第七章作业

191220008 陈南疃

2、

（1）

引起异常控制流的事件主要有：

① 进程的上下文切换：进程上下文切换意味着原来在 CPU 上正在执行的进程将被暂时中断，操作系统执行相应的上下文切换程序，将另外一个进程切换到 CPU 上执行。

② 发生内部异常事件：某个进程在执行过程中若发生内部异常事件，则 CPU 会对异常事件进行相应的处理，通过执行一系列的操作步骤，最终把操作系统内核中相应的异常处理程序调出来执行。

③ 发生外部中断请求：某个进程在执行过程中若发生外部中断请求，则 CPU 会在当前指令执行结束后响应中断请求，通过执行一系列的操作步骤，最终把操作系统内核中相应的中断服务程序调出来执行。

（2）

程序是代码和数据的集合，程序的代码是一个机器指令序列，因而是一种静态的概念，它可以作为目标文件模块存放在磁盘中，或者作为一个存储段存在于一个地址空间中。

进程指程序的一次运行过程，是一个具有一定独立功能的程序关于某个数据集合的一次运行活动，因而具有动态的含义。

因此，进程和程序之间的最大区别就是程序是静态的概念，而进程是动态的概念。一个程序被启动执行后就变成了一个进程。

（3）

进程的引入为应用程序提供了两个方面的假象：一个独立的逻辑控制流和一个私有的虚拟地址空间。

每个进程拥有一个独立的逻辑控制流，使得程序员以为自己的程序在执行过程中独占处理器；每个进程拥有一个私有的虚拟地址空间，使得程序员以为自己的程序在执行过程中独占存储器。

程序员和语言处理系统认为一台计算机的所有资源被自己的程序独占，都以为自己的程序是在处理器上执行的和在存储空间中存放的唯一的用户程序。显然，这是一种“错觉”简化了程序员的编程以及语言处理系统的处理，即简化了编程、编译、链接、共享和加载等整个过程。

（4）

计算机系统中主要靠操作系统和 CPU 硬件提供的进程上下文切换机制和异常/中断处理机制来实现，它们可以保存被中断进程的断点、所有现场以及状态信息，从而确保每次被打断执行的进程在下次重新开始执行的时候能够从被打断的地方继续运行下去。

（5）

在进行进程上下文切换时，操作系统主要完成以下三件事情：

① 将当前进程的寄存器上下文（即现场信息）保存到当前进程的系统级上下文的现场信息中。

② 将新进程的系统级上下文中的现场信息作为新的寄存器上下文恢复到处理器的各个寄存器中。

③ 将控制转移到新进程执行。

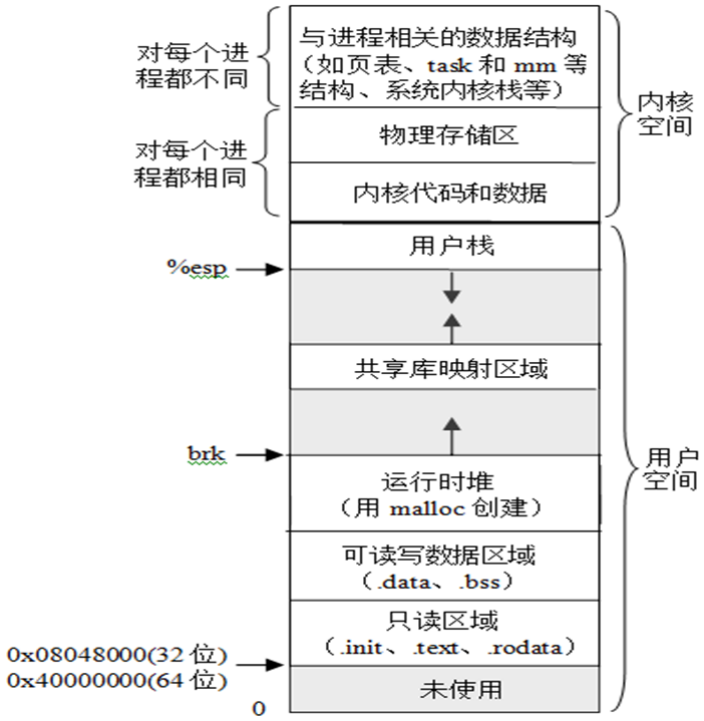
（6）

一个虚拟地址空间分为两大部分：

内核虚拟存储空间（简称内核空间）和进程虚拟存储空间（简称用户空间）。

通常，只读数据和代码段从0x8048000开始，向高地址增长。

用户栈从0xc0000000开始向低地址增长，0xc00000000 ~ 0xffffffff 为操作系统内核区。



（7）

检测到异常或中断时，CPU须进行以下基本处理：

① 关中断（“中断允许位” 清0）：使CPU处于“禁止中断”状态，以防止新中断破坏断点（PC）、程序状态（PSW）和现场（通用寄存器）。

② 保护断点和程序状态：将断点和程序状态保存到栈或特殊寄存器中。

PC→栈 或 EPC（专门存放断点的寄存器）

PSWR →栈 或 EPSWR （专门保存程序状态的寄存器）

PSW（Program Status Word）：程序状态字

PSWR（PSW寄存器）：如IA-32中的的EFLAGS寄存器

③ 识别异常事件

有软件识别和硬件识别（向量中断）两种不同的方式。

a）软件识别（MIPS采用）

设置一个异常状态寄存器（MIPS中为Cause寄存器），用于记录异常原因。操作系统使用一个统一的异常处理程序，该程序按优先级顺序查询异常状态寄存器，识别出异常事件。

（例如：MIPS中位于内核地址0x80000180处有一个专门的异常处理程序，用于检测异常的具体原因，然后转到内核中相应的异常处理程序段中进行具体的处理）

b）硬件识别（向量中断）（IA-32采用）

用专门的硬件查询电路按优先级顺序识别异常，得到“中断类型号”，根据此号，到中断向量表中读取对应的中断服务程序的入口地址。

所有事件都被分配一个“中断类型号”，每个中断都有相应的“中断服务程序”，可根据中断类型号找到中断服务程序的入口地址。

（8）

调试程序时的单步跟踪是通过陷阱机制实现的。

（9）

在异常和中断响应过程中，CPU 保存的最基本的信息应该包括断点（中断处理后返回的地址）、断点处的机器
状态（比如各种标志信息：在IA-32中，包括CS、EIP、标志寄存器、SS、ESP）。

（10）

（软件）保存的信息主要包含通用寄存器中的内容（即现场信息）。

（11）

普通的过程（函数）调用和操作系统提供的系统调用的相同之处是：在形式上没有区别，即它们都会从一个程序段跳转到另一个程序段执行，而且都会返回到被打断的程序段继续执行。

普通的过程（函数）调用和操作系统提供的系统调用的不同之处是：① 过程调用是在同一个进程内的代码之间进行跳转，而且不会改变运行级别；系统调用则是从用户态下执行的用户进程代码陷入内核态的操作系统内核代码去执行，相当于进行一个特殊的异常处理，因而执行 iret 或 sysexit 指令返回时，需要从内核态返回到用户态。显然，系统调用指令执行的开销比过程调用指令执行的开销大得多。② 二者在机器代码上的具体实现不同。系统调用的参数通过寄存器来传递，而普通函数的参数通过栈来传递。系统调用有一个统一入口，即系统调用处理程序system_call的首地址。

（12）

在 IA-32 实地址模式下，异常处理程序或中断服务程序的入口地址（由 16 位段地址和 16 位偏移地址组成）称为中断向址，用来存放 256 个中断向量的数据结构就是中断向量表。

在IA-32 的保护模式下，借助于中断描述符表来获得异常处理程序或中断服务程序的入口地址。中断描述符表是操作系统内核中的表，每个表项是一个中断门描述符、陷阱门描述符或任务门描述符。中断门描述符和陷阱门描述符中都会给出一个 16 位的段选择符和 32 位的偏移地址。段选择符用来指示异常处理程序或中断服务程序所在段的段描述符在 GDT 中的位置，偏移地址则给出异常处理程序或中断服务程序第一条指令所在的偏移量。

4、

（1）

上述7条指令的执行过程中，不会发生缺页故障。

因为第一行指令的虚拟地址为0x80482c0，可知其线性地址为0x80482c0。

已知页大小为4KB，故第一行指令不是一个页面的起始处，因此在执行第一行指令前面的指令时，上述的7条指令会被一起装入主存，故在执行上述7条指令时不会发生缺页故障。

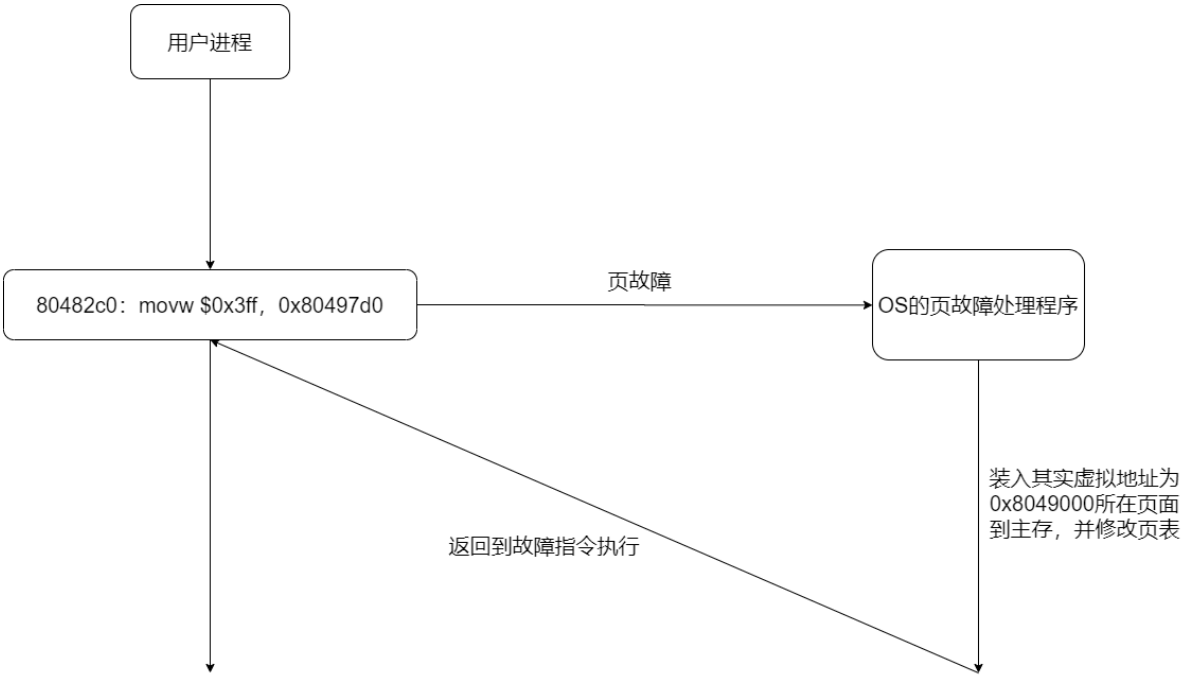
（2）

① 第1行指令：

访问存储器操作数时会发生缺页，是可恢复的故障。

当访问地址为0x80497d0的b[1000]时，是对所在页面的第一次访问（起始地址为0x8049000），故该页不在主存中，会发生缺页异常，即页故障。此时CPU将暂停P所对应的用户进程的执行，将控制转移到操作系统内核，调出页故障处理程序执行，检查与否地址越界或访问越权。这里并未发生上述情况，故将0x80497d0所在页面从磁盘调入主存。再回到这条movw指令执行时，访问数据便没有问题了。

故障处理过程示意图如下：



② 第2行指令：

访问存储器操作数时会发生缺页，是可恢复的故障。

因为地址0x804a324位于起始地址为0x804a000的页面中，当访问地址为0x804a324时是对该页面的第一次访问，故该页不在主存中，会发生缺页异常，即页故障。此时CPU将暂停P所对应的用户进程的执行，将控制转移到操作系统内核，调出页故障处理程序执行，检查与否地址越界或访问越权。这里并未发生上述情况，故将0x804a324所在页面从磁盘调入主存。再回到这条movw指令执行时，访问数据便没有问题了。

③ 第6行指令：

访问存储器操作数时不会发生缺页。

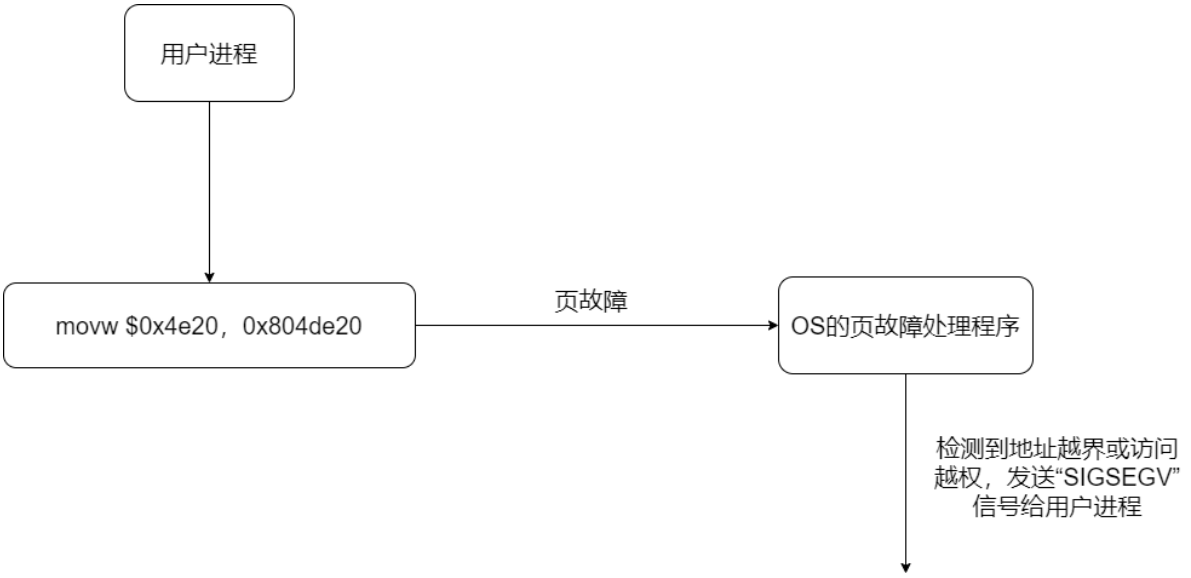
因为地址0x804a324所在页面已在执行第2行指令时调入主存中。但是b[2500]并不存在，由于编译器未对数组边界进行检查，且0x804a324处可能是变量k的地址，故该指令的执行会修改变量k的值。

④ 第7行指令：

访问存储器操作数时会发生缺页，是不可恢复的故障。

b[10000]并不存在，但编译器生成的指令中的地址0x804de20，已经偏离了数组首地址0x8049000达20002个单元，即可能偏离了4、5个页面，可能已经炒出来可读写数据区的范围。故当CPU执行该条指令时，很可能发生地址越界或访问越权。因此，CPU会通过一场响应机制转到操作系统内核，跳出内阁中的页故障异常处理程序执行。当页故障处理程序检测到地址越界或访问越权时，会发送一个“段错误”信号（SIGSEGV）给用户进程，用户进程接收到后将调出一个信号处理程序执行，根据信号类型，在屏幕上显示“段故障”信息（segmentation fault），并终止用户进程。

故障处理过程示意图如下：



（3）

会发生“整除0”故障，是不可恢复的故障。

因为k是未初始化的变量，所以k在.bss节中。

而.bss节中的变量通常初值会自动设为0，因此当执行到第5条指令时，会发生“整除0”故障，且不可恢复。

5、

（1）

执行该段代码时，系统处于用户态。因为第5行指令前的代码执行的是用户程序的功能。

执行完第5行指令后的下一个时钟周期，系统处于内核态。因为执行第5行指令时软中断指令（int $0x80），进行系统调用，转为内核态。

（2）

第5行指令属于陷阱指令。

执行该指令时，通过系统们描述符来激活异常处理程序。

对应的中断类型号是128。

对应门描述符中：P=1，DPL=3，TYPE=1111B。

取出的GDT中的段描述符中（内核代码段）：基地址=0，限界=FFFFFH，G=1，S=1，TYPE=1010，DPL=0，D=1，P=1。

（3）

① 确定中断类型号为128（0x80），从IDTR指向的IDT中取出第128个表项，这个表项实际上就是Linux初始化时在IDT的第128项中设定的系统门描述符，其中P=1，DPL=3，TYPE=1111B，段选择符为-x60，指向GDT中的内核代码段描述符。

② 根据IDT中的段选择符，从GDTR指向的GDT中取出相应的段描述符，得到对应异常处理程序或中断处理程序所在段的DPL、基地址等信息。Linux 下该段为内核代码段，因此 DPL 为 0，基地址为 0。将当前特权级 CPL (CS 寄存器最低两位，00 为内核特权级，11 为用户特权级) 与段描述符中的 DPL 比较，若 CPL 小于 DPL，则产生 13 号异常。因为 Linux 内核代码段的 DPL 总是 0, 因此不管怎样都不会发生 CPL 小于 DPL 的情况。

检查是否发生了特权级变化，即判断 CPL 是否与相应段描述符中的 DPL 不同。因为执行第 5 行指令时处于用户态，因此 CPL=3, 而 DPL=0，显然此时两者是不同的，故需要从用户态切换至内核态，以使用内核栈保存相关信息。

通过以下步骤完成用户找到内核栈的切换:

a) 读 TR 寄存器，以访问正在运行进程的 TSS 段。

b) 将 TSS 段中保存的内核栈的段选择符和栈指针分别装入寄存器 SS 和 ESP，然后在内核栈中保存原来的用户栈的 SS 和 ESP。

③ 将第 5 行后面一条指令的逻辑地址写入 CS 和 EIP，以保证内核程序处理后回到下条指令执行。在当前内核栈中保存 EFLAGS，CS 和 EIP 寄存器的内容。

④ 将 IDT 中的段选择符 (0x60) 装入 CS。将 IDT 中的偏移地址装入 EIP, 它指向内核代码段中的系统调用处理程序 system_call 的第一条指令。

这样，从下一个时钟开始，就执行系统调用处理程序 system_call 的第一条指令。在内核完成系统调用服务后，执行最后一条指令 iret，以回到第 5 行指令的下一条指令继续执行。