Lab5 Report

12011702 张镇涛

1

通过阅读手册，我们得知可以通过fork()的返回值判断“当前”两个进程哪个是父进程，哪个是子进程。

当fork()的返回值为0时，该进程为子进程，否则为父进程(返回值为子进程的PID)。

父子进程执行顺序**不固定**。

2

该父进程的所有子进程会被过继给init进程或者注册过的祖父进程，由这个进程处理后续wait()操作，当子进程结束后回收僵尸进程。

3

```
os12011702@vmos-tony:~$ ps -al
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000    1054    1048  0  80   0 - 74029 ep_pol tty2     00:02:32 Xorg
0 S  1000    1171    1048  0  80   0 - 48451 do_sys tty2     00:00:00 gnome-ses
0 R  1000   35789   35684 99  80   0 -   624 -      pts/0    00:00:20 a
1 Z  1000   35790   35789  0  80   0 -     0 -      pts/0    00:00:00
0 R  1000   35810   35801  0  80   0 -  3622 -      pts/1    00:00:00 ps
os12011702@vmos-tony:~$
```

```
os12011702@vmos-tony:~/oslab/lab5$ ./a
Parent PID=35789, Child PID=35790
```

Open　zombie.c　~/oslab/lab5　Save

```c
#include <stdio.h>
#include <unistd.h>

int main(){
int pid;
pid=fork();
if (pid>0){
   printf("Parent PID=%d, Child PID=%d\n", getpid(), pid);
   while(1){
   }
}
else{

}
return 0;
}
```

注意到PID=35790的子进程已经成为僵尸进程（Stat为Z）。

4

```
int pid = kernel_thread(init_main, "Hello world!!", 0);
if (pid <= 0) {
    panic("create init_main failed.\n");
}
initproc = find_proc(pid);
set_proc_name(initproc, "init");
```

在proc.c文件中，通过kernel_thread()创建进程

```
int kernel_thread(int (*fn)(void *), void *arg, uint32_t clone_flags) {
    struct trapframe tf;
    memset(&tf, 0, sizeof(struct trapframe));
    tf.gpr.s0 = (uintptr_t)fn;
    tf.gpr.s1 = (uintptr_t)arg;
    tf.status = (read_csr(sstatus) | SSTATUS_SPP | SSTATUS_SPIE) & ~SSTATUS_SIE;
    tf.epc = (uintptr_t)kernel_thread_entry;
    return do_fork(clone_flags | CLONE_VM, 0, &tf);
}
```

在kernel_thread()中，设置tf中s0寄存器指向进程要执行的函数。然后将epc寄存器设为指向kernel_thread_entry的指针，这是进程执行的入口函数。

```
.text
.globl kernel_thread_entry
kernel_thread_entry:        # void kernel_thread(void)
    move a0, s1
    jalr s0
    jal do_exit
```

在entry.S中，a0寄存器中从s1设置执行函数需要的参数。随后通过**jalr指令跳转到s0寄存器的地址**（即我们想要执行的函数init_main()）。

5

调用过程：

在进程创建时，do_fork() 函数把当前的进程复制一份, 其中调用的copy_thread()函数，将上下文中的ra寄存器设置为了forkret()函数的入口。

进程切换时，运行proc_run()函数, 其中会调用switch.S中的switch_to, 将需要保存的寄存器进行保存和调换。

switch_to结尾的ret语句会使得pc寄存器内写入ra的值，从而下一步跳转至ra的存储的指令地址执行。

之前提到ra寄存器已经设置为了forkret函数的入口，所以跳转至forkret(), 而forkret函数再进一步调用forkrets（位于trapentry.S）：

```
    .globl __trapret
__trapret:
    RESTORE_ALL
    # go back from supervisor call
    sret

    .globl forkrets
forkrets:
    # set stack to this new process's trapframe
    move sp, a0
    j __trapret
```

这里把传进来的参数a0，也就是进程的中断帧的地址放在了sp，最后在__trapret 中就可以直接从中断帧里面恢复所有的寄存器，sret会跳转至epc所指向的函数（即前面kernel_thread_entry）。