Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством каналов.**

Студент: Ивенкова Л.В.

Группа: М80 – 208Б-19

Вариант: 13

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

1. **Постановка задачи**

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.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

13 вариант: Child1 переводит строки в нижний регистр. Child2 превращает все пробельные

символы в символ «\_».

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). В программе создается два дочерних процесса child1 и child2. Каждый дочерний процесс связан с родительским при помощи отдельного канала pipe. \

Программа принимает на вход неограниченное количество строк произвольной длины.

Программа для дочерних процессов запускается при помощи функции execl.

Программа завершает свою работу при нажатии Ctrl+D.

Программа обрабатывает все возможные системные ошибки и выводит соответствующие сообщения в случае их возникновения.

1. **Общий метод и алгоритм решения**

Проект состоит из 3-х программ: **parent.c**, **child1.c** и **child2.c**.В программе **parent.c** осуществляются системные вызовы **fork()** для создания дочерних процессов и системный вызов **pipe()** для передачи информации с помощью потоков между процессами. Дочерние программы **child1.c** и **child2.c** запускаются при помощи функции **execl()**, которой передаются необходимые аргументы в виде списка. Системный вызов **dup2()** используется для перенаправления стандартных потоков ввода и вывода.

1. **Основные файлы программы**

**parent.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

int main(){

int fd1[2];

int fd2[2];

int fd3[2];

if(pipe(fd1) < 0 || pipe(fd2) < 0){

printf("Pipe error!\n");

exit(-1);

}

pid\_t id = fork();

if(id == -1){

printf("Fork error\n");

exit(0);

}

else if(id > 0){

printf("Введите строки:\n");

char \*str = NULL, c;

int len = 1;

str = (char\*) malloc(sizeof(char));

while((c = getchar()) != EOF) {

str[len - 1] = c;

len++;

str = (char\*) realloc(str, len);

}

str[len - 1] = '\0';

close(fd1[0]);

write(fd1[1], str, len + 1);

close(fd1[1]);

free(str);

if(pipe(fd3) < 0){

printf("Pipe error!\n");

exit(-1);

}

pid\_t id2 = fork();

if(id2 == -1){

perror("Fork error!");

return -1;

}

else if(id2 > 0){

close(fd2[0]);

close(fd2[1]);

close(fd3[1]);

int len = 1;

char \*strr = (char\*) malloc(sizeof(char));

while(str[len - 1] != '\0') {

len++;

strr = (char\*) realloc(str, len);

read(fd3[0], &strr[len], sizeof(char));

}

close(fd3[0]);

printf("\nРезультат работы:\n");

printf("%s\n", strr);

free(strr);

}

else if(id2 == 0){

dup2(fd2[0], 0);

close(fd2[0]);

close(fd2[1]);

dup2(fd3[1], 1);

close(fd3[0]);

close(fd3[1]);

execl("child2", "child2", NULL);

}

}

else if(id == 0){

dup2(fd1[0], 0);

close(fd1[0]);

close(fd1[1]);

dup2(fd2[1], 1);

close(fd2[0]);

close(fd2[1]);

execl("child1", "child1", NULL);

}

return 0;

}

**child1.c**

#include <stdio.h>

#include <unistd.h>

#include <stdbool.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

int main(){

char \*str = NULL, c;

int len = 1;

str = (char\*) malloc(sizeof(char));

while((c = getchar()) != EOF) {

str[len - 1] = c;

len++;

str = (char\*) realloc(str, len);

}

str[len - 1] = '\0';

for(int i = 0; i < len + 1; ++i){

str[i] = tolower(str[i]);

}

write(1, str, len + 1);

free(str);

return 0;

}

**Child2.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

int main(){

int l = -1;

char \*str = (char\*) malloc(sizeof(char));

do{

++l;

str = (char\*) realloc(str, (l+1) \* sizeof(char));

read(0, &str[l], sizeof(char));

}while(str[l] != '\0');

int s = l;

for(int i = 0; i < s + 1; ++i){

if(str[i] == ' ')

str[i] = '\_';

}

str[l] = '\0';

write(1, str, l + 1);

free(str);

}

1. **Демонстрация работы программы**

parsifal@DESKTOP-3G70RV4:~/OS/Lab2$ ./parent

Введите строки:

MrYaaaaUUuuuu?

Nya Crya cvA

beWARe of tHe

WeEpiNg aNgeLs

Or... ..

Puffff!!

Результат работы:

mryaaaauuuuuu?

nya\_crya\_cva

beware\_\_\_of\_the\_

\_\_\_\_weeping\_angels

or...\_\_..

puffff!!

parsifal@DESKTOP-3G70RV4:~/OS/Lab2$ strace -o log.txt ./parent

Введите строки:

Qfw FUK KCGCfufFT1

Результат работы:

qfw\_fuk\_\_\_kcgcfufft1

parsifal@DESKTOP-3G70RV4:~/OS/Lab2$ cat log.txt

execve("./parent", ["./parent"], 0x7fffc785c070 /\* 27 vars \*/) = 0

brk(NULL) = 0x7fffb8a10000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffc0b78700) = -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=32310, ...}) = 0

mmap(NULL, 32310, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f90d7968000

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) = 0x7f90d79a0000

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) = 0x7f90d7770000

mprotect(0x7f90d7795000, 1847296, PROT\_NONE) = 0

mmap(0x7f90d7795000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f90d7795000

mmap(0x7f90d790d000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f90d790d000

mmap(0x7f90d7958000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f90d7958000

mmap(0x7f90d795e000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f90d795e000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7f90d79a1380) = 0

mprotect(0x7f90d7958000, 12288, PROT\_READ) = 0

mprotect(0x7f90d79aa000, 4096, PROT\_READ) = 0

mprotect(0x7f90d799d000, 4096, PROT\_READ) = 0

munmap(0x7f90d7968000, 32310) = 0

pipe([3, 4]) = 0

pipe([5, 6]) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f90d79a1650) = 606

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL) = 0x7fffb8a10000

brk(0x7fffb8a31000) = 0x7fffb8a31000

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270:\n", 29) = 29

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

read(0, "Qfw FUK KCGCfufFT1\n", 1024) = 21

read(0, "", 1024) = 0

close(3) = 0

write(4, "Qfw FUK KCGCfufFT1\n\0\0", 23) = 23

close(4) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=606, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

pipe([3, 4]) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f90d79a1650) = 607

close(5) = 0

close(6) = 0

close(4) = 0

read(3, "q", 1) = 1

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=607, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

read(3, "f", 1) = 1

read(3, "w", 1) = 1

read(3, "\_", 1) = 1

read(3, "f", 1) = 1

read(3, "u", 1) = 1

read(3, "k", 1) = 1

read(3, "\_", 1) = 1

read(3, "\_", 1) = 1

read(3, "\_", 1) = 1

read(3, "k", 1) = 1

read(3, "c", 1) = 1

read(3, "g", 1) = 1

read(3, "c", 1) = 1

read(3, "f", 1) = 1

read(3, "u", 1) = 1

read(3, "f", 1) = 1

read(3, "f", 1) = 1

read(3, "t", 1) = 1

read(3, "1", 1) = 1

read(3, "\n", 1) = 1

read(3, "\0", 1) = 1

close(3) = 0

write(1, "\n", 1) = 1

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \321\200\320\260\320\261\320\276\321\202\321\213:"..., 33) = 33

write(1, "qfw\_fuk\_\_\_kcgcfufft1\n", 21) = 21

write(1, "\n", 1) = 1

exit\_group(0) = ?

+++ exited with 0 +++

P.S. Для работы с необходимыми опциями выполним в терминале команду:

**«strace -f -e trace=write,read -o strace.txt ./parent».**

Таким образом, в файле **strace.txt** будет содержаться информация о системных вызовах **read()** и **write()**, а так же дерево процессов. При этом к каждой строке вывода добавляется **pid** процесса, делающего системный вывод

parsifal@DESKTOP-3G70RV4:~/OS/Lab2$ strace -f -e trace=write,read,pipe -o strace.txt ./parent

Введите строки:

udwYDFY FC

Результат работы:

udwydfy\_\_fc

parsifal@DESKTOP-3G70RV4:~/OS/Lab2$ cat strace.txt

562 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

562 pipe([3, 4]) = 0

562 pipe([5, 6]) = 0

562 write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270:\n", 29) = 29

562 read(0, <unfinished ...>

563 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

563 read(0, <unfinished ...>

562 <... read resumed>"udwYDFY FC", 1024) = 11

562 read(0, "\n", 1024) = 1

562 read(0, "", 1024) = 0

562 write(4, "udwYDFY FC\n\0\0", 14) = 14

563 <... read resumed>"udwYDFY FC\n\0\0", 4096) = 14

563 read(0, <unfinished ...>

562 pipe( <unfinished ...>

563 <... read resumed>"", 4096) = 0

562 <... pipe resumed>[3, 4]) = 0

563 write(1, "udwydfy fc\n\0\0\0\0", 16) = 16

563 +++ exited with 0 +++

562 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=563, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

562 read(3, <unfinished ...>

564 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

564 read(0, "u", 1) = 1

564 read(0, "d", 1) = 1

564 read(0, "w", 1) = 1

564 read(0, "y", 1) = 1

564 read(0, "d", 1) = 1

564 read(0, "f", 1) = 1

564 read(0, "y", 1) = 1

564 read(0, " ", 1) = 1

564 read(0, " ", 1) = 1

564 read(0, "f", 1) = 1

564 read(0, "c", 1) = 1

564 read(0, "\n", 1) = 1

564 read(0, "\0", 1) = 1

564 write(1, "udwydfy\_\_fc\n\0", 13) = 13

562 <... read resumed>"u", 1) = 1

562 read(3, "d", 1) = 1

562 read(3, <unfinished ...>

564 +++ exited with 0 +++

562 <... read resumed>"w", 1) = 1

562 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=564, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

562 read(3, "y", 1) = 1

562 read(3, "d", 1) = 1

562 read(3, "f", 1) = 1

562 read(3, "y", 1) = 1

562 read(3, "\_", 1) = 1

562 read(3, "\_", 1) = 1

562 read(3, "f", 1) = 1

562 read(3, "c", 1) = 1

562 read(3, "\n", 1) = 1

562 read(3, "\0", 1) = 1

562 write(1, "\n", 1) = 1

562 write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \321\200\320\260\320\261\320\276\321\202\321\213:"..., 33) = 33

562 write(1, "udwydfy\_\_fc\n", 12) = 12

562 write(1, "\n", 1) = 1

562 +++ exited with 0 +++

1. **Выводы**

Управление процессами – одна из ключевых задач операционной системы. Обычно ОС сама создаёт необходимые для себя и для других программ процессы, но возникают ситуации, когда пользователю требуется вмешаться в работу системы.

Язык Си при подключении библиотеки unistd.h (для Unix-подобных ОС) обладает возможностью совершать системные вызовы, связанные с вводом/выводом данных, управлением файлами и каталогами и, что самое важное, управлением процессами.

Внутри программы на языке Си можно создать дополнительный, т.н. дочерний процесс, который продолжит выполнение текущей программы параллельно с родительским процессом. Для этого используется функция fork, совершающая соответствующий системный вызов. Удобство в том, что при помощи ветвлений в коде программы можно отделить код родительского процесса от кода, предназначенного для ребёнка. А можно заставить ребёнка запустить другую программу. Для этого предназначено семейство функций exec\*. Обеспечить связь между процессами можно при помощи канала pipe, запрос на создание которого можно также совершить в языке Си.

Однако не только язык Си способен совершать системные вызовы, связанные с управлением процессами. Похожие библиотеки есть на многих других языках программирования, ведь современное программное обеспечение крайне редко состоит из одного процесса.