Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работ №1 по курсу

«Операционные системы»

Группа: М80 – 201Б-18
Студент: Петрухин Д.О.
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: _____

Москва, 2020.

Содержание

Постановка задачи.

Приобретение практических навыков диагностики работы программного обеспечения.

Общие сведения о программе

Программа компилируется из одного файла lab.c. В данном файле используются заголовочные файлы sys/types.h, unistd.h, stdlib.h, string.h. В программе используются следующие системные вызовы:

1. **read/write** – предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для pipe, файлов и для потокового ввода.
2. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса. При нормальном завершении вызова в первый элемент массива(аргумент pipe) – fd[0] – будет занесен файловый дескриптор, соответствующий выходному потоку данных pip’a и позволяющий выполнять только операцию чтения, а во второй элемент массива – fd[1] – будет занесен файловый дескриптор, соответствующий входному потоку данных и позволяющий выполнять только операцию записи. Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибок.
3. **fork** – системный вызов для порождения нового процесса. Процесс, который инициировал системный вызов fork, принято называть родительским процессом (parent process). Вновь порожденный процесс принято называть процессом-ребенком (child process). Процесс-ребенок является почти полной копией родительского процесса. У порожденного процесса по сравнению с родительским изменяются значения следующих параметров: PID, PPID. При однократном системном вызове возврат из него может произойти дважды: один раз в

родительском процессе, а второй раз в порожденном процессе. Если создание нового процесса произошло успешно, то в порожденном процессе системный вызов вернет значение 0, а в родительском процессе – положительное значение, равное идентификатору процесса-ребенка. Если создать новый процесс не удалось, то системный вызов вернет в инициировавший его процесс отрицательное значение.

Средство диагностики

Утилита **strace**.

Основные файлы программы.

Файл main.c

```
#include <sys/types.h>
#include <unistd.h> // for fileno, read/write
#include <stdlib.h> //for atoi/exit
#include <string.h>


void ftoa(float n, char s[]) //float_to_char
 {
    int i, sign, k, d;

    if ((sign = n) < 0)  /* записываем знак */
        n = -n;          /* делаем n положительным числом */
    i = 0;
    k = n;
    n = n - k;
    while (n - (int)n)
        n*=10;
    d = (int)n;
    do {       /* генерируем цифры после точки в обратном порядке */
        s[i++] = d % 10 + '0';   /* берем следующую цифру */
    } while ((d /= 10) > 0);     /* удаляем */
    s[i++] = '.';
    do {
        s[i++] = k % 10 + '0';
    } while ((k /= 10) > 0);
     if (sign < 0)
         s[i++] = '-';
     s[i] = '\0';
     reverse(s);
 }

 void reverse(char s[])
 {
     int i, j;
     char c;
```

```
     for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
         c = s[i];
         s[i] = s[j];
         s[j] = c;
     }
 }



int main(){
    char num1[10], num2[10], s;
    int trub1[2];
    int trub2[2];
    double x=0, summ;
    int i, m, n;
    pid_t p;

    if(pipe(trub1)<0) return 1;
    if(pipe(trub2)<0) return 1;

    p = fork(); // create two process

    if(p == -1){ // everything is bad, the fork is not working
      return -1;
    } else if (p == 0){

      read(trub1[0], num2, sizeof(num2));
      m = atoi(num2);
      read(trub1[0], num2, sizeof(num2));
      n = atof(num2);

      // for first matrix
      for(i=0;i<m*n;i++){
        read(trub1[0], num2, sizeof(num2));
        x +=atof(num2);
      }

      summ = x / (m*n); // means value first matrix
      x = 0;
      ftoa(summ,num2);
      write(trub2[1], &num2, sizeof(num2));

      //for second matrix
      for(i=0;i<m*n;i++){
        read(trub1[0], num2, sizeof(num2));
        x +=atof(num2);
      }
      summ = x / (m*n);
      ftoa(summ,num2);
      write(trub2[1], &num2, sizeof(num2));

   //child process
    } else {

      read(STDIN_FILENO,num1,10);
      m = atoi(num1);
      if (!atoi(num1)){
        write(STDOUT_FILENO, "m should not be zero \n", sizeof "m should not
be zero\n" - 1);
        write(STDOUT_FILENO, "enter number m\n", sizeof "enter number m\n" -
1);
```

```
        read(STDIN_FILENO,num1,10);
        m = atoi(num1);
      }
      write(trub1[1], &num1, sizeof(num1));
      write(STDOUT_FILENO, "enter n not equal to zero\n", sizeof "enter n not
equal to zero\n" - 1);
      read(STDIN_FILENO,num1,10);
      n = atoi(num1);
      write(trub1[1], &num1, sizeof(num1));


      for(i=0;i<m*n;i++) {
        read(STDIN_FILENO,num1,10);
        write(trub1[1], &num1, sizeof(num1));
      }


      for(i=0;i<m*n;i++) {
        read(STDIN_FILENO,num1,15);
        write(trub1[1], &num1, sizeof(num1));
      }


      read(trub2[0], &num1, sizeof(num1));
      write(STDOUT_FILENO, "Means value first matrix\n", sizeof "Means value
first matrix\n" - 1);
      write(STDOUT_FILENO,&num1,strlen(num1));
      s = '\n';
      write(STDOUT_FILENO,&s, sizeof(s));

      read(trub2[0], &num1, sizeof(num1));
      write(STDOUT_FILENO, "Means value second matrix\n", sizeof "Means value
second matrix\n" - 1);
      write(STDOUT_FILENO,&num1,strlen(num1));
      write(STDOUT_FILENO,&s, sizeof(s));
      //parent process
    }
    return 0;
}
```

Демонстрация работы программы.

Запустим утилиту strace для 2 лабораторной работы.

```
dobb2@dobb2-Laptop:~/Рабочий стол/os_lab_02-master/src$ strace ./test1
execve("./test1", ["./test1"], 0x7ffe3449eb30 /* 63 vars */) = 0
brk(NULL)                               = 0x5583caf30000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffea344de50) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=60156, ...}) = 0
mmap(NULL, 60156, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f81fff26000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32
```

```
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0cBR\340\305\370\2609W\242\345)q\235A\1"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f81fff24000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0cBR\340\305\370\2609W\242\345)q\235A\1"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f81ffd32000
mprotect(0x7f81ffd57000, 1847296, PROT_NONE) = 0
mmap(0x7f81ffd57000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) =
0x7f81ffd57000
mmap(0x7f81ffecf000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7f81ffecf000
mmap(0x7f81fff1a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) =
0x7f81fff1a000
mmap(0x7f81fff20000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7f81fff20000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f81fff25540) = 0
mprotect(0x7f81fff1a000, 12288, PROT_READ) = 0
mprotect(0x5583cae0a000, 4096, PROT_READ) = 0
mprotect(0x7f81fff62000, 4096, PROT_READ) = 0
munmap(0x7f81fff26000, 60156)           = 0
pipe([3, 4])                            = 0
pipe([5, 6])                            = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f81fff25810) = 9260
read(0, 2
"2\n", 10)                      = 2
write(4, "2\n\0\0@\201\340\312\203U", 10) = 10
write(1, "enter n not equal to zero\n", 26enter n not equal to zero
) = 26
read(0, 2
"2\n", 10)                      = 2
write(4, "2\n\0\0@\201\340\312\203U", 10) = 10
read(0, 2
"2\n", 10)                      = 2
write(4, "2\n\0\0@\201\340\312\203U", 10) = 10
read(0, 5
"5\n", 10)                      = 2
write(4, "5\n\0\0@\201\340\312\203U", 10) = 10
read(0, 9
"9\n", 10)                      = 2
write(4, "9\n\0\0@\201\340\312\203U", 10) = 10
read(0, 9
"9\n", 10)                      = 2
write(4, "9\n\0\0@\201\340\312\203U", 10) = 10
read(0, 9
"9\n", 15)                      = 2
write(4, "9\n\0\0@\201\340\312\203U", 10) = 10
read(0, 3
```

```
"3\n", 15)                  = 2
write(4, "3\n\0\0@\201\340\312\203U", 10) = 10
read(0, 9
"9\n", 15)                  = 2
write(4, "9\n\0\0@\201\340\312\203U", 10) = 10
read(0, 3
"3\n", 15)                  = 2
write(4, "3\n\0\0@\201\340\312\203U", 10) = 10
read(5, "6.25\0\201\340\312\203U", 10)  = 10
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9260, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
write(1, "Means value first matrix\n", 25Means value first matrix
) = 25
write(1, "6.25", 46.25)                  = 4
write(1, "\n", 1
)                   = 1
read(5, "6.0\0@\201\340\312\203U", 10)  = 10
write(1, "Means value second matrix\n", 26Means value second matrix
) = 26
write(1, "6.0", 36.0)                    = 3
write(1, "\n", 1
)                   = 1
exit_group(0)                       = ?
+++ exited with 0 +++
```

Как видно из отслеживания за выполняемыми процессами наиболее часто выполняетсясистемный вызов write/read и имеется вызов pipe

Так же можно использовать ключ –T и выводить длительность сис-го вызова.

Вывод.

Я научился наблюдать за системными вызовами в Unix, используя утилиту strace. Данная утилита имеет много ключей, тем самым она является гибким инструментом для нахождения ошибок, связанных с системными вызовами. Утилита идеально подходит для отслеживания дочерних процессов, присоединению к процессу на лету и для других задач, возникающих при

отладке программы. Помимо этого с помощью утилиты можно получить много информации о программе, до изучения кода самой программы.