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-211Б-23

Студент: Косов В.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

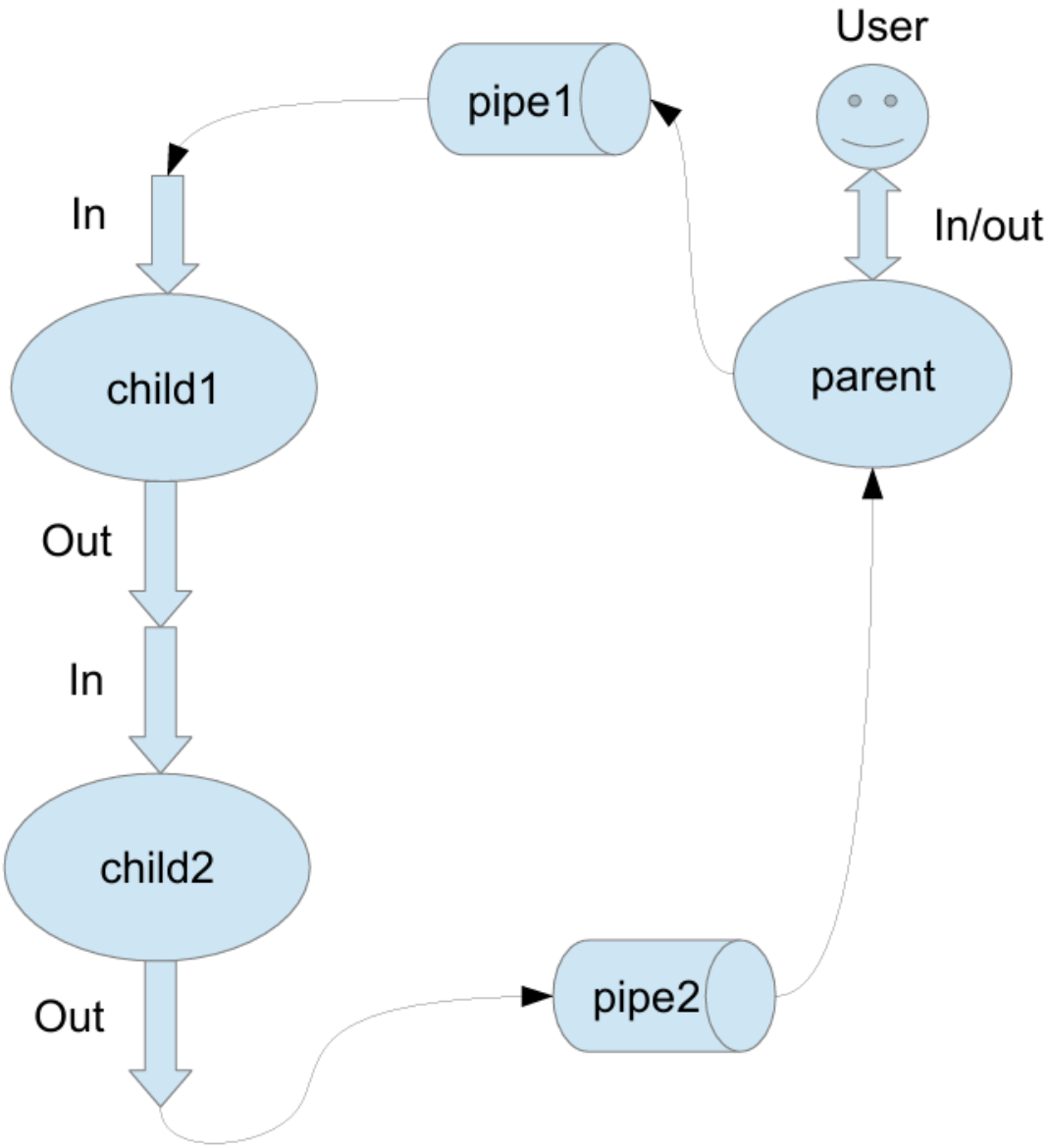
Дата: 03.12.24

Москва, 2024

Постановка задачи

Вариант 14.

Группа вариантов 3



14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

1. **pid_t fork(void);** – создает дочерний процесс.
2. **int pipe(int *fd);** – создает канал (pipe) для межпроцессного взаимодействия.
3. **int dup2(int oldfd, int newfd);** – дуплицирует файловый дескриптор.
4. **int execl(const char *path, const char *arg, ...);** – заменяет текущий процесс новым процессом.
5. **int close(int fd);** – закрывает файловый дескриптор.

6. **ssize_t read(int fd, void *buf, size_t count);** – читает данные из файлового дескриптора.
7. **ssize_t write(int fd, const void *buf, size_t count);** – записывает данные в файловый дескриптор.
8. **pid_t waitpid(pid_t pid, int *status, int options);** – ожидает завершения дочернего процесса.

Описание лабораторной работы

В рамках лабораторной работы была разработана программа, которая демонстрирует межпроцессное взаимодействие с использованием каналов (pipes) и системных вызовов. Программа состоит из родительского процесса и двух дочерних процессов, которые обрабатывают данные последовательно.

Цель лабораторной работы

Целью лабораторной работы было изучение и применение системных вызовов для создания и управления процессами, а также использование каналов для межпроцессного взаимодействия.

Описание программы

Программа состоит из трех частей:

1. **Родительский процесс (parent.c)**: Считывает данные из стандартного ввода (stdin), передает их первому дочернему процессу через канал. Получив результат работы второго дочернего процесса, записывает результат в стадартный поток вывода (stdout)
2. **Первый дочерний процесс (child1.c)**: Считывает данные из стандартного ввода (stdin), преобразует их в нижний регистр и записывает результат в именованный поток childrens_pipe.
3. **Второй дочерний процесс (child2.c)**: Считывает данные из именнованого потока childrens_pipe, убирает задвоенные пробелы и передает данные через канал родительскому процессу.

Код программы

parent.c

```
#include <stdint.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

typedef enum{
    OK,
    MemoryError
}status_code;
```

```
status_code get_string(char **string, int *len_string) {
    char c;
    *len_string = 0;
    int capacity_string = 2;
    *string = (char *)malloc(capacity_string * sizeof(char));
    if (*string == NULL) {
        return MemoryError;
    }

    do {
        c = getchar();
        if (c == EOF || c == '\n') break;
        (*string)[*len_string] = c;
        (*len_string)++;

        if (*len_string == capacity_string) {
            capacity_string *= 2;
            char *tmp_string = (char *)realloc(*string, capacity_string *
sizeof(char));
            if (tmp_string == NULL) {
                free(*string);
                *string = NULL;
                return MemoryError;
            }
            *string = tmp_string;
        }
    } while (c != EOF && c != '\n');

    (*string)[*len_string] = '\0';
    return OK;
}


int main(){
    const char* args[] = {
        "./child.",
        NULL
    };

    int len_string;
    char *string;
    if (get_string(&string, &len_string) == MemoryError){
        const char message[] = "MemoryError: Failed to allocate memory";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    int pipe_between_child1[2]; int pipe_between_child2[2];
    pipe(pipe_between_child1); pipe(pipe_between_child2);

    int pid1 = fork();
    if (pid1 == -1){
        const char message[] = "ProcessError: Failed to create a new process";
        write(STDERR_FILENO, message, sizeof(message));
```

```
        exit(EXIT_FAILURE);
    }
    else if (pid1 == 0){
        dup2(pipe_between_child1[0], STDIN_FILENO);
        execv("./child1", (char **) args);
    }
    else{
        write(pipe_between_child1[1], &len_string, sizeof(int));
        write(pipe_between_child1[1], string, sizeof(char) * len_string);
        free(string); string = NULL;
    }

    int pid2 = fork();
    if (pid2 == -1){
        const char message[] = "ProcessError: Failed to create a new process";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }
    else if (pid2 == 0){
        dup2(pipe_between_child2[1], STDOUT_FILENO);
        execv("./child2", (char**) args);
    }
    else{
        read(pipe_between_child2[0], &len_string, sizeof(int));
        string = (char *)malloc(len_string * sizeof(char));
        if (string == NULL){
            const char message[] = "MemoryError: Failed to allocate memory";
            write(STDERR_FILENO, message, sizeof(message));
            exit(EXIT_FAILURE);
        }
        read(pipe_between_child2[0], string, len_string * sizeof(char));
        write(STDOUT_FILENO, string, len_string * sizeof(char));
        free(string); string = NULL;

        int child_status;
                wait(&child_status);

                if (child_status != EXIT_SUCCESS) {
                        const char msg[] = "Error: Child exited with error\n";
                        write(STDERR_FILENO, msg, sizeof(msg));
                        exit(child_status);
        }
    }
    return 0;
}
```

child1.c

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

```
#include <stdbool.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

void to_lower(char *string, int len_string) {
    for (int i = 0; i < len_string; ++i) {
        if (string[i] >= 'A' && string[i] <= 'Z') {
            string[i] += 32;
        }
    }
}

int main() {
    int fd = open("childrens_pipe", O_WRONLY);
    if (fd == -1) {
        const char message[] = "OpenFileError: Failed to open file fifo";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    int len_string;
    if (read(STDIN_FILENO, &len_string, sizeof(int)) != sizeof(int)) {
        const char message[] = "ReadFileError: Failed to read length from stdin";
        write(STDERR_FILENO, message, sizeof(message));
        close(fd);
        exit(EXIT_FAILURE);
    }

    char *string = (char *)malloc(len_string * sizeof(char));
    if (string == NULL) {
        const char message[] = "MemoryError: Failed to allocate memory";
        write(STDERR_FILENO, message, sizeof(message));
        close(fd);
        exit(EXIT_FAILURE);
    }

    if (read(STDIN_FILENO, string, len_string) != len_string) {
        const char message[] = "ReadFileError: Failed to read string from stdin";
        write(STDERR_FILENO, message, sizeof(message));
        free(string);
        close(fd);
        exit(EXIT_FAILURE);
    }

    to_lower(string, len_string);

    if (write(fd, &len_string, sizeof(int)) == -1 || write(fd, string, len_string) ==
-1) {
        const char message[] = "WriteFileError: Failed to write in fifo";
        write(STDERR_FILENO, message, sizeof(message));
```

```
        free(string);
        close(fd);
        exit(EXIT_FAILURE);
    }

    close(fd);
    free(string);
    return 0;
}
```

child2.c

```
#include <stdint.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void remove_extra_spaces(char *str, int *length) {
    int i = 0, j = 0;
    int in_space = 0;

    while (str[i] != '\0') {
        if (str[i] == ' ') {
            if (!in_space && i != 0) {
                str[j++] = ' ';
                in_space = 1;
            }
        } else {
            str[j++] = str[i];
            in_space = 0;
        }
        i++;
    }

    str[j] = '\0';
    *length = j;
}

int main() {
    int fd = open("childrens_pipe", O_RDONLY);
    if (fd == -1) {
        const char message[] = "OpenFileError: Failed to open file fifo";
```

```
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    int len_string;
    if (read(fd, &len_string, sizeof(int)) != sizeof(int)) {
        const char message[] = "ReadFileError: Failed to read length from fifo";
        write(STDERR_FILENO, message, sizeof(message));
        close(fd);
        exit(EXIT_FAILURE);
    }

    char *string = (char *)malloc(len_string * sizeof(char));
    if (string == NULL) {
        const char message[] = "MemoryError: Failed to allocate memory";
        write(STDERR_FILENO, message, sizeof(message));
        close(fd);
        exit(EXIT_FAILURE);
    }

    if (read(fd, string, len_string) != len_string) {
        const char message[] = "ReadFileError: Failed to read string from fifo";
        write(STDERR_FILENO, message, sizeof(message));
        free(string);
        close(fd);
        exit(EXIT_FAILURE);
    }

    remove_extra_spaces(string, &len_string);

    string[len_string] = '\n';
    len_string++;

    if (write(STDOUT_FILENO, &len_string, sizeof(int)) == -1 ||
        write(STDOUT_FILENO, string, len_string) == -1) {
        const char message[] = "WriteFileError: Failed to write result to stdout";
        write(STDERR_FILENO, message, sizeof(message));
        free(string);
        close(fd);
        exit(EXIT_FAILURE);
    }

    free(string);
    close(fd);
    return 0;
}}
```

Протокол работы программы

```
vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_1$ ./parent

Hello            WORLD

hello world

vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_1$ strace ./parent

execve("./parent", ["./parent"], 0x7ffd14a08290 /* 35 vars */) = 0

brk(NULL)                                = 0x5627509a3000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff5f665880) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f508bd00000

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=21167, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 21167, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f508bcfa000

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\0P\237\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 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848)
= 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68,
896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

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

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f508bad1000

mprotect(0x7f508baf9000, 2023424, PROT_NONE) = 0

mmap(0x7f508baf9000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f508baf9000

mmap(0x7f508bc8e000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f508bc8e000

mmap(0x7f508bce7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f508bce7000
```

```
mmap(0x7f508bced000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f508bced000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f508bace000

arch_prctl(ARCH_SET_FS, 0x7f508bace740) = 0

set_tid_address(0x7f508bacea10)         = 46875

set_robust_list(0x7f508bacea20, 24)     = 0

rseq(0x7f508bacf0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f508bce7000, 16384, PROT_READ) = 0

mprotect(0x562723797000, 4096, PROT_READ) = 0

mprotect(0x7f508bd3a000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f508bcfa000, 21167)           = 0

getrandom("\xde\xd9\xfb\xe6\xaf\x92\x8d\xe1", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x5627509a3000

brk(0x5627509c4000)                     = 0x5627509c4000

newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x6), ...},
AT_EMPTY_PATH) = 0

read(0, Hello     WORLD

"Hello     WORLD\n", 1024)      = 16

pipe2([3, 4], 0)                        = 0

pipe2([5, 6], 0)                        = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f508bacea10) = 46964

write(4, "\17\0\0\0", 4)                = 4

write(4, "Hello     WORLD", 15)         = 15

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f508bacea10) = 46965

read(5, "\f\0\0\0", 4)                  = 4

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=46964, si_uid=1000,
si_status=0, si_utime=0, si_stime=1} ---

read(5, "hello world\n", 12)            = 12

write(1, "hello world\n", 12hello world

)           = 12
```

```
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 46964
exit_group(0)                           = ?
+++ exited with 0 +++
vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_1$
```

Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая межпроцессное взаимодействие с использованием каналов (pipes) и системных вызовов. Программа состоит из родительского процесса и двух дочерних процессов, которые обрабатывают данные последовательно. Родительский процесс считывает данные из стандартного ввода, передает их первому дочернему процессу, который преобразует их в нижний регистр, затем передает обработанные данные второму дочернему процессу, который заменяет удаляет задвоенные пробелы, и передает данные родитескому процессу через канал, после чего родительский канал записывает результат в стандартный поток вывода.