Pg.351

2.

1)

引起异常控制流的事件主要有哪几类?

答：①进程的上下文切换; ②内部异常;③外部中断请求;

2)

进程和程序最大的区别在哪里?

答：进程是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数据集合的一次运行活动, 进程是动态的. 而程序是静态的, 程序时代码和数据的集合, 是机器指令序列. 程序被执行后就形成了进程.

3)

进程的引入为应用程序提供了哪两个方面的假象？这种假象带来了哪些好处?

答：假象①独立的逻辑控制流; 假象②私有虚拟地址空间. 好处：使程序员认为自己的程序在运行中独占处理器和存储器.

4)

“一个进程的逻辑控制流总是确定的, 不管中间是否被其他进程打断, 也不管被打断几次或哪里被打断, 这样, 就可以保证一个进程的执行不管怎么样被打断其行为总是一致的.“计算机系统主要靠什么机制实现这个能力?

答：计算机系统主要由操作系统和CPU硬件提供的进程上下文切换机制, 异常和中断处理机制实现这个能力， 以确保能够保存被中断进程的状态信息, 运行现场和断点位置, 在下次再执行被打断的进程时能够正确的继续运行.

5)

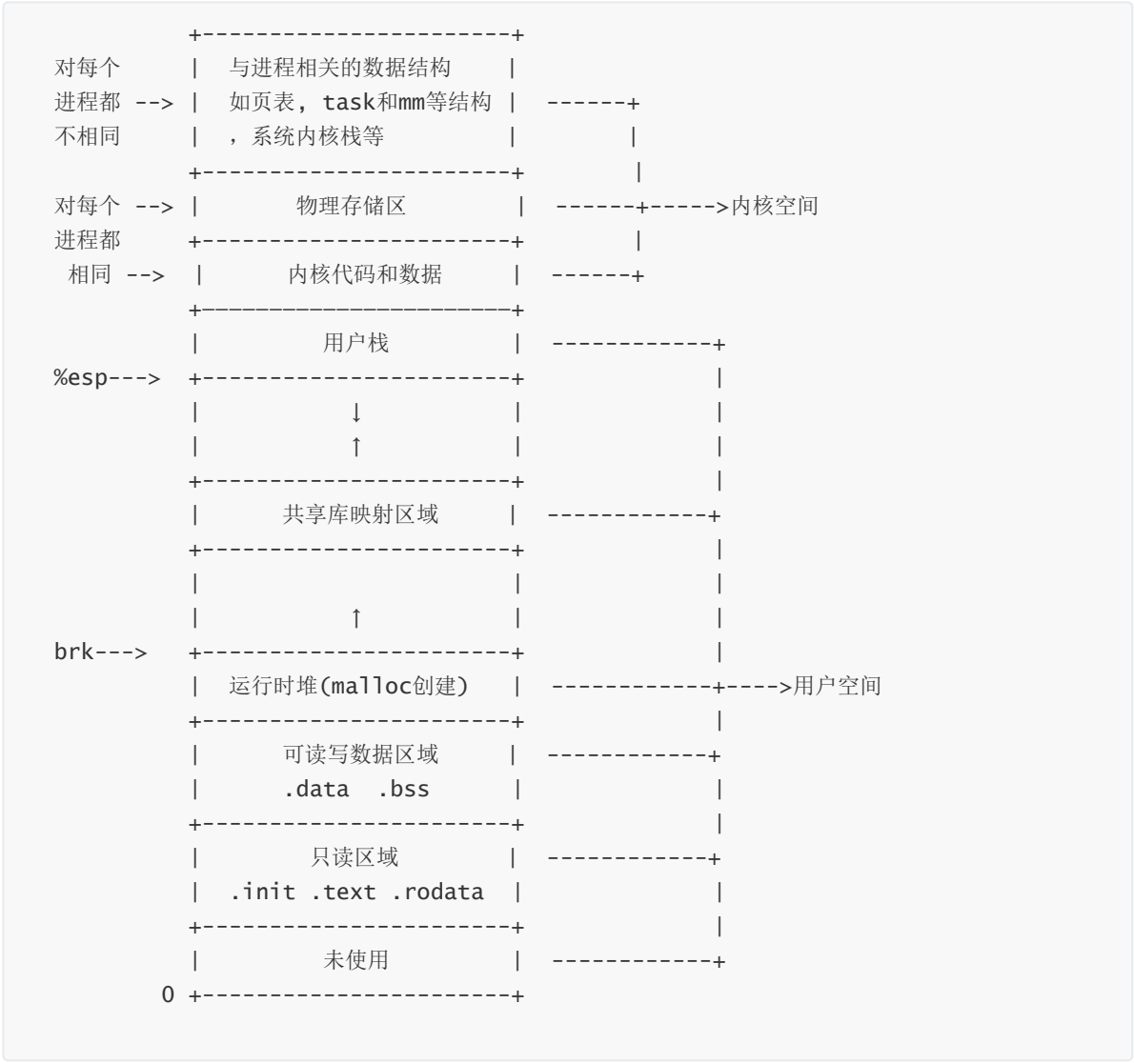
在进行进程上下文切换时, 操作系统主要是完成哪几工作?

答：①将当前进程的现场(寄存器上下文)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将控制器转移到新进程中执行.

6)

在`IA-32 + Linux`系统平台中, 一个进程的虚拟地址空间布局是怎样的?

答：如下图所示. 虚拟地址空间分为内核空间(内核虚拟存储空间)和用户空间(进程虚拟存储空间). 用户程序没有权限访问内核空间. 程序的可执行目标文件在装入时会映射到同样的虚拟地址空间. `IA32`的内核空间在0xc0000000以上的地址, 映射操作系统的内核代码数据、物理存储区以及和每个进程相关的系统级上下文数据结构(进程标识信息, 进程现场信息, 页表等进程控制信息和内核栈). 内核代码数据区在每个进程的地址空间都相同. 用户空间映射用户进程的代码数据、堆栈和用户级上下文信息. `IA-32`的用户栈从内核起始位置0xc0000000向低地址增长, 用户堆栈区中的共享库映射区域从0x40000000开始向高地址增长, 只读区域从0x08048000开始向高地址增长, 只读区域后面是可读写数据区域, 其起始地址一般要求与4KB字节对齐.



7)

简述异常和中断事件形成异常控制流的过程.

答：一个进程的正常执行过程中，可能会出现一些特殊事件，例如ctrl+c组合按键，除0，访存错误等，这些事件称为异常或中断，将会导致当前进程无法继续运行。发生异常或中断时，CPU必须跳转到具体的处理特殊事件的异常处理程序或中断服务程序去执行，故正在执行的进程逻辑流会被打断，引起一个异常控制流。

8)

调试程序时的单步跟踪是通过什么机制实现的?

答：陷阱机制。在IA-32中，当CPU处于单步跟踪状态时（`TF == 1 && IF == 1`）,每条指令都被设置成了陷阱指令，执行每条指令后，都会发生终端类型号为1的异常，转去执行单步跟踪处理程序。

9)

在异常和中断的响应过程中, CPU(硬件)要保存一些信息, 这些信息包含哪些内容?

答：①断点，即完成中断处理后的返回地址；②状态信息（CPU的标志位等信息）

IA-32中，CPU会保存用户栈的SS：ESP、EFLAGS、CS：EIP.

10)

在执行异常处理程序和中断服务程序的过程中, (软件)要保存一些信息, 这些信息包含哪些内容?

答：现场（通用寄存器的内容）。

11)

普通的过程(函数)调用和操作系统提供的系统调用之间有哪些相同之处？有哪些不同之处?

答:

相同之处：它们都会从一个程序段跳转到另一个程序段执行，都会返回到被打断的程序段继续执行。

不同之处：过程调用是在同一个进程内的代码间进行跳转，不会改变运行级别；系统调用是从用户态下的进程代码陷入内核态的操作系统代码执行，再从内核态返回用户态。

过程调用通过过程调用指令（例如IA-32中的call指令）和过程返回指令（如IA-32中的ret指令），系统调用采用陷阱指令（如IA-32中的`int $0x80`指令）和异常/中断返回指令（如IA-32中的iret指令）。

12)

在`IA-32`中, 中断向量表和中断描述符表各自记录了什么样的信息?

答：中断向量表记录了实地址模式下异常处理程序和中断服务程序的入口地址，能够存放256个4字节的中断向量，高16位是段地址，低16位是偏移量，固定在`0x00000-0x003ff`的内存区域。

中段描述符表记录了保护模式下异常处理程序和终端服务程序的入口地址。共有256个8字节的表项。表项可以是中断门描述符、陷阱门描述符或任务门描述符。下图为中断门描述符格式:

```
+-------------------------------+
|       偏移地址（A31~A16）         |
+-------------------------------+
|P|DPL|0|1|1|1|0|0 0 0 0 0 0 0 0|
+-------------------------------+
|            段选择符              |
+-------------------------------+
|       偏移地址（A31~A16）         |
+-------------------------------+
         中段描述符格式
```

Linux把P位一直置1，因为不会把段交换到磁盘上而是整页交换。DPL为访问该段需要的最低权限。DPL后面位是0，再后4位是门类型。中断门是`1110`, 陷阱门是`1111`, 任务门是`0101`.

中断门描述符和陷阱门描述符都会给出一个16位的段选择符和32位的偏移地址。

4.

1)

由页大小为4KB，第一行指令的虚拟地址为0x80482c0, 其线性地址为0x80482c0，不在页起始地址，故在执行第一行指令之前的指令时，其对应的页已被装入内存，以上7条在执行时均不会发生缺页异常。

2)

①:

在执行第一行指令时，访问数据会发生缺页，该故障可恢复。因为这是对地址为0x80497d0的b[1000]的访问，是对所在起始位置为0x8049000的页的初次访问。

故障处理过程如下图所示:

第1条指令执行过程示意图

②执行第二条指令时，访问数据也会发生缺页。因为地址0x804a324位于起始地址为0x804a000页面，这是对该页面的第一次访问。该故障可恢复。故障处理过程和第一条指令执行过程类似。

⑥执行第六条指令时，地址0x804a324在执行二条指令已经访问过，其所在页已经加载至主存，故不会发生缺页异常。

⑦执行第七条指令时，地址0x804de20所在页0x804d000为初次访问，故会发生缺页异常。该故障可能无法恢复，因为b[10000]不存在。该地址偏移了数组首地址4、5个页面，可能已经超出了可读写数据区范围，故执行时可能会发生地址越界或访问越权。这样CPU将会在页故障处理程序中发送段错误信号给用户进程，用户进程将收到该信号并终止。处理过程如下所示:

第7条指令执行过程示意图

3)

除0故障。不能恢复。

5.

1)

用户态。因为该段代码执行的是用户程序的功能。

执行完第5条指令的下一个时钟周期，系统处于内核态。

2)

第5行指令属于陷阱指令。通过系统门描述符激活异常处理程序。对应的中断类型号为128.

门描述符字段`P=1, DPL = 3, TYPE = 1111`.

GDT中段描述符中的基地址为`0x00000000`, 限界为`0xfffff`， `G=1`, `S=1`, `TYPE = 10`, `DPL = 0`, `D=1`, `P=1`.

3)

①中断类型号为`0x80`，从IDTR指向的IDT表中取出第128个表项（门描述符），其中`P=1`, `DPL=3`, `TYPE=1111`, 段选择符为`0x60`， 指向GDT表中内核代码段描述符。

②取出对应GDT中相应的段描述符，得到对应异常处理程序所在段的DPL，基地址等信息。内核代码段`DPL=0`,基地址为`0`, 当前特权级`CPL`满足`CPL>=DPL`，故不会发生越权访问。

检查是否发生特权级变化，即`CPL`是否与相应段描述符的`DPL`不同。由于执行第5行指令时处于用户态，`CPL=3`， `DPL=0`，故需要从用户态切换至内核态，用内核栈保存相关信息。切换过程：i)读TR寄存器，访问正在运行进程的TSS段；ii)将TSS段中保存的内核栈的段选择符和栈指针分别装入SS和ESP寄存器，在内核栈中保存原来用户栈的SS和ESP.

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

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