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.

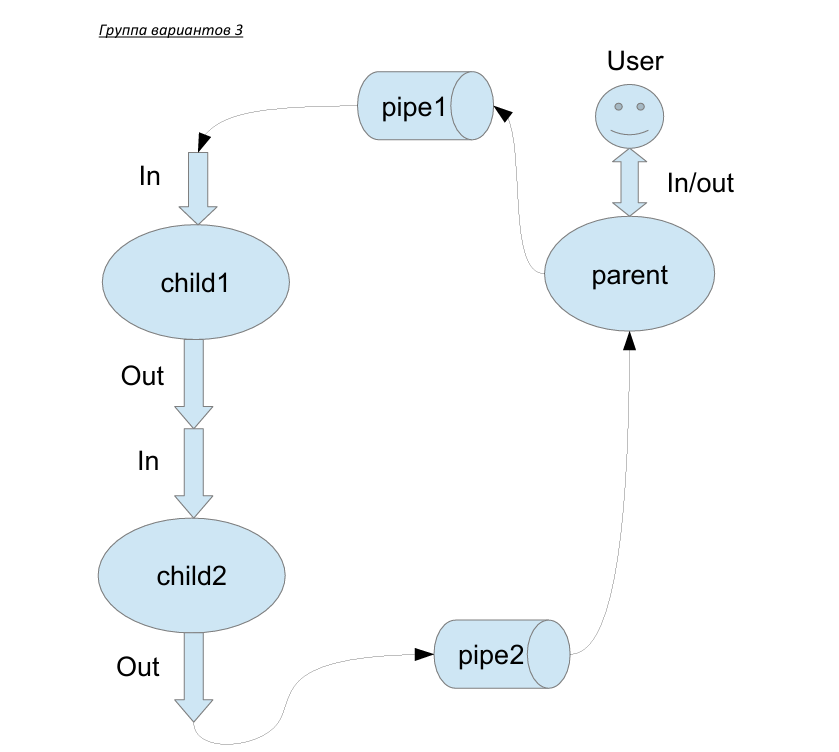
Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 03.12.24

Москва, 2024

**Постановка задачи**

**Вариант 14.**

****

**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 образ текущего процесса на образ нового процесса, определенного в пути path
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.