OSLab3 进程与线程

实验报告

吴澄杰　　151220122　　151220122@smail.nju.edu.cn　　2017年5月4日

一、实验进度

此次实验，我完成了所有要求内容。我实现了类似Linux的段页式虚拟存储管理（包括缺页处理函数），实现了进程控制模块PCB，实现了fork()、exit()、sleep(int t)、getpid()等系统调用函数，实现了不同进程之间的切换和时间片轮转调度。

二、实验进度报告

1、段页式虚拟存储管理

我实现了一个比较完整的段页式存储管理。具体实现细节在\kernel\memory\mm.c中。

在分段上，仿照Linux，采用扁平模式，并且用不同的特权级区分开了内核段和用户段，用户程序使用内核的段描述符时会触发存储保护错，实现了段上的存储保护。

在分页上，首先，我以链表的形式实现了对所有物理页框的管理，并且实现了释放物理页（page_free）、获得空白物理页（page_alloc）等一系列函数供调用。其次，我实现了二级页表的虚拟存储管理，实现了包括获得对应PTE指针函数（pgdir_walk）、映射地址（boot_map_region）、插入虚拟页（page_insert）等一系列函数。然后，我实现了缺页处理函数，使得虚拟页对应的物理页不存在时可以自动分配物理页。

Kernel启动后，会初始化并且开启分段分页。Kernel的虚拟地址从0xc0000000起。在加载用户程序之前，会填写用户程序的页目录表和页表，建立用户程序的虚拟地址空间。用户程序的代码段从0x8048000起，用户栈底为0xbfffffff。并且，将Kernel的地址空间也映射到用户程序地址空间的相同位置，使得用户程序可以陷入内核态。然后利用缺页处理函数，为用户程序代码、数据自动分配物理页，从而加载并执行用户程序。

此外，内核代码的页表项的特权级为内核级，这就使得在用户态无法直接访问内核空间的内容；并且不同进程拥有不同的页表，相互之间也无法相互干涉。这就实现了分页上的存储保护。

2、进程调度

我实现了进程的时间片轮转调度。具体实现细节在\kernel\process\process.c和\kernel\device\timer.c中。

首先是进程管理模块PCB。我用链表对PCB池进行管理，以获得或者释放PCB。在初始化一个进程时，主要工作是填写页目录表和页表，并且将内核空间映射到进程空间即可。

我使用ready、blocked两个链表来管理所有的进程。所有可执行的进程依照时间先后顺序，排列在ready队列中；所有等待时钟中断唤醒的进程依照唤醒时间先后顺序，排列在blocked队列中。

每个PCB中保存着一个进程的CPU时间，达到相应的值就自动进入ready队列，并且执行ready队列中的第一个进程。若一个进程调用sleep(int time)函数休眠time毫秒，那么就会被插入blocked队列对应位置。时钟中断到来，会将current进程的CPU时间加1，并且将blocked进程的休眠时间减1。若休眠时间到达0则唤醒并且加入ready队列。

3、fork()、exit()、sleep(int time)、getpid()等系统调用函数

这些系统调用函数的内核态函数fork_kernel()、exit_kernel()、get_pid()、sleep_kernel(int hl)、wakeup()的具体实现细节见\kernel\process\call.c中。

fork()函数会创建一个与当前进程完全一样的新进程，并且加入ready队列。首先会申请一个空白的PCB，设置pid与parent等成员的值。然后，会将父进程的页目录表和页表拷贝到子进程，并且在子进程中申请新的物理页，拷贝父进程的所有物理页（deep copy）。

exit()函数会依照该进程的页表中有效的项，释放所有的物理页，并且释放PCB。然后执行ready队列的第一个进程。若没有ready进程，则停止机器。

sleep(int time)前文已经叙述。getpid()很简单，不再赘述。

4、实现功能在游戏中的体现

由于还没有实现进程间通信，因此各进程之间的数据无法访问，所以游戏功能改变有限。目前实现的情况是：在Kernel加载游戏后，首先是0号进程。0号进程会fork出1号进程，作为游戏时间的计数器。接着，0号进程与1号进程会并发执行。0号进程就是游戏本身，1号进程每隔1s会通过串口输出当前已经进行游戏的时间。

另外，为了测试和更好地展示进程调度的实现情况，在OSLab工程目录下有测试样例test。\kernel\test\test.c是测试样例的源代码。只要在Makefile的第12行将

TARGET := $(GAME)

改为

TARGET := $(TEST)

后，make clean，再重新make qemu执行，就可以执行测试样例test了。test中有5个进程，

其中1号、2号、3号、4号分别隔1s、2s、4s、8s打印长度为3、6、9、12的星号串。通过串口输出结果就可以比较好地验证fork()、sleep(int time)、时间片轮转实现的正确性。

三、实验心得

1、KISS原则的重要性

一开始面对实验题目，有一些无从下手。既要实现虚拟存储管理，又要实现进程切换。因此，必需先确定一些阶段性的小目标。比如，依照讲义上提供的分页“蓝图”，首先实现一些基础功能的函数供调用，再在将这些功能组合成分页虚存所要的管理函数。

进程切换的实现过程也是如是。首先先实现一个进程，填好页表，使之可以被加载运行。然后再实现fork函数与进程切换，验证实现的正确性。最后再考虑时间片轮转的问题。将大问题化解为小问题可以提升解决问题的系统性与连贯性。

2、一些踩过的坑

（1）由于引入了段页式虚拟存储管理，因此必须对访存的准确性给予足够的重视。内核拥有对所有物理页的访问能力，物理内存映射到内核空间的0xc0000000-0xc8000000而用户进程只能访问自己的虚页，或者通过陷入内核态调用内核的功能。在fork()函数等需要修改页表的函数中，尤其要注意虚拟地址和物理地址的区别。PDE和PTE项中填的都是物理地址，但是修改PDE和PTE项时必须在内核态用虚拟地址进行访问，否则会出发存储保护错。

（2）由于在do_irq.S中处理中断是用到了指向当前进程的指针current，因此它默认是无法处理未加载用户程序时kernel状态中产生的中断的，因为此时尚没有中断存在。在加载第一个用户程序，给current赋值之后，由于之后处理了kernel中的中断，因此其内核栈栈顶不再是设置时候的值了，变成了kernel的栈地址。在iret之前需要重新对current的内核栈栈顶赋值初始化。如果不这样做的话，在fork()的拷贝内核栈过程中会产生存储保护错。

（3）在当前实现下，用户程序的入口函数必须是void类型。由于用户栈栈底设置在0xbfffffff，它上面1个字节就是内核空间了。如果不是void类型，那么编译器默认会访问栈中本身不存在的返回值的存储位置，就导致了在用户栈的访问过程中进入了内核空间，出发存储保护错。因此，必需使用没有返回值的void类型函数作为用户程序的入口函数。