Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Операционные системы»

Лабораторная работа №2

Тема: Управление процессами в ОС

Студент: Будникова В.П.
Группа: М8О-207Б-19
Преподаватель: Миронов Е. С.
Дата:
Оценка:

Москва, 2020

Цель работы: Приобретение практических навыков в управление процессами в ОС и обеспечение обмена данных между процессами посредством каналов.

Задача: составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант(22):

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

Листинг программы

main.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    char filename1[64];
    char filename2[64];
    scanf("%s", filename1);
```

```
   scanf("%s", filename2);
        int fd1[2];
   int fd2[2];
   int f1 = creat(filename1, S_IWRITE | S_IREAD);
   int f2 = creat(filename2, S_IWRITE | S_IREAD);
   if (pipe(fd1) == -1 || pipe(fd2) == -1 || f1 == -1 || f2 == -1 ) {
      printf("error");
      exit(EXIT_FAILURE);
   }
   int id = fork();
   if (id == -1) {
      printf("error");
      exit(EXIT_FAILURE);
   } else if (id == 0) {                              //CHILD1
      //printf("child1: %d\n", getpid());
      if (close(fd1[1]) == -1 || close(fd2[1]) == -1 || close(fd2[0]) == -1 || dup2(fd1[0], fileno(stdin))
== -1
      || dup2(f1, fileno(stdout)) == -1 || close(fd1[0]) == -1 || execl("child1", 0)) {
         printf("error");
         exit(EXIT_FAILURE);
      }
   } else {
      int id2 = fork();
      if (id2 == -1) {
         printf("error");
         exit(EXIT_FAILURE);
      } else if (id2 == 0) {                          //CHILD2
         //printf("child2: %d\n", getpid());
         if (close(fd2[1])== -1 || close(fd1[1]) == -1 || close(fd1[0]) == -1 || dup2(fd2[0],
fileno(stdin)) == -1
         || dup2(f2, fileno(stdout)) == -1 || close(fd2[0]) == -1 || execl("child1", 0)) {
            printf("error22");
            exit(EXIT_FAILURE);
         }
      } else {                                        //PARENT
         //printf("parent: %d\n", getppid());
         if (close(fd1[0]) == -1 || close(fd2[0]) == -1) {
```

```
    printf("error");
    exit(EXIT_FAILURE);
}
char fl = 0;
int temp = 64;
int size = 0;
int count = 0;
char * str;
str = (char *)malloc(sizeof(char) * temp);
char c;
while (scanf("%c", &c) > 0) {
    if (c != '\n') {
        if (count > temp) {
            temp *= 2;
            str = (char *)realloc(str, temp);
        }
        str[count] = c;
        count++;
    } else {
        if (count != 0) {
            str[count] = '\n';
            size = count + 1;
            count = 0;
            fl = 0;
        } else {
            fl = 1;
        }
    }
    if (count == 0 && fl == 0) {
        if (rand() % 9 + 1 < 4) {
            //printf("2\n");
            if (write(fd2[1], str, sizeof(char) * size) == -1) {
                printf("error");
                exit(EXIT_FAILURE);
            }
        } else {
```

```
                    //printf("1\n");
                    if (write(fd1[1], str, sizeof(char) * size) == -1){
                        printf("error");
                        exit(EXIT_FAILURE);
                    }
                }
            }
        }
        if (close(fd1[1]) == -1 || close(fd2[1]) == -1 ) {
            printf("error");
            exit(EXIT_FAILURE);
        }
    }
}

if (close(f1) == -1 || close(f2) == -1) {
    printf("error");
    exit(EXIT_FAILURE);
}
    return 0;
}
```

child.c

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

void invert(char * str, int size) {
    char c;
    for (int i = 0; i < size / 2 + 1; ++i) {
        c = str[i];
        str[i] = str[size - i];
        str[size - i] = c;
    }
```

```
}

int main(int argc, char* argv[]) {
    int temp = 64;
    int size = 0;
    int count = 0;
    char * str;
    str = (char *)malloc(sizeof(char) * temp);
    char c;
    while (scanf("%c", &c) > 0) {
        if (c != '\n') {
            if (count > temp) {
                temp *= 2;
                str = (char *)realloc(str, temp);
            }
            str[count] = c;
            count++;
        } else {
            str[count] = '\0';
            invert(str, count - 1);
            printf("%s\n", str);
            count = 0;
        }
    }
    free(str);
}
```

makefile

```
all:
        gcc main.c -o main
        gcc child1.c -o child1
clean:
        rm -r main child1 1.txt 2.txt
read:
        cat 1.txt
        cat 2.txt

run:
        ./main < 1.t
```

Тесты и протокол исполнения

```
Lera:lab2 valeriabudnikova$ make run
./main < 1.t
Lera:lab2 valeriabudnikova$ make read
cat 1.txt
edsba
ercb
ec
ee
gfd321
fdsa
jjkl
hgffd
aaaaercb
yyyec
oiurd
cat 2.txt
rd
543
edsba
ooee
```

Вывод strace

```
strace ./main < 1.t > stracelab2.txt
execve("./main", ["./main"], 0x7fff2f094970 /* 67 vars */) = 0
brk(NULL)                               = 0x55e9ccf32000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffddb9bc250) = -1 EINVAL (Недопустимый аргумент)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=118993, ...}) = 0
mmap(NULL, 118993, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe1bf1c8000
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\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\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\0\363\377?
\332\200\270\27\304d\245n\355Y\377\t\334"..., 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)
= 0x7fe1bf1c6000
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\0\363\377?
\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fe1befd4000
mprotect(0x7fe1beff9000, 1847296, PROT_NONE) = 0
mmap(0x7fe1beff9000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x25000) = 0x7fe1beff9000
mmap(0x7fe1bf171000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x19d000) = 0x7fe1bf171000
mmap(0x7fe1bf1bc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1e7000) = 0x7fe1bf1bc000
mmap(0x7fe1bf1c2000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7fe1bf1c2000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fe1bf1c7540) = 0
mprotect(0x7fe1bf1bc000, 12288, PROT_READ) = 0
mprotect(0x55e9cc42c000, 4096, PROT_READ) = 0
mprotect(0x7fe1bf213000, 4096, PROT_READ) = 0
munmap(0x7fe1bf1c8000, 118993)          = 0
fstat(0, {st_mode=S_IFREG|0664, st_size=91, ...}) = 0
brk(NULL)                               = 0x55e9ccf32000
brk(0x55e9ccf53000)                     = 0x55e9ccf53000
read(0, "1.txt\n2.txt\nabsde\nbcre\nce\ndr\nee\n"..., 4096) = 91
creat("1.txt", 0600)                    = 3
creat("2.txt", 0600)                    = 4
pipe([5, 6])                            = 0
pipe([7, 8])                            = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7fe1bf1c7810) = 50362
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7fe1bf1c7810) = 50363
close(5)                                = 0
close(7)                                = 0
write(8, "absde\n", 6)                  = 6
```

```
write(6, "bcre\n", 5)                   = 5
write(8, "ce\n", 3)                    = 3
write(6, "dr\n", 3)                    = 3
write(6, "ee\n", 3)                    = 3
write(6, "123dfg\n", 7)                = 7
write(8, "asdf\n", 5)                  = 5
write(6, "lkjj\n", 5)                  = 5
write(6, "dffgh\n", 6)                 = 6
write(8, "345\n", 4)                   = 4
write(6, "absde\n", 6)                 = 6
write(6, "bcreaaaa\n", 9)              = 9
write(6, "ceyyy\n", 6)                 = 6
write(6, "druio\n", 6)                 = 6
write(6, "eeoo\n", 5)                  = 5
read(0, "", 4096)                      = 0
close(6)                               = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=50362, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
close(8)                               = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=50363, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
close(4)                               = 0
exit_group(0)                          = ?
+++ exited with 0 +++
```

Strace:

creat - системный вызов для создания файла(возвращает -1, при ошибке). Если файл существует, то creat сделает длину файла равной 0(очистит файл) для записи в него.

pipe - системный вызов, который создает два файловых дескриптора и помещает их в массив, первый элемент(дескриптор) массива - для чтения, второй - для записи.

close - системный вызов, который закрывает файловый дескриптор, после этого дескриптор не будет ссылаться на файл и может быть(при необходимости) использован еще раз.

clone - системный вызов для создания нового процесса. Обеспечивает разделяемые адресные пространства, родительский процесс(откуда происходит вызов) не может выполняться в том же стеке, что и дочерний. родительский процесс передает указатель на пространство памяти для дочернего.

write - системный вызов, для записи в файл. Первый аргумент - файловый дескриптор. Второй аргумент - адрес, с которого начинается буфер. Третий аргумент - количество байтов, которое нужно записать в файл, на который ссылается файловый дескриптор из буфера. При удачном завершении возвращается количество байтов, которые были записаны(в случае ошибки - "-1").

Выводы

Я изучила работу процессов и и каналов (fork и pipe), составила программу на языке си и отладила ее. Также я научилась работать с такими системными вызовами, как dub2, execl. Я поняла важность проверки системных вызовов, а точнее их возвращаемого значения, это помогает быстрее найти ошибку в программе. При выполнении лабораторной работы у меня возникли сложности с пониманием моего варианта(какие данные куда должны передаваться, и где печататься в файл), оказывается у меня просто не открылись картинки из документа. Чтение схемы помогло мне решить эту проблему.

Список литературы

1. Таненбаум Э., Бос Х. *Современные операционные системы. — 4-е изд.* — СПб.: Издательский дом «Питер», 2018. — С. 111 - 123

2. Поисковик Google [электронный ресурс] URL: https://google.com/ (дата обращения: 22.09.2020)