GDT（全局描述符表）和LDT（局部描述符表）**：   * GDT和LDT是用于内存管理的描述符表 * GDT主要存储系统全局的内存段信息，而LDT用于描述每个特定进程自己的内存段   **（4）TSS（任务状态段）**：   * TSS是一个特殊的数据结构，主要用于保存任务（进程）的状态信息，尤其在x86的保护模式中。 * TSS保存了任务切换时的一些必要信息，例如寄存器的值和堆栈指针。   **（5）数据结构之间的关系**：  PCB、GDT、LDT和TSS之间有紧密的关系。PCB记录进程的信息，TSS保存进程状态以便于切换，GDT和LDT用于管理内存段。他们之间数据结构的具体关系以及流程图参见课后题回答部分  **2. 掌握构造进程的关键技术：– 初始化进程控制块的过程、初始化GDT和TSS、实现进程的启动**  **初始化进程表**    cs指向LDT中第一个描述符ds、es、fs、ss都设为指向LDT中的第二个描述符gs仍然指向显存只是其RPL发生改变。  接下来eip指向TestA这表明进程将从TestA的入口地址开始运行。另外esp指向了单独的栈栈的大小为STACK\_SIZE\_TOTAL。  **初始化GDT和TSS**  GDT的准备    TSS的准备    为两者填充内容    如此，初始化工作完成  **实现进程的启动**  进程结构体的定义    首先我们使用命令行启动makefile Make clean 之后再 make，可以看到如下图所示的结果    看到了不断出现的字符“A”和不断增加的数字  **3. 进程的现场保护与切换，弄清楚需要哪些关键数据结构与步骤– 时钟中断与进程调度关系，现场保护与恢复机理，从ring0-->ring1的上下文切换方法，中断重入机理**  **（1）理解时钟中断与进程调度的关系**：   * 时钟中断是操作系统进行进程调度的基础。通过时钟中断，操作系统可以在每个时钟周期后检查是否需要切换进程，以实现多任务。 * 每次时钟中断到来时，当前进程的状态会被保存，然后通过调度算法选择下一个要运行的进程。   **（2）现场保护与恢复的机制**：   * **现场保护**：当操作系统中断一个进程时，它必须保存当前进程的所有CPU寄存器状态，以便未来恢复。这些状态通常被保存到PCB或者内核堆栈中。 * **现场恢复**：当调度程序决定恢复某个进程时，它需要将之前保存的CPU状态恢复，继续执行中断时的代码。   **（3）从ring0到ring1的上下文切换方法**：   * **ring0到ring1的切换**涉及特权级的改变。ring0是操作系统内核的特权级，ring1则可以用于一些相对可信但权限受限的操作。 * 在某些系统设计中，操作系统可能会将部分操作交由ring1来执行，以增强系统的安全性。实现从ring0到ring1的上下文切换，通常涉及调整段寄存器以更改代码和数据段的权限。 * 用一个iretd指令可以做到ring0到ring1的转移     代码解释如下：  **restart:**  mov esp, [p\_proc\_ready]  将指针 p\_proc\_ready 指向的值加载到栈指针 esp 中。这通常是在多任务环境中，切换到另一个进程的栈。  lldt [esp + P\_LDT\_SEL]  使用 lldt 指令加载局部描述符表（LDT）选择子，LDT用于定义进程的内存段信息，切换到对应进程的LDT有助于确保该进程可以访问它自己的内存段。  **lea eax, [esp + P\_STACKTOP]**  **mov dword [tss + TSS3\_S\_SP0], eax**  计算进程栈顶的位置，并将结果存储到eax寄存器中。  接着将eax的值存储到TSS（任务状态段）中，以更新TSS中的ESP0字段。这部分代码实际上是在设置特权级为0的栈顶地址，当从用户态（低特权级）切换到内核态（ring0）时，处理器会使用这个栈指针。  **pop gs**  **pop fs**  **pop es**  **pop ds**  **popad**  依次从栈中弹出gs、fs、es、ds等段寄存器，以及使用popad弹出通用寄存器的值。这一步是在恢复被中断或切换前保存的寄存器值，以便让进程可以从中断或者切换的地方继续执行。  **iretd**  iretd指令会从栈中弹出返回地址以及段选择子，用于从中断服务例程返回。iretd 不仅会恢复指令指针（EIP）和代码段寄存器（CS），还会恢复处理器的标志寄存器（EFLAGS）。  **关键部分**：iretd指令用于从中断返回，如果中断发生在特权级不同的情况下，它会自动完成特权级的切换。例如，如果之前的中断发生在ring3（用户态），而中断服务在ring0（内核态）处理，那么 iretd 会将控制权交还给ring3，即实现了从内核态返回到用户态。  **正确的段描述符配置**：  ring0 到 ring1 的切换需要段描述符中的权限级别位正确设置。一般来说，ring0 是最高权限级，ring3 是最低权限级。需要通过GDT/LDT中的段描述符设置来确定转移的目标权限级。  **中断处理的正确配置**：  代码使用 iretd 来完成中断返回，iretd可以从 ring0 返回到更低权限的特权级，比如 ring1 或 ring3，前提是返回的段选择子指向的段权限正确。  **（4）中断重入的实现**：   * **中断重入**问题出现在中断服务例程在处理时再次被相同的中断打断，这可能会导致数据混乱。 * 解决方案之一是使用一个重入计数器（例如k\_reenter），来跟踪当前中断是否是重入的。通过设计合理的逻辑来处理中断重入，可以保证系统的稳定性   书上的示例程序按照同样的方法进行make，生成的结果如下：    看到不断出现的字符“^”说明函数disp\_str运行正常而且没有影响到中  断处理的其他部分以及进程A。之所以在两次字符A的打印中间有多个“^”是  因为我们的进程执行体中加入了delay( )函数在此函数的执行过程中发生了多次中断 | | | | | |
| 1. 实验过程分析   （详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） | | | | | |
| 故障及其排除方法：  **(1)makefile时的报错：**    在尝试使用make命令编译操作系统时，出现了一些链接错误。错误信息表明kernel.bin的目标未成功生成。  报错详情  **链接器（ld）报错：**  undefined reference to '\_\_stack\_chk\_fail' 出现在多个地方：例如lib/klib.o 在函数disp\_int中，出现了 undefined reference to '\_\_stack\_chk\_fail'。  kernel/protect.o 在函数 exception\_handler 中，也遇到了类似的问题。  这些错误的关键是 undefined reference to '\_\_stack\_chk\_fail'，它们都是在链接阶段出现的问题  解决方法：  在编译代码时，添加一个编译器选项 -fno-stack-protector，以禁用栈保护功能：    增添图中阴影部分的内容  这样可以防止编译器生成需要 \_\_stack\_chk\_fail 的代码。还有一点需要注意的是make完一次，下次再次执行make时需要make clean  **(2)在终端执行命令时需要所有相关文件同时处于一个文件夹**  错误：如果没有处于同一个文件夹，make的时候会显示undefined，这就是找不到文件内的定义  解决方法：将他给的a文件夹内的所有东西移动到一个空文件夹之下，再make，这样就可以正常运行。需要主义的另外一点是bochsrc的内容要重新修改为原来的a.img，因为此时make的功能就是生成内容挂载到a.img | | | | | |
| 1. 实验结果总结   （对实验结果进行分析，完成思考题目，并提出实验的改进意见） | | | | | |
| **结果分析**：  **思考题目：**  **1. 描述进程数据结构的定义与含义：– 进程控制块(进程表)、进程结构体、进程相关的GDT/LDT、进程相关的TSS，画出数据结构的关系图**   **进程控制块 (PCB)**：PCB 是操作系统用来管理进程的核心数据结构，通常包含以下信息：   * 进程 ID（PID） * 进程状态（如就绪、运行、阻塞等） * 程序计数器（PC）：记录程序执行的下一条指令地址 * CPU 寄存器状态：保存进程切换时的寄存器内容 * 内存管理信息（如基址、限长等） * I/O 状态信息：例如打开的文件等    **进程表**：操作系统使用一个表来管理所有 PCB 的集合，称为进程表。进程表是一个 PCB 数组，便于进程管理和查找。   **GDT (全局描述符表)**：GDT 是 CPU 中存储段描述符的表，每个进程需要在 GDT 中注册自己的段描述符，描述了代码段、数据段等的权限、基址和长度等信息。   **LDT (局部描述符表)**：LDT 是进程私有的段描述符表，用于管理进程自己的段信息，通常在多进程和多线程环境中提高数据隔离性。   **TSS (任务状态段)**：TSS 保存当前任务（进程）的 CPU 状态，包括寄存器、堆栈指针等。进程切换时，操作系统通过 TSS 保存和恢复进程的上下文  相关数据结构关系图如下：      **2. 画出以下关键技术的流程图：– 初始化进程控制块的过程、初始化GDT和TSS、实现进程的启动**      **3. 怎么实现进程的现场保护与恢复？**  **（1）现场保护：**  在时钟中断或进程切换时，将当前进程的寄存器状态保存到 TSS 或 PCB  使用 push、pushad 指令将寄存器内容推入栈中  **（2）现场恢复：**  从 TSS 或 PCB 中加载下一个进程的寄存器状态。  使用 pop、popad 指令恢复寄存器内容，保证进程从上次中断位置继续执行  **4. 为什么需要从ring0-->ring1，怎么实现？**  **（1）必要性：**  操作系统将核心代码运行在最高权限的 Ring 0，而用户进程通常在较低权限的 Ring 3。Ring 1 可作为一个中间环节，用于部分需要更高权限但不完全属于内核的代码或模块。  这种隔离能够保护系统安全，防止普通应用程序直接访问系统关键资源，避免恶意操作或程序崩溃对内核的影响。  **（2）实现方法：**  修改段选择子（Segment Selector），指定段的特权级别为 Ring 1。  设置段描述符的 DPL（Descriptor Privilege Level）字段，指定为 1。  通过 iret 指令完成特权级的切换，从 Ring 0 切换到 Ring 1。  **5. 进程为什么要中断重入，具体怎么实现，画出流程图？**  **（1）原因：**  中断重入是指在处理中断时再次发生中断。它的发生是因为中断服务程序的响应时间可能较长，可能会延迟其他高优先级的中断。  允许中断重入可以提高系统的实时性，使高优先级任务可以抢占低优先级任务的 CPU 时间。  **（2）实现流程：**  在进入中断处理程序时检查重入标志（如 k\_reenter）。  如果 k\_reenter 表示重入过多，则忽略新的中断。  如果可以重入，保存当前寄存器状态，设置重入标志，处理中断。  中断处理结束后，清除重入标志，恢复寄存器状态，返回原始进程  **（3）流程图：**    **6. 动手做：修改例子程序的进程运行于ring3，设计一个模块，每隔一个自定义时间就运行，并对当前运行的进程代码段和数据段进行完整性检查**  go\_to\_ring3:      ; 设置用户态的数据段选择子      mov ax, 0x23          ; 用户态数据段选择子 (DPL = 3)      mov ds, ax      mov es, ax      mov fs, ax      mov gs, ax      ; 设置用户态栈      mov eax, USER\_STACK\_TOP ; 用户栈顶地址      push dword 0x23          ; 用户态数据段选择子 (DPL = 3)      push eax                 ; 用户态栈指针      ; 设置返回地址      pushfd                   ; 保存标志寄存器      push dword 0x1B          ; 用户态代码段选择子 (DPL = 3)      push dword user\_entry    ; 用户态入口地址      ; 切换到用户态      iretd  user\_entry:      ; 用户态代码，简单测试      mov eax, 0xDEADBEEF     ; 简单例子：将值存入eax  .loop:      ; 模拟工作，保持在ring3      nop      jmp .loop  timer.h  #ifndef TIMER\_H  #define TIMER\_H  void timer\_init();  void timer\_handler();  #endif // TIMER\_H  Integrity.h  #ifndef INTEGRITY\_H  #define INTEGRITY\_H  #include <stdint.h>  void integrity\_check();  uint32\_t calculate\_checksum(uint8\_t \*start, size\_t size);  void integrity\_init();  // 定义代码段和数据段的起始地址及大小（示例值）  #define USER\_CODE\_START 0x400000   // 假设用户代码段起始地址  #define USER\_CODE\_SIZE  0x1000     // 用户代码段大小  #define USER\_DATA\_START 0x500000   // 假设用户数据段起始地址  #define USER\_DATA\_SIZE  0x1000     // 用户数据段大小  #endif // INTEGRITY\_H  timer.c  #include "timer.h"  #include "integrity.h"  #include "print.h"      // 假设该文件用于内核打印输出函数，例如 printk  static int tick\_count = 0;  // 定时器中断处理函数  void timer\_handler() {      tick\_count++;      // 每隔指定的时间触发完整性检查      if (tick\_count >= CUSTOM\_INTERVAL) {          integrity\_check();          tick\_count = 0;      }  }  // 定时器初始化函数  void timer\_init() {      register\_interrupt\_handler(IRQ0, timer\_handler);      printk("Timer initialized.\n");  }  integrity.c  #include "print.h"  // 原始的代码段和数据段的哈希值，用于后续检查（启动时计算）  uint32\_t original\_code\_checksum;  uint32\_t original\_data\_checksum;  // 简单的哈希函数用于完整性检查  uint32\_t calculate\_checksum(uint8\_t \*start, size\_t size) {      uint32\_t checksum = 0;      for (size\_t i = 0; i < size; i++) {          checksum += start[i];      }      return checksum;  }  // 完整性检查函数  void integrity\_check() {      uint32\_t code\_checksum = calculate\_checksum((uint8\_t \*)USER\_CODE\_START, USER\_CODE\_SIZE);      uint32\_t data\_checksum = calculate\_checksum((uint8\_t \*)USER\_DATA\_START, USER\_DATA\_SIZE);      // 与之前保存的原始哈希值进行比较      if (code\_checksum != original\_code\_checksum || data\_checksum != original\_data\_checksum) {          printk("[ERROR] Integrity check failed: code or data segment modified!\n");          // 可以采取进一步的措施，例如终止进程或系统报警      } else {          printk("[INFO] Integrity check passed.\n");      }  }  // 初始化完整性检查的原始哈希值（通常在系统启动时调用）  void integrity\_init() {      original\_code\_checksum = calculate\_checksum((uint8\_t \*)USER\_CODE\_START, USER\_CODE\_SIZE);      original\_data\_checksum = calculate\_checksum((uint8\_t \*)USER\_DATA\_START, USER\_DATA\_SIZE);      printk("Integrity check initialized.\n");  }  这就是到达ring3的汇编源码，每隔一定时间自动运行  文件组织情况如下：   * **timer.h**：声明与定时器中断相关的函数，如 void timer\_handler(); 和 void timer\_init();。 * **integrity.h**：声明完整性检查函数和工具函数，如 void integrity\_check(); 和 uint32\_t calculate\_checksum(uint8\_t \*, size\_t);   **文件总结**   * **kernel/timer.c**：负责定时器中断的逻辑，包含定时器的初始化和定时器中断处理函数 timer\_handler()。 * **kernel/integrity.c**：包含与内存段和代码段的完整性检查相关的功能，包括 integrity\_check() 函数及其他辅助函数   **运行结果：保存在res.txt中，便于查看**    **改进意见：**  **1. 增加清理目标和编译过程的说明**  实验过程中，手动清理之前的中间文件可能比较麻烦，这可能导致一些旧文件的错误积累，影响最终编译的结果。  **改进建议**：   * 在Makefile中增加make clean目标，用于清除所有中间文件。 * **示例代码**： * clean:   rm -f \*.o \*.bin \*.img os-image  **2.深入实验建议**：  **添加更多调试工具的使用**：鼓励学生使用调试工具（如GDB）来深入理解内核代码执行情况。可以提供一些调试技巧，如如何在Bochs中启用GDB调试模式，以及如何设置断点来观察内核启动过程。  **探索不同的内核功能实现**：增加一些扩展任务，例如实现简单的内存管理单元（MMU）或者中断处理程序，帮助学生更好地理解操作系统各模块之间的关系。  **挑战性任务**：如修改内核代码以支持多任务调度或实现一个简化的文件系统。这些任务可以帮助学生加深对操作系统复杂机制的理解 | | | | | |
|  | | | | | |
| 1. 各人实验贡献与体会（每人各自撰写） | | | | | |
| 实际上每个人都各自独立地完成了实验  1.王亚鹏：全程参与实验  通过对内核代码的编译、链接以及Bochs和makefile的结合使用。这次实验不仅能够深入理解了操作系统的基本原理，还帮助我提高了对系统底层技术的动手能力  **1. 深入理解编译与链接**  遇到了\_\_stack\_chk\_fail的链接错误问题。这个问题源于GCC默认启用的栈保护功能，而内核代码因为不适合直接依赖标准C库，导致了链接失败。查阅资料，对Makefile的研究和修改，增加了-fno-stack-protector选项，最终成功解决了这一问题  **2. make clean的重要性**  在最初的几次尝试中，我遇到了一些由之前未清理的中间文件引发的问题，这些文件导致了不一致的编译结果。在增加了make clean目标并在每次编译前执行后，这些问题得到了有效解决  **3. 使用Bochs与makefile结合**  Bochs之前使用时需要输入很多命令去逐个产生依赖文件，这是相当耗费时间的一件事。通过makefile，我们可以只是用一个make指令，其他的所有指令都在makefile文件中定义好了。这样可以极大地提升效率  **4. 深入理解进程的工作原理**  通过实验，我对栈保护（Stack Protector）、全局描述符表（GDT）、任务状态段（TSS）等操作系统底层概念有了更加清晰的认知。在理解这些概念的基础上，能够更好地理解进程管理和特权级的实现。尤其是在特权级切换和现场保护的部分，学会了如何通过代码实现从ring0到ring3的安全切换  2.杨依磊：全程参与实验  1. 进程相关数据结构的掌握在本次实验中，我了解了PCB的定义和作用 ，它包含了进程的状态、寄存器内容等关键信息，为操作系统在不同进程之间切换提供了必要的数据。通过绘制PCB与GDT、LDT、TSS的关系图，我能够更直观地认识这些数据结构之间的关系。2. 构造进程的关键技术实验的一个重点在于构造进程的过程，包括初始化PCB，GDT和TSS，以及实现进程的启动，其实现在代码层面的核心在于保证各个数据结构的初始化及其相互关联的正确性。 我对实验代码进行了调试，理解了这些步骤的具体实现方法3. 进程的现场保护与切换 r文件夹中的代码实现的是进程的上下文切换，本次实验中是时钟中断引发的进程调度机制。在ring0到ring1的上下文切换过程中，需要注意数据的保存与恢复机制。实验通过简化的k\_reenter机制解决了重入问题，这是在复杂操作系统设计中处理类似的多重中断情况的一种有效方式。4. 进程的中断重入机制在本次实验中，我了解了中断重入的原因及解决方法。中断重入是由于中断在未处理完上一个中断时再次发生，导致系统状态不一致。在实验中，我实现了通过k\_reenter标志位的简单处理机制，有效地规避了多次中断引发的混乱现象。 5. 实践操作与思考通过修改例子程序，使进程运行在ring3模式并设计了定时完整性检查模块， 使我了解到操作系统如何利用CPU的不同权限级别实现对资源的保护以及操作系统如何周期性地监控进程的状态，以确保进程代码段和数据段的未被篡改。  3.杜泓波：全程参与实验  在本次实验中，我深入学习并实践了操作系统进程管理的核心概念与实现机制，尤其是进程控制块（PCB）、全局描述符表（GDT）、任务状态段（TSS）、现场保护与恢复等关键技术。通过实验，我对进程的生命周期管理、上下文切换和中断处理机制有了更加全面的理解，尤其是在涉及到硬件支持的部分，如GDT和TSS的初始化与使用，以及如何通过它们实现进程切换。  1. 进程控制块（PCB）与进程管理  首先，我掌握了进程控制块（PCB）的结构及其在操作系统进程管理中的核心作用。PCB存储了与进程相关的所有重要信息，包括进程状态、程序计数器、寄存器内容、内存管理信息等。通过对PCB的初始化和管理，操作系统能够高效地调度和管理进程。PCB作为进程调度的核心数据结构，直观地体现了进程的各种状态信息，也为进程的切换提供了必要的基础。  2. GDT/TSS 的初始化与进程切换  在实验中，我对GDT和TSS的初始化过程有了深刻的认识。GDT是实现内存保护和权限控制的重要机制，它保存了每个段（如代码段和数据段）的描述符，确保不同特权级别的代码和数据段能够被正确访问。TSS则是支持进程切换的关键硬件机制，它记录了当前任务的CPU状态，包括寄存器信息、栈指针等。在进程切换时，TSS帮助保存当前进程的上下文，并恢复下一个进程的状态。  通过将进程的上下文信息保存到TSS中，实现了进程的现场保护与恢复。在时钟中断发生时，操作系统能够根据调度算法选择下一个进程，并通过TSS和GDT完成上下文切换。这一过程让我深刻理解了操作系统如何通过硬件机制实现多任务处理，并确保不同进程的独立性和安全性。  3. 现场保护与进程切换  进程切换的核心在于现场保护与恢复。在实验中，我学习了如何通过Push和Pop指令保存和恢复进程的寄存器状态，确保进程在切换回时能从切换点继续执行。尤其是在时钟中断发生时，操作系统需要保存当前进程的状态（现场保护），然后调度下一个进程并恢复其状态（恢复现场）。这一过程让我更深刻地理解了进程调度的本质：通过保存和恢复进程的运行状态，实现多任务并发运行。  4. 特权级切换（Ring 0到Ring 3）  实验中，环级切换（从Ring 0到Ring 3）让我理解了操作系统如何通过硬件机制实现内核态和用户态的隔离。通过修改段选择子（Segment Selector），内核可以安全地切换到用户态进程，而不会影响系统的整体安全性。这种机制确保了用户进程不能直接访问或修改内核的关键数据，增强了系统的稳定性和安全性。  5. 中断重入问题及其解决  中断重入问题是操作系统设计中的一个经典问题。在处理中断时，如果再次发生中断，可能会导致数据不一致或嵌套错误。通过实验，我学习了如何设置标志位（如 ）来控制中断重入的次数，从而防止系统进入不可控的嵌套中断状态。这一机制确保了中断处理的有效性和系统的稳定性。  4.侯名扬：全程参与实验  1、 进程相关数据结构的定义： 进程控制块（PCB）：维护进程状态、优先级、程序计数器等信息。 进程结构体：用于描述进程的具体数据结构，通常包含PCB。 GDT/LDT：全局描述符表和局部描述符表，分别管理系统和特定进程的段。 TSS：任务状态段，存储进程的现场信息，如寄存器状态和栈指针。 2、构造进程的关键技术： 初始化PCB：在创建新进程时，分配内存并设置初始状态。 初始化GDT和TSS：为新进程配置段描述符和任务状态。 实现进程启动：通过设置适当的寄存器和调用适当的启动例程来激活进程。 3、进程的现场保护与切换： 时钟中断与进程调度：时钟中断用于定期唤醒调度程序，决定哪个进程获得CPU时间。 现场保护与恢复机理：在进程切换时保存当前进程的状态，切换到新进程时恢复其状态。 4、上下文切换：从ring0到ring1的切换需要注意安全和数据保护。 | | | | | |
| 1. 教师评语   （实验报告的考评：依据实验内容完整度、实验步骤清晰度、实验结果与分析正确性、实验心得与思考的全面性、实验报告文档的规范性等五个维度综合考评）   |  |  | | --- | --- | | 85-100 | * 实验内容完整或者有超出课程实验大纲的内容； * 实验步骤详尽，能够体现完整的实验过程； * 实验结果正确且实验数据分析得当； * 实验心得与思考全面并且有自己的独立思考； * 实验报告文档规范、排版整齐。 | | 75-84 | * 实验内容较为完整； * 实验步骤较为详尽，能够体现实验过程； * 实验结果正确且实验数据分析较为得当； * 实验心得与思考全面； * 实验报告文档规范、排版较为整齐。 | | 60-74 | * 实验内容有缺失； * 实验步骤不够详尽，不能够体现完整的实验过程； * 实验结果部分正确； * 实验心得与思考无或者不够深入； * 实验报告文档规范性有待增强。 | | 60以下 | * 实验内容严重缺失、实验态度不够端正 * 实验步骤不够详尽，不能够体现完整的实验过程； * 实验结果部分正确； * 实验心得与思考无或者不够深入； * 实验报告文档规范性有待增强。 | | | | | | |
|  | | | | | |
| **教师评分（请填写好姓名、学号）** | | | | | |
| 姓名 | | 学号 | | 分数 | |
| 王亚鹏 | | 2022302181161 | |  | |
| 杨依磊 | | 2022302181159 | |  | |
| 杜泓波 | | 20223021811 | |  | |
| 侯名扬 | | 20223021811 | |  | |
| 教师签名：  年 月 日 | | | | | |