LAB5 田新韬 2012011268

Ex0

先 diff 修改后的 lab4 与原 lab4，再 patch 到 lab5 中

Ex1 加载应用程序并执行

更新了 LAB4 中的代码，在 proc_alloc 方法中额外初始化了成员 wait_state，cptr，yptr，optr；在 do_fork 方法中调用 set_links
初始化 tf 中的变量，按照注释编码即可。
然后跑不通，发现需要更新 LAB1 中代码，使系统调用对应的中断描述符可以在用户态被调用。同时需要更新 LAB3 中代码，使处理时钟中断时每过 TICKNUM 将当前进程设置为 need_reschedule。

请在实验报告中描述当创建一个用户态进程并加载了应用程序后，CPU 是如何让这个应用程序最终在用户态执行起来的。即这个用户态进程被 ucore 选择占用 CPU 执行（RUNNING 态）到具体执行应用程序第一条指令的整个经过。
用户态进程调用链：
exec(中断处理,修改 eip)->syscall(查找系统调用号)->sys_exec->do_execve(换 PDT，换页表)->load_icode(初始化内存空间，加载 elf，current->tf)
中断返回，切换堆栈，转换特权级，跳转到用户程序入口。

Ex2 实现 Copy_range

按照注释提示编码即可。

请在实验报告中简要说明如何设计实现”Copy on Write 机制“，给出概要设计，鼓励给出详细设计。
在拷贝页时，如果是读操作，将原来的页 W 置 0，刷新 TLB，返回原来的页；如果是写操作，进行拷贝。
对于引用计数 1 的页面，写操作则直接写，不复制。
若尝试对只读页面进行写操作，抛出异常，并在异常处理中拷贝页面给试图写操作的进程，修改页表，重新映射虚拟地址，W 置 1。

Ex3 fork/exec/wait/exit

请分析 fork/exec/wait/exit 在实现中是如何影响进程的执行状态的？

fork 不改变当前进程的执行状态，对于 fork 出来的新进程，先是 uninit，在 wakeup_proc 过后变为 runnable。
exec 不改变进程执行状态。
wait, 若存在 zombie 状态子进程, 不会改变执行状态, 若不存在, 将当前进程置为 sleeping, 直至出现 zombie 状态的子进程，exit 掉子进程，将父进程置为 runnable。
exit,先将该进程置为 zombie，如有父进程则唤醒（由 sleeping 置为 runnable）。

请给出 ucore 中一个用户态进程的执行状态生命周期图（包执行状态，执行状态之间的变换关系，以及产生变换的事件或函数调用）。（字符方式画即可）

--alloc_page()--> UNINIT --wakeup_proc()--> RUNNABLE – proc_run()-->RUNNING
--do_exit()--> ZOMBIE --do_wait()--> exit

实验结果

```
[~/moocos/ucore_lab/labcodes/lab5]
moocos-> make grade
badsegment:              (2.0s)
  -check result:                             OK
  -check output:                             OK
divzero:                 (2.0s)
  -check result:                             OK
  -check output:                             OK
softint:                 (1.7s)
  -check result:                             OK
  -check output:                             OK
faultread:               (1.9s)
  -check result:                             OK
  -check output:                             OK
faultreadkernel:         (2.0s)
  -check result:                             OK
  -check output:                             OK
hello:                   (2.0s)
  -check result:                             OK
  -check output:                             OK
testbss:                 (1.9s)
  -check result:                             OK
  -check output:                             OK
pgdir:                   (2.0s)
  -check result:                             OK
  -check output:                             OK
yield:                   (2.0s)
  -check result:                             OK
  -check output:                             OK
badarg:                  (2.0s)
  -check result:                             OK
  -check output:                             OK
```

```
exit:                   (1.8s)
  -check result:                        OK
  -check output:                        OK
spin:                   (5.0s)
  -check result:                        OK
  -check output:                        OK
waitkill:               (14.1s)
  -check result:                        OK
  -check output:                        OK
forktest:               (2.0s)
  -check result:                        OK
  -check output:                        OK
forktree:               (2.1s)
  -check result:                        OK
  -check output:                        OK
Total Score: 150/150
```