贵州大学 计算机科学与技术学院 实验报告

院（系）名称	示范性软件学院	班级	软工 206	课程名称	Linux 系统
实验名称	实验四	日期	9 月 18 日	指导教师	王老师
学号、姓名	2000770081 秦小龙	成绩(10 分制)：		批改日期：	

1. 实验名称
   进程管理

2. 实验目的
   熟悉 Linux 进程创建系统调用 fork、vfork、clone；认识进程的地址空间（共享的以及独占的）；进一步认识进程并发以及简单的进程调试。

2. 内容（原理、方法）
   分别用 fork、vfork、clone 创建进程；查看父子和兄弟进程中变量值的变化情况，查看进程中变量的逻辑地址和物理地址；查看进程的状态等信息。

3. 结果、分析与建议
   a) 了解 fork、vfork、clone 系统调用的区别；了解 gdb 的调试命令。
      - man fork

```
[root@hadoop100 ~]# man fork
```

```
FORK(2)                    Linux Programmer's Manual                    FORK(2)

NAME
       fork - create a child process

SYNOPSIS
       #include <unistd.h>

       pid_t fork(void);

DESCRIPTION
       fork()  creates  a  new  process by duplicating the calling process.  The new
       process, referred to as the child, is  an  exact  duplicate  of  the  calling
       process, referred to as the parent, except for the following points:

       *  The  child  has its own unique process ID, and this PID does not match the
          ID of any existing process group (setpgid(2)).

       *  The child's parent process ID is the same as the parent's process ID.

       *  The child does not inherit its parent's  memory  locks  (mlock(2),  mlock-
          all(2)).
Manual page fork(2) line 1 (press h for help or q to quit)
```

      - man vfork

```
[root@hadoop100 ~]# man vfork
```

```
VFORK(2)                    Linux Programmer's Manual                    VFORK(2)

NAME
       vfork - create a child process and block parent

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t vfork(void);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       vfork():
           Since glibc 2.12:
               _BSD_SOURCE ||
                   (_XOPEN_SOURCE >= 500 ||
                       _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED) &&
                   !(_POSIX_C_SOURCE >= 200809L || _XOPEN_SOURCE >= 700)
           Before glibc 2.12:
               _BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
               _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
 Manual page vfork(2) line 1 (press h for help or q to quit)
```

- man clone

```
[root@hadoop100 ~]# man clone
```

```
CLONE(2)                    Linux Programmer's Manual                    CLONE(2)

NAME
       clone, __clone2 - create a child process

SYNOPSIS
       /* Prototype for the glibc wrapper function */

       #define _GNU_SOURCE
       #include <sched.h>

       int clone(int (*fn)(void *), void *child_stack,
                 int flags, void *arg, ...
                 /* pid_t *ptid, void *newtls, pid_t *ctid */ );

       /* For the prototype of the raw system call, see NOTES */

DESCRIPTION
       clone() creates a new process, in a manner similar to fork(2).

       This  page describes both the glibc clone() wrapper function and the under-
       lying system call on which it is based.  The main text describes the  wrap-
 Manual page clone(2) line 1 (press h for help or q to quit)
```

b)实验步骤

i. 进程中变量的值和地址

代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdint.h>
#include <sys/stat.h>
intptr_t mem_addr(unsigned long vaddr, unsigned long *paddr)
{
    int pagesize = getpagesize();

    unsigned long v_pageindex = vaddr / pagesize;
    unsigned long v_offset = v_pageindex * sizeof(uint64_t);
    unsigned long page_offset = vaddr % pagesize;
    uint64_t item = 0;
    int fd = open("/proc/self/pagemap", O_RDONLY);

    lseek(fd, v_offset, SEEK_SET);
    read(fd, &item, sizeof(uint64_t));

    if((((uint64_t)1 << 63) & item) == 0)
    {
        printf("page present is 0\n");
        return 0 ;
    }

    uint64_t phy_pageindex = (((uint64_t)1 << 55) - 1) & item;
    *paddr = (phy_pageindex * pagesize) + page_offset;

    return *paddr;
}

int main(void)
{
    char str[10];
    int count = 1;
    unsigned long pa[2]={0,0};

    int fd = open("test.txt", O_RDWR);
    if(fork() == 0)
    {
        printf("pid:%d, ppid:%d\n",getpid(), getppid());
        read(fd, str, 10);
        count += 1;
        printf("Child process : %s\n", (char *)str);
        mem_addr((unsigned long)str, &pa[0]);
        mem_addr((unsigned long)&count, &pa[1]);
        printf("virtual addr of str=%p and count=%p, physical addr of str=%p, &count=%p\n",str,&count, p
a[0], pa[1]);
        printf("count: %d (%p), pid: %d\n", count, &count, getpid());
    }
    else
    {
        printf("pid:%d, ppid:%d\n",getpid(), getppid());
        read(fd, str, 10);

        printf("virtual addr of str=%p and &count=%p, physical addr of str=%p, &count=%p\n",str,&count,
mem_addr((intptr_t)str, &pa[0]), mem_addr((intptr_t)&count, &pa[1]));
        printf("Father process : %s\n", (char *)str);
        printf("count: %d (%p), pid: %d\n", count, &count, getpid());
    }

    sleep(10);


    return 0;
}
"proc-1.c" 67L, 1843C                                         18,5          顶端
```

编译运行：

普通用户运行

```
[q@hadoop100 experiment_004]$ ./proc-1
pid:3941, ppid:3890
virtual addr of str=0x7ffe5e93d2d0 and &count=0x7ffe5e93d2cc, physical addr of str=0x2d0, &count=0x2cc
Father process :
@
count: 1 (0x7ffe5e93d2cc), pid: 3941
pid:3942, ppid:3941
Child process :
@
virtual addr of str=0x7ffe5e93d2d0 and count=0x7ffe5e93d2cc, physical addr of str=0x2d0, &count=0x2cc
count: 2 (0x7ffe5e93d2cc), pid: 3942
```

root 超级用户运行

```
[root@hadoop100 experiment_004]# gcc -g proc-1.c -o proc-1
[root@hadoop100 experiment_004]# ./proc-1
pid:3843, ppid:2605
virtual addr of str=0x7ffda00492b0 and &count=0x7ffda00492ac, physical addr of str=0x5c9562b0, &count=0x
5c9562ac
Father process :
@
count: 1 (0x7ffda00492ac), pid: 3843
pid:3844, ppid:3843
Child process :
@
virtual addr of str=0x7ffda00492b0 and count=0x7ffda00492ac, physical addr of str=0x8e75c2b0, &count=0x8
e75c2ac
count: 2 (0x7ffda00492ac), pid: 3844
```

ii. 将上面程序 fork 替换为 vfork，并以普通用户和超级用户运行，检查输出的结果。

代码：

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdint.h>
#include <sys/stat.h>
intptr_t mem_addr(unsigned long vaddr, unsigned long *paddr)
{
    int pagesize = getpagesize();

    unsigned long v_pageindex = vaddr / pagesize;
    unsigned long v_offset = v_pageindex * sizeof(uint64_t);
    unsigned long page_offset = vaddr % pagesize;
    uint64_t item = 0;
    int fd = open("/proc/self/pagemap", O_RDONLY);

    lseek(fd, v_offset, SEEK_SET);
    read(fd, &item, sizeof(uint64_t));

    if((((uint64_t)1 << 63) & item) == 0)
    {
        printf("page present is 0\n");
        return 0 ;
    }

    uint64_t phy_pageindex = (((uint64_t)1 << 55) - 1) & item;
    *paddr = (phy_pageindex * pagesize) + page_offset;

    return *paddr;
}

int main(void)
{
    char str[10];
    int count = 1;
    unsigned long pa[2]={0,0};

    int fd = open("test.txt", O_RDWR);
    if(vfork() == 0)
    {
        printf("pid:%d, ppid:%d\n",getpid(), getppid());
        read(fd, str, 10);
        count += 1;
        printf("Child process : %s\n", (char *)str);
        mem_addr((unsigned long)str, &pa[0]);
        mem_addr((unsigned long)&count, &pa[1]);
        printf("virtual addr of str=%p and count=%p, physical addr of str=%p, &count=%p\n",str,&count, p
a[0], pa[1]);
        printf("count: %d (%p), pid: %d\n", count, &count, getpid());
    }
    else
    {
        printf("pid:%d, ppid:%d\n",getpid(), getppid());
        read(fd, str, 10);

        printf("virtual addr of str=%p and &count=%p, physical addr of str=%p, &count=%p\n",str,&count,
mem_addr((intptr_t)str, &pa[0]), mem_addr((intptr_t)&count, &pa[1]));
        printf("Father process : %s\n", (char *)str);
        printf("count: %d (%p), pid: %d\n", count, &count, getpid());
    }

    sleep(10);


    return 0;
}
```

-- 插入 -- 67,1 底端

普通用户运行

```
[root@hadoop100 experiment_004]# gcc -g proc-2.c -o proc-2
[root@hadoop100 experiment_004]# su q
[q@hadoop100 experiment_004]$ ./proc-2
pid:4098, ppid:4097
Child process :
@
virtual addr of str=0x7ffdbf686370 and count=0x7ffdbf68636c, physical addr of str=0x370, &count=0x36c
count: 2 (0x7ffdbf68636c), pid: 4098



pid:4097, ppid:4060
virtual addr of str=0x7ffdbf686370 and &count=0x7ffdbf68636c, physical addr of str=0x370, &count=0x36c
Father process :

count: 0 (0x7ffdbf68636c), pid: 4097
pid:4105, ppid:4097
Child process :
virtual addr of str=0x7ffdbf686370 and count=0x7ffdbf68636c, physical addr of str=0x370, &count=0x36c
count: 2 (0x7ffdbf68636c), pid: 4105
段错误
```

root 超级用户运行

```
[root@hadoop100 experiment_004]# ./proc-2
pid:4136, ppid:4135
Child process :
@
virtual addr of str=0x7fff93fb52d0 and count=0x7fff93fb52cc, physical addr of str=0x8d3772d0, &count=0x8
d3772cc
count: 2 (0x7fff93fb52cc), pid: 4136
pid:4135, ppid:2605

virtual addr of str=0x7fff93fb52d0 and &count=0x7fff93fb52cc, physical addr of str=0x8d3772d0, &count=0x
8d3772cc
Father process :

count: 0 (0x7fff93fb52cc), pid: 4135
pid:4166, ppid:4135
Child process :
virtual addr of str=0x7fff93fb52d0 and count=0x7fff93fb52cc, physical addr of str=0x8d3772d0, &count=0x8
d3772cc
count: 2 (0x7fff93fb52cc), pid: 4166



段错误
```

iii. ps 如何工作

```
[root@hadoop100 experiment_004]# ps
   PID TTY          TIME CMD
  2605 pts/0    00:00:00 bash
  4200 pts/0    00:00:00 ps
[root@hadoop100 experiment_004]#
```

```
[root@hadoop100 experiment_004]# strace -e file -o x ps -l 2605
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
4 S     0   2605   2596  0  80   0 - 29135 do_wai pts/0      0:00 -bash
[root@hadoop100 experiment_004]#
```

```
[root@hadoop100 experiment_004]# grep 2605 x
execve("/usr/bin/ps", ["ps", "-l", "2605"], 0x7ffeff417bd0 /* 26 vars */) = 0
stat("/proc/2605", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/2605/stat", O_RDONLY)       = 6
open("/proc/2605/status", O_RDONLY)     = 6
open("/proc/2605/cmdline", O_RDONLY)    = 6
open("/proc/2605/wchan", O_RDONLY)      = 6
readlink("/proc/2605/fd/2", "/dev/pts/0", 127) = 10
[root@hadoop100 experiment_004]#
```

```
[root@hadoop100 experiment_004]# cat /proc/2605/stat
2605 (bash) S 2596 2605 2605 34816 4240 4202752 5144 119584 1 46 6 12 128 217 20 0 1 0 8408 119336960 80
6 18446744073709551615 4194304 5100836 140722994558496 140722994557144 140613722527244 0 65536 3686404 1
266761467 18446744072460308870 0 0 17 3 0 0 1 0 0 7200240 7236240 39456768 140722994560641 1407229945606
47 140722994560647 140722994561006 0
[root@hadoop100 experiment_004]#
```

```
[root@hadoop100 experiment_004]# man proc
```

```
proc(3tcl)                          Tcl Built-In Commands                          proc(3tcl)

------------------------------------------------------------------------------------------------

NAME
       proc - 建立一个 Tcl 过程

总览 SYNOPSIS
       proc name args body
------------------------------------------------------------

描述 DESCRIPTION
       proc   命令建立一个叫做name   的新的   Tcl   过程，替换已经叫这个名字的任何现存的命令或过程。
       当调用这个新命令的时候，Tcl   解释器将执行   body   的内容。通常，name   是未限定的(unquali-
       fied)(不包括任何包含(这个过程)的名字空间的名字)，在当前名字空间中建立这个新过程。如果   name
       包含任何名字空间限定符(qualifier)，则在指定的名字空间中建立这个过程。Args
       指定给这个过程的形式参数。它由一个列表组成，可以为空，它的每个元素指定一个参数。每个参数指定符(
spec-
       i-
       fier)也可以是有一个或两个字段(field)的一个列表。如果在指定符中只有一个单一字段则它是参数的名字
；如果有两个字段，则第一个是参数名而第二个是它的缺省值。

       在调用                                                                    name
       时，为过程的每个形式参数建立一个局部变量；它的值将是在调用命令中相应的(实际)参数的值或这个参数
的缺省值。在过程调用中可以不指定有缺省值的参数。但是，必须有足够的实际参数给所有没有缺省值的形式参数，
并且没有多余的实际参数。有一种特殊情况可以允许过程有可变数目的参数。如果最后的形式参数的名字是
       args，则到这个过程的一个调用包含的实际参数可以多于过程拥有的形式参数。此时，把开始于应当被赋给
 Manual page proc(3tcl) line 1/59 54% (press h for help or q to quit)
```

```
[root@hadoop100 experiment_004]# cat /proc/2605/status
Name:   bash
Umask:  0022
State:  S (sleeping)
Tgid:   2605
Ngid:   0
Pid:    2605
PPid:   2596
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 256
Groups: 0
VmPeak:   116540 kB
VmSize:   116540 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:      3224 kB
VmRSS:      3224 kB
RssAnon:            1372 kB
RssFile:            1852 kB
RssShmem:              0 kB
VmData:     1284 kB
VmStk:       132 kB
VmExe:       888 kB
VmLib:      2148 kB
VmPTE:        60 kB
VmSwap:        0 kB
Threads:        1
SigQ:   0/15603
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 0000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 0000001fffffffff
CapEff: 0000001fffffffff
CapBnd: 0000001fffffffff
CapAmb: 0000000000000000
NoNewPrivs:     0
Seccomp:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   ffffffff,ffffffff,ffffffff,ffffffff
Cpus_allowed_list:      0-127
Mems_allowed:   00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000
000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        477
nonvoluntary_ctxt_switches:     4
[root@hadoop100 experiment_004]#
```

4. 附录(如源程序)

源代码以上截屏均有