

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт №8 “Компьютерные науки и прикладная математика”
Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу
«Операционные системы»

Группа: М8О-210Б-23

Студент: Коваль М. Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 09.10.24

Москва, 2024

Постановка задачи

Вариант 14.

Необходимо составить и отладить программу на Си. 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 переводит строки в нижний регистр, а child2 убирает все задвоенные пробелы. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void);` – создает дочерний процесс.
- `int pipe(int pipefd[2]);` - создание неименованного канала для передачи данных между процессами
- `pid_t waitpid(pid_t pid, int *status, int options)` - Ожидание завершения дочернего процесса
- `int dup(int oldfd)` — создание дубликата файлового дескриптора
- `void exit(int status)` - завершения выполнения процесса и возвращение статуса
- `int open(const char *pathname, int flags, mode_t mode)` - открытие\создание файла
- `int close(int fd)` - закрыть файл

Для динамического чтения строки была написана отдельная функция `char* get_string(int* len)`. Моя программа считывает строки. Далее создаются 2 пайпа - один для транспортировки данных между parent и child1, второй для child2. При помощи вызова `fork`, создаются 2 дочерних процесса, каждый из которых считывает строки из своего пайпа, переводит строки в нижний регистр и отправляет их в child2. При помощи еще 1 `fork`, создаются еще 2 дочерних процесса, каждый из которых убирает задвоенные пробелы, после отправляет в родительский процесс. Родительский процесс выводит результат манипуляций.

Код программы

main.c

```
#include <config/config.h>
```

```
#include <fcntl.h>
```

```
#include <log/log.h>
```

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

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

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

```
#include <string.h>
```

```
#include <sys/wait.h>
```

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

```
char *get_string(int *len) {
```

```
    *len = 0;
```

```
    int capacity = 1;
```

```
    char *s = (char *)malloc(sizeof(char));
```

```
    char c = getchar();
```

```
    while (c != '\n' && c != EOF) {
```

```
        s[(*len)++] = c;
```

```
        if (*len >= capacity) {
```

```
            capacity *= 2; // увеличиваем ёмкость строки в два раза
```

```
            s = (char *)realloc(s, capacity * sizeof(char)); // перевыделяем память на строку с увеличенной ёмкостью
```

```
        }
```

```
        c = getchar();
```

```
    }
```

```
    s[*len] = '\0'; // завершаем строку символом конца строки
```

```
    return s;
```

```
}
```

```
int main() {
```

```
    const char *args[] = {"/.trash", NULL};
```

```

int pipe1[2], pipe2[2], pipe3[2];

pipe(pipe3);

pipe(pipe1);

pipe(pipe2);


pid_t pId1 = fork();


if (pId1 == 0) {

    dup2(pipe1[0], STDIN_FILENO); // Redirect standard input to read from pipe1
    dup2(pipe3[1], STDOUT_FILENO); // Redirect standard output to write to pipe3
    close(pipe1[1]);
    close(pipe3[0]);

    int status = execv("./child1", (char **)args);

    exit(status);
}

//

pid_t pId2 = fork();

if (pId2 == 0) {

    dup2(pipe3[0], STDIN_FILENO); // Redirect standard input to read from pipe3
    dup2(pipe2[1], STDOUT_FILENO); // Redirect standard output to write to pipe2
    close(pipe2[0]);
    close(pipe3[1]);

    int status = execv("./child2", (char **)args);

    exit(status);
}

close(pipe2[1]);

```

```

close(pipe1[0]);

// dup2(STDOUT_FILENO, pipe2[0]);

int log_file = create_log_file("main");

int len = 0;

do {

    char buffer[128];

    // char *data = get_string(&len);

    char data2[128];

    ssize_t read_b = read(STDIN_FILENO, data2, 128);

    if (read_b == -1) {

        break;

    }

    write(pipe1[1], data2, strlen(data2));

    if (data2[0] == '\n') {

        break;

    }

    // free(data);

    ssize_t bytes = read(pipe2[0], buffer, 128);

    if (bytes == -1) {

        break;

    }

    log_to_file(log_file, "got %ld bytes\n", bytes);

    char str_buff[128];

    snprintf(str_buff, 128, "Got output: %s\n", buffer);

    write(STDOUT_FILENO, str_buff, strlen(str_buff));

    log_to_file(log_file, "got %s", buffer);

    // printf("Got output: %s\n", buffer);

} while (true);

```

```

        waitpid(pId1, NULL, 0);

        waitpid(pId2, NULL, 0);

        close(pipe1[1]);

        close(pipe2[0]);

        close(pipe3[1]);

        return 0;

}

```

ChildOne.c

```

#include <config/config.h>

#include <fcntl.h>

#include <log/log.h>

#include <stdarg.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

int main() {

    char buffer[LINE_BUFFER_LEN];

    int log_file = create_log_file("child1");

    log_to_file(log_file, "Dolbaeb is working, pid %d", getpid());

    ssize_t bytes = 0;

    while ((bytes = read(STDIN_FILENO, buffer, LINE_BUFFER_LEN)) != -1) {

        log_to_file(log_file, "Dolbaeb with pid %d got '%s', bytes read %d", getpid(),
buffer, bytes);

        if (buffer[0] == '\n') {

            log_to_file(log_file, "Dobaeb exitiing");

            write(STDOUT_FILENO, buffer, strlen(buffer));

```

```

        break;
    }

    for (int i = 0; buffer[i] != '\0'; i++) {
        char c = buffer[i];
        if (c >= 'A' && c <= 'Z') {
            buffer[i] += 'a' - 'A';
        }
    }

    if (write(STDOUT_FILENO, buffer, strlen(buffer)) == -1) {
        log_to_file(log_file, "Dolbaeb with pid %d got error during writing",
getpid(), buffer);
    }

    log_to_file(log_file, "Dolbaeb with pid %d wrote '%s'", getpid(), buffer);
}

write(log_file, "", 1);
close(log_file);
return 0;
}

```

ChildTwo.c

```
#include <config/config.h>
```

```
#include <log/log.h>
```

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

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

```
#include <string.h>
```

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

```

int main() {
    char buffer[LINE_BUFFER_LEN];
    char resultString[LINE_BUFFER_LEN];
    int log_file = create_log_file("child2");

```

```

log_to_file(log_file, "Dodik is working, pid %d", getpid());

ssize_t bytes = 0;

while ((bytes = read(STDIN_FILENO, buffer, LINE_BUFFER_LEN))) {

    log_to_file(log_file, "Dodik with pid %d got '%s', bytes read %d", getpid(), buffer,
bytes);

    if (buffer[0] == '\n') {

        log_to_file(log_file, "Dodik exiting");

        break;

    }

    int j = 0;

    for (int i = 0; buffer[i] != '\0'; i++) {

        if (!(buffer[i] == ' ' && buffer[i + 1] == ' ')) {

            resultString[j++] = buffer[i];

        }

    }

    resultString[j] = '\0';

    log_to_file(log_file, "Dodik with pid %d wrote '%s'", getpid(), resultString);

    write(STDOUT_FILENO, resultString, strlen(resultString));

}

write(log_file, "", 1);

close(log_file);

return 0;

}

```

Протокол работы программы

Тестирование

\$./a.out

TestTinGG

got: testingg

Strace

SYSCALL(args) = return

munmap(0x105398000, 0x8C000) = 0 0

munmap(0x105424000, 0x8000) = 0 0

munmap(0x10542C000, 0x4000) = 0 0

munmap(0x105430000, 0x4000) = 0 0


```

munmap(0x105434000, 0x50000)                = 0 0
crossarch_trap(0x0, 0x0, 0x0)                = -1 Err#45
open("./\0", 0x100000, 0x0)                  = 3 0
fcntl(0x3, 0x32, 0x16AE131C8)                = 0 0
close(0x3)                                    = 0 0
fsgetpath(0x16AE131D8, 0x400, 0x16AE131B8)    = 33 0
fsgetpath(0x16AE131E8, 0x400, 0x16AE131C8)    = 14 0
csrctl(0x0, 0x16AE135EC, 0x4)                = -1 Err#1
__mac_syscall(0x195135ACF, 0x2, 0x16AE13530)  = 0 0
csrctl(0x0, 0x16AE135DC, 0x4)                = -1 Err#1
__mac_syscall(0x195132902, 0x5A, 0x16AE13570)  = 0 0
= 0 0 sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16AE12AF8, 0x16AE12AF0, 0x195134553, 0xD)
= 0 0 sysctl([CTL_KERN, 140, 0, 0, 0, 0] (2), 0x16AE12BA8, 0x16AE12BA0, 0x0, 0x0)

open("/\0", 0x20100000, 0x0)                  = 3 0
openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0
dup(0x4, 0x0, 0x0)                            = 5 0
fstatat64(0x4, 0x16AE12681, 0x16AE125F0)      = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0
fcntl(0x6, 0x32, 0x16AE12680)                = 0 0
dup(0x6, 0x0, 0x0)                            = 7 0
dup(0x5, 0x0, 0x0)                            = 8 0
close(0x3)                                    = 0 0
close(0x5)                                    = 0 0
close(0x4)                                    = 0 0
close(0x6)                                    = 0 0
__mac_syscall(0x195135ACF, 0x2, 0x16AE13070)  = 0 0
shared_region_check_np(0x16AE12C90, 0x0, 0x0) = 0 0
fsgetpath(0x16AE131F0, 0x400, 0x16AE13138)    = 82 0
fcntl(0x8, 0x32, 0x16AE131F0)                = 0 0
close(0x8)                                    = 0 0
close(0x7)                                    = 0 0
getfsstat64(0x0, 0x0, 0x2)                    = 11 0
getfsstat64(0x1053FE090, 0x5D28, 0x2)        = 11 0
getattrlist("/\0", 0x16AE13120, 0x16AE13090)  = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm
64e\0", 0x16AE13480, 0x0) = 0 0
dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid
address (0x0) in action #11 at DIF offset 12
stat64("/Users/mk/OSlabs/OSlabs/src/main\0", 0x16AE12930, 0x0) = 0 0
open("/Users/mk/OSlabs/OSlabs/src/main\0", 0x0, 0x0) = 3 0
mmap(0x0, 0x8578, 0x1, 0x40002, 0x3, 0x0) = 0x105440000 0
fcntl(0x3, 0x32, 0x16AE12A48)                = 0 0
close(0x3)                                    = 0 0
munmap(0x105440000, 0x8578)                    = 0 0
stat64("/Users/mk/OSlabs/OSlabs/src/main\0", 0x16AE12EA0, 0x0) = 0 0
stat64("/usr/lib/libSystem.B.dylib\0", 0x16AE11DE0, 0x0) = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",
0x16AE11D90, 0x0) = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16AE0FA00, 0x0) = -1 Err#2

```

```

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0",
0x16AE0F9B0, 0x0) = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16AE0FA00, 0x0) = -1 Err#2
open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0
ioctl(0x3, 0x80086804, 0x16AE119D8) = 0 0
close(0x3) = 0 0
open("/Users/mk/OSlabs/OSlabs/src/main\0", 0x0, 0x0) = 3 0
__mac_syscall(0x195135ACF, 0x2, 0x16AE110E0) = 0 0
map_with_linking_np(0x16AE10F80, 0x1, 0x16AE10FB0) = 0 0
close(0x3) = 0 0
mprotect(0x104FF0000, 0x4000, 0x1) = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0
mprotect(0x1053FC000, 0x40000, 0x1) = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0) = -1 Err#2
bsdthread_register(0x195438D2C, 0x195438D20, 0x4000) = 1073746399 0
getpid(0x0, 0x0, 0x0) = 7664 0
shm_open(0x1952D3F51, 0x0, 0x69636573) = 3 0
fstat64(0x3, 0x16AE11D70, 0x0) = 0 0
mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0) = 0x105448000 0
close(0x3) = 0 0
ioctl(0x2, 0x4004667A, 0x16AE11E1C) = 0 0
mprotect(0x105454000, 0x4000, 0x0) = 0 0
mprotect(0x105460000, 0x4000, 0x0) = 0 0
mprotect(0x105464000, 0x4000, 0x0) = 0 0
mprotect(0x105470000, 0x4000, 0x0) = 0 0
mprotect(0x105474000, 0x4000, 0x0) = 0 0
mprotect(0x105480000, 0x4000, 0x0) = 0 0
mprotect(0x10544C000, 0xA0, 0x1) = 0 0
mprotect(0x10544C000, 0xA0, 0x3) = 0 0
mprotect(0x10544C000, 0xA0, 0x1) = 0 0
mprotect(0x105484000, 0x4000, 0x1) = 0 0
mprotect(0x105488000, 0xA0, 0x1) = 0 0
mprotect(0x105488000, 0xA0, 0x3) = 0 0
mprotect(0x105488000, 0xA0, 0x1) = 0 0
mprotect(0x10544C000, 0xA0, 0x3) = 0 0
mprotect(0x10544C000, 0xA0, 0x1) = 0 0
mprotect(0x105484000, 0x4000, 0x3) = 0 0
mprotect(0x105484000, 0x4000, 0x1) = 0 0
mprotect(0x1053FC000, 0x40000, 0x3) = 0 0
mprotect(0x1053FC000, 0x40000, 0x1) = 0 0
objc_bp_assist_cfg_np(0x195065000, 0x80000018001C1048, 0x0) = -1 Err#5
issetugid(0x0, 0x0, 0x0) = 0 0
mprotect(0x1053FC000, 0x40000, 0x3) = 0 0
getentropy(0x16AE11488, 0x20, 0x0) = 0 0
mprotect(0x1053FC000, 0x40000, 0x1) = 0 0
mprotect(0x1053FC000, 0x40000, 0x3) = 0 0
mprotect(0x1053FC000, 0x40000, 0x1) = 0 0
= 0 0 getattrlist("/Users/mk/OSlabs/OSlabs/src/main\0", 0x16AE11D00, 0x16AE11D18)
access("/Users/mk/OSlabs/OSlabs/src\0", 0x4, 0x0) = 0 0

```

```

open("/Users/mk/OSlabs/OSlabs/src\0", 0x0, 0x0)          = 3 0
fstat64(0x3, 0x14B6045B0, 0x0)                          = 0 0
csrctl(0x0, 0x16AE11F2C, 0x4)                            = 0 0
fcntl(0x3, 0x32, 0x16AE11BE8)                          = 0 0
close(0x3)                                                = 0 0
open("/Users/mk/OSlabs/OSlabs/src/Info.plist\0", 0x0, 0x0) = -1 Err#2
proc_info(0x2, 0x1DF0, 0xD)                             = 64 0
csops_audittoken(0x1DF0, 0x10, 0x16AE11F70)              = 0 0
= 0 0 sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16AE122C8, 0x16AE122C0, 0x19886AD3D, 0x15)
= 0 0 sysctl([CTL_KERN, 138, 0, 0, 0, 0] (2), 0x16AE12358, 0x16AE12350, 0x0, 0x0)
csops(0x1DF0, 0x0, 0x16AE123FC)                          = 0 0
mprotect(0x1053FC000, 0x40000, 0x3)                     = 0 0
pipe(0x0, 0x0, 0x0)                                       = 3 0
pipe(0x0, 0x0, 0x0)                                       = 5 0
pipe(0x0, 0x0, 0x0)                                       = 7 0
fork()                                                    = 7665 0
fork()                                                    = 7666 0
close(0x8)                                                = 0 0
close(0x5)                                                = 0 0

```

Вывод

Я изучил, как управлять процессами в ОС при помощи системных вызовов на языке Си. Изучил способы обмена данных между процессами, понял, что при вызове `exec`, программа полностью замещает текущий процесс. Данная лабораторная работа показалась мне интересной, т.к ранее я столько не мучался, пока пытался найти ошибку в коде. В ходе выполнения данной лабораторной работы я столкнулся с многими проблемами, но почти со всеми справился. Мне понравилось.