

6 实验五 缺页异常

6.1 实验目的

1. 了解发生缺页异常的原因。
2. 掌握缺页异常发生后的处理过程。
3. 复习串口的使用方式，掌握使用串口打印调试信息的方法。

6.2 页面出错异常处理

运行于开启分页机制的状态下时 ($PG = 1$)，若 CPU 在执行线性地址变换到物理地址的过程中检测到以下条件后，就会引起页面出错异常中断 int 14:

1. 当前地址变换中用到的页目录项或页表项中存在位 P 为 0。
2. 当前执行程序没有足够的特权访问指定的页面。

此时，CPU 会向出错异常处理程序提供两方面信息来诊断及纠正错误。

第一，栈中的一个出错码 (error code)。出错码格式为一个 32 位的长字，但只有最低 3 个比特位有用，它们的名称与页表项中最后三位相同 (U/S、W/R、P)，它们的含义与作用分别为：

- 位 0 (P)，异常是由于页面不存在或违反访问特权而引发。P=0，表示页面不存在；P=1 表示违反页级保护权限。
- 位 1 (W/R)，异常是由于内存读或写操作引起。W/R=0，表示由读操作引起；W/R=1，表示由写操作引起。
- 位 2 (U/S)，发生异常时 CPU 执行的代码级别。U/S=0，表示 CPU 正在执行超级用户代码；U/S=1，表示 CPU 正在执行一般用户代码。

第二，在控制寄存器 CR2 中的线性地址。CPU 会把引起异常的访问使用的线性地址存在 CR2 中，页面出错异常处理程序便可以用这个地址来定位相关的页目录和页表项。

6.3 写时复制 (Copy on Write) 机制

写时复制是一种推迟或免除复制数据的一种方法。此时内核并不复制进程整个地址空间中的数据，而是让父进程和子进程共享同一个拷贝。当进程 A 使用系统调用 `fork` 创建出一个子进程 B 时，由于子进程 B 实际上是父进程 A 的一个拷贝，因此会拥有与父进程相同的物理页。

为了达到节约内存和加快进程创建速度的目的，`fork` 函数会让子进程 B 以只读方式共享父进程 A 的物理页面，同时将父进程 A 对这些物理页面的访问权限也设为只读（该操作详见 `memory.c` 中 `copy_page_tables` 函数）。当父进程 A 或子进程 B 任何一方对这些共享物理页面执行写操作时，都会产生页面出错异常（`page_fault int 14`）中断，此时 CPU 会执行系统提供的异常处理函数 `do_wp_page` 来试图解决该异常。这就是写时复制机制。它把对内存页面的复制操作推迟到实际要进行写操作的时刻，免除了页面不会被写的情况下的页面复制操作。

`do_wp_page` 与 `un_wp_page` 函数在写时复制机制中发挥了重要作用，详见下一节。

6.4 内存管理的重要函数

以下函数参数 `error_code` 表示错误码，`address` 表示线性地址。

6.4.1 缺页异常处理函数

```
void do_no_page (unsigned long error_code, unsigned long address);
```

该函数是访问不存在页面的处理函数，页异常中断处理过程中调用的函数，在 `mm/page.s` 中被调用。

6.4.2 写时复制处理函数

```
void do_wp_page (unsigned long error_code, unsigned long address);
```

在 `mm/page.s` 中被调用，对写时复制机制中导致写入异常中断的物理页面进行取消共享操作（使用 `un_wp_page` 函数），并为写进程复制一新的物理页面，使父进程 A 和子进程 B 各自拥有一块内容相同的物理页面。该

函数还将要执行写入操作的这块物理页面标记为可访问的。最后，从异常处理函数中返回时，CPU 会重新执行刚才导致异常的写入操作指令，使进程能继续执行下去。

6.4.3 解除页面的写入保护

```
void un_wp_page (unsigned long * table_entry);
```

如果该页面存在并且在 1MB 以上（内核代码地址空间以外），直接在 FLAG 上添加 W 并刷新 TLB；否则申请一个新页面，并复制原页面的内容到新页面（Copy On Write）。

练习 exp5/mm/page.s 中包括页异常中断处理过程（注意与 BIOS 中断调用区分），主要分两种情况处理。

一. 由缺页引起的页异常中断,通过调用 memory.c 中的 do_no_page (error_code, address) 函数来处理；

二. 由页写保护引起的页异常,此时调用页写保护处理函数 do_wp_page (error_code, address) 进行处理。error_code 和 address 是函数参数, error_code（出错码）是由 CPU 自动产生并压入堆栈的，出现异常时访问的线性地址 address 是从控制寄存器 CR2 中取得的。

了解缺页异常的触发方式，在 mm/mm_test.c 中的 mmtest_main 函数中编写触发缺页异常的代码，简述你引发缺页异常的原理并展示关键代码。

代码编写完毕后运行 NEUOS，便会看到界面中提示的 page_fault，如图 13 所示。由于未处理缺页，这里会无限次触发中断。

练习 进入实验环境目录下 mm/memory.c 文件，找到 do_no_page, do_wp_page, un_wp_page 函数，其中，do_no_page 为缺页处理函数，该函数在发生缺页异常时会调用 get_free_page 函数申请一页物理内存，接下来将该 page 映射到指定的线性地址 address 处，若操作成功则返回，否则释放内存页，并调用 oom 函数报错。（此处暂不考虑进程管理以及页面共享，故代码与 Linux 0.11 内核有一定区别）

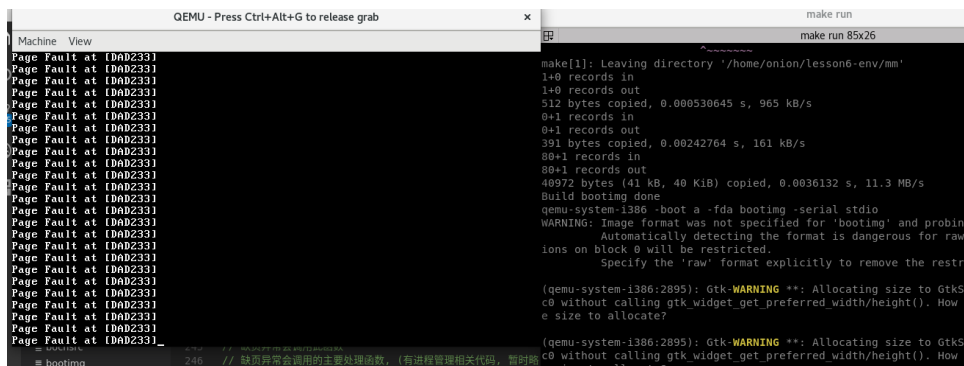


图 13: 缺页异常时的无限中断

请尝试完成 `do_no_page` 函数，之后运行 NEUOS，便可以看到缺页异常提示以及申请的新的页信息，如图 14 所示。

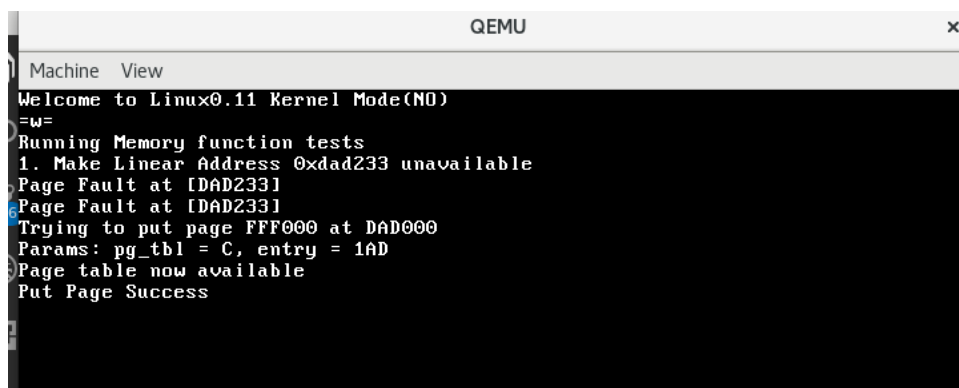


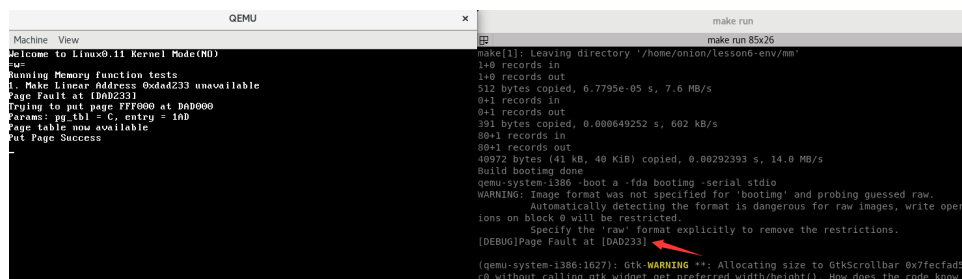
图 14: 内存函数测试

练习 读懂 `exp5/mm/memory.c` 中的 `do_wp_page` 函数调用 `un_wp_page` 时传递的 `table_entry` 表达式，并描述出具体含义。其中，`table_entry` 是 `un_wp_page` 函数所需的参数。

练习 根据实验四所学串口知识及相关代码，在 exp5/kernel/serial.c 实现空缺的函数。在 exp5/include/linux/kernel.h 中这些函数已有声明，Makefile 中的 OBJS 也已补充。

将 exp5/mm/memory.c 源文件中 do_no_page 函数中的 printk(“Page Fault at...” 修改为 s_printk(“Page Fault at...”

编译后使用 QEMU 运行 NEUOS 后即可在终端中看到串口输出的缺页异常信息，如图 15 所示。



```
Machine View
Welcome to Linux0.11 Kernel Mode(M0)
Running Memory Function Tests
1. Make Linear Address 0x0ad233 unavailable
Page Fault at 0x0ad233
Trying to put page 77f900 at 0ad900
Param: pg_tbl = C, entry = 1ad
Page Table now available
Put Page Success

make run
make run 85x26
make[1]: Leaving directory '/home/onion/lesson6-env/mm'
140 records in
140 records out
512 bytes copied, 6.7795e-05 s, 7.6 MB/s
0+1 records in
0+1 records out
391 bytes copied, 0.000649252 s, 602 KB/s
80+1 records in
80+1 records out
40972 bytes (41 KB, 40 KiB) copied, 0.00292393 s, 14.0 MB/s
Build booting done.
qemu-system-i386 -boot a -fda bootimg -serial stdio
WARNING: Image format was not specified for 'bootimg' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
(DEBUG)Page Fault at 0x0ad233
(qemu-system-i386:1627): Gtk-WARNING **: Allocating size to GtkScrollbar 0x7fecfad5c0 without calling gtk_widget_get_preferred_width/height(). How does the code know
```

图 15: 缺页异常信息

接下来，在 QEMU 的 log 中找到中断发生时的寄存器信息。

提示：qemu-system-i386 -d 命令用于输出 log 信息，要输出中断的 log 信息，尝试使用 qemu-system-i386 -d help 命令找到输出中断信息的命令。然后，补充 exp5/Makefile 中的 run 命令，使用 make run 运行 NEUOS，如图 16 所示。请描述你是如何使用 QEMU 查看中断信息的，并展示运行截图。

```
onion@onion-VM: ~/lesson6-env master *
$ 20134025

vim log 137x31
check_exception old: 0xffffffff new 0xe
0: v=0e e=0000 i=0 cpl=0 IP=0008:00007d6d pc=00007d6d SP=0010:0000bef0 CR2=00dad233
EAX=00dad233 EBX=00000003 ECX=000b8000 EDX=000006b4
ESI=00000000 EDI=00000ffc EBP=0000bf18 ESP=0000bef0
EIP=00007d6d EFL=00000406 [D---P-] CPL=0 II=0 A20=1 SMM=0 HLT=0
ES =0010 00000000 00ffffff 00c09300 DPL=0 DS [-WA]
CS =0008 00000000 007ffffff 00c09a00 DPL=0 CS32 [-R-]
SS =0010 00000000 00ffffff 00c09300 DPL=0 DS [-WA]
DS =0010 00000000 00ffffff 00c09300 DPL=0 DS [-WA]
FS =0010 00000000 00ffffff 00c09300 DPL=0 DS [-WA]
GS =0010 00000000 00ffffff 00c09300 DPL=0 DS [-WA]
LDT=0000 00000000 0000ffff 00008200 DPL=0 LDT
TR =0000 00000000 0000ffff 00008b00 DPL=0 TSS32-busy
GDT= 00005cb8 000007ff
IDT= 000054b8 000007ff
CR0=80000013 CR2=00dad233 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
CCS=000006b4 CCD=000046b4 CC0=ADDL
EFER=0000000000000000
```

图 16: QEMU 中断日志