



Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Московский государственный технический университет имени  
Н.Э. Баумана (национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

## ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине «Разработка программных систем»

|              |                                  |
|--------------|----------------------------------|
| Студент:     | Кильдишев Петр Степанович        |
| Группа:      | РК6-66Б                          |
| Тип задания: | лабораторная работа              |
| Тема:        | Многопроцессное программирование |
| Вариант:     | 11                               |

Студент

\_\_\_\_\_  
подпись, дата

Кильдишев П.С.  
Фамилия, И.О.

Преподаватель

\_\_\_\_\_  
подпись, дата

Козов А.В.  
Фамилия, И.О.

Москва, 2023

## Содержание

|   |    |
|---|----|
| Задание . . . . .   | 3  |
| Описание структуры программы и реализованных способов взаимодействия<br>процессов . . . . . | 5  |
| Описание основных используемых структур данных . . . . .                                    | 6  |
| Блок-схема . . . . .  | 7  |
| Примеры работы программы . . . . .  | 11 |
| Текст программы . . . . .   | 13 |

## Задание

Разработать программу, реализующую обработку строк текста и функционирующую в рамках 3-х процессов.

Процесс 0 (корневой) порождает 2-ух потомков (напрямую или опосредованно, решать вам) и органиует конвейер (pipe) передачи строк символов в последовательности "процесс-0 => процесс-1 => процесс-2".

Процесс 0 получает строку символов латинского алфавита со стандартного ввода и передаёт ее в неизменном виде процессу 1.

Процесс 1 умеет преобразовывать полученную от процесса 0 строку символов различными способами:

- трансляция строки в неизменном виде;
- инвертирование строки - первый символ становится последним и т.д.;
- обмен соседних символов - нечетный становится на место четного и наоборот;
- перевод в КОИ-8 - установление в 1 старшего (8-ого) бита каждого символа.

Процесс 2 умеет преобразовывать полученную от процесса 1 строку символов следующими способами:

- трансляция строки в неизменном виде;
- перевод всех символов строки в "верхний регистр";
- перевод всех символов строки в "нижний регистр";
- смена "регистра" всех символов строки.

Основная функциональность процесса 0 - считывание со стандартного ввода строк латинского текста и передача их (строк) процессу 1. Однако сигнал SIGINT (Ctrl/C) переводит процесс 0 в состояние **однократного** чтения со стандартного ввода команды перехода процесса 1 или 2 к новому способу обработки поступающих к нему строк

текста (синтаксис команд придумать самостоятельно).

Обязательно обеспечить индикацию результатов работы каждого процесса в правильной временной последовательности.

## Описание структуры программы и реализованных способов взаимодействия процессов

Для решения поставленной задачи реализованы 3 процесса, как указано в задании.

Процесс-родитель 0 создает 2 процесса-потомка и организует структуру общения между процессами. Процесс 0 также циклично считывает строки текста из потока стандартного ввода при помощи функции **read** до тех пор, пока не будет получен символ завершения ввода или процесс не будет прерван. При этом происходит проверка на то, что символы полученной строки являются либо текстовыми символами, либо цифрами, либо знаками припинания, либо символом ". Дополнительно процесс 0 имеет возможность менять режимы работы процессов 1 и 2 при получении сигнала **SIGINT**.

Работа со строкой в процессе 0 выполняется в функции **proc0()**.

При получении сигнала **SIGINT** в процессе 0 вызывается пользовательская функция **set\_mode()**. Функция спрашивает, в каком из процессов требуется выбор режима (1 или 2), и какой из режимов выбирается (с 0-го по 3-й). При каждом запросе требуется ввести одну цифру и совершить ввод клавишей Enter. На любой другой ввод будет получен ответ в виде ошибки. Изменения режимов процессов записываются в массив строк процесса 0, хранящий режимы процессов.

Для общения процессы используют сигнальный механизм и 2 канала: **pipe01** - для общения между родительским процессом 0 и процессом 1; **pipe12** - для общения между процессом 1 и процессом 2. По каналам в первых 2-х байтах передается режим работы процессов. В последующих байтах передается текст, полученный в результате работы предыдущего процесса. Взаимодействие процессов представлено на рисунке 1.

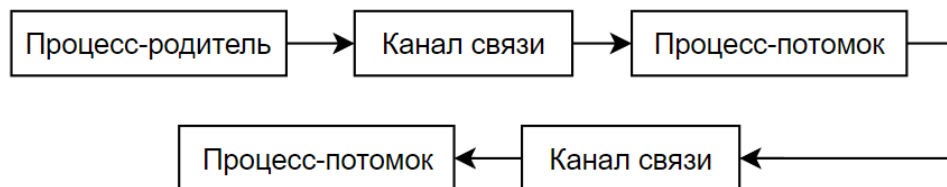


Рис. 1. Взаимодействие процессов

Процессы 1 и 2 циклично приостанавливают свою работу вызовом **pause()** до тех пор, пока не будет получен сигнал **SIGUSR1** и **SIGUSR2** соответственно. При получении сигнала процессы выполняют функции **proc1()** и **proc2()** соответственно, выполняющие описанную в задании логику. Сигнал **SIGUSR1** отправляется процессом 0 процессу 1 при считывании очередной строки. Сигнал **SIGUSR2** отправляется процессом 1 процессу 2 при завершении обработки полученной строки. Процессу 1 известен PID процесса 2 по той причине, что он создается позднее процесса 2. В процессах 1 и 2 также включено игнорирование сигнала **SIGINT**, так как его вызов не должен влиять на работу процессов.

При завершении работы каждого процесса на одной итерации выводится сообщение, демонстрирующее результат работы данного процесса.

Работа процессов 1 и 2 реализована в функциях **proc1()** и **proc2()** соответственно.

## Описание основных используемых структур данных

Для хранения режимов процессов используется массив строк длиной в 1 символ **proc\_mode**, состоящий из двух элементов:

- **proc\_mode[0]** - режим процесса 1,
- **proc\_mode[1]** - режим процесса 2.

Тип данных массива выбран для удобства считывания функцией **write**.

Для хранения файловых дескрипторов используется целочисленные массивы **fds01** и **fds12**, состоящие из двух элементов:

- **fds01[0]** и **fds12[0]** - для чтения,
- **fds01[1]** и **fds12[1]** - для записи.

## Блок-схема

На рисунке 2 представлена блок-схема программы. На рисунках 3, 4, 5 представлены блок-схемы работы функций, вызываемых в процессе работы программы.

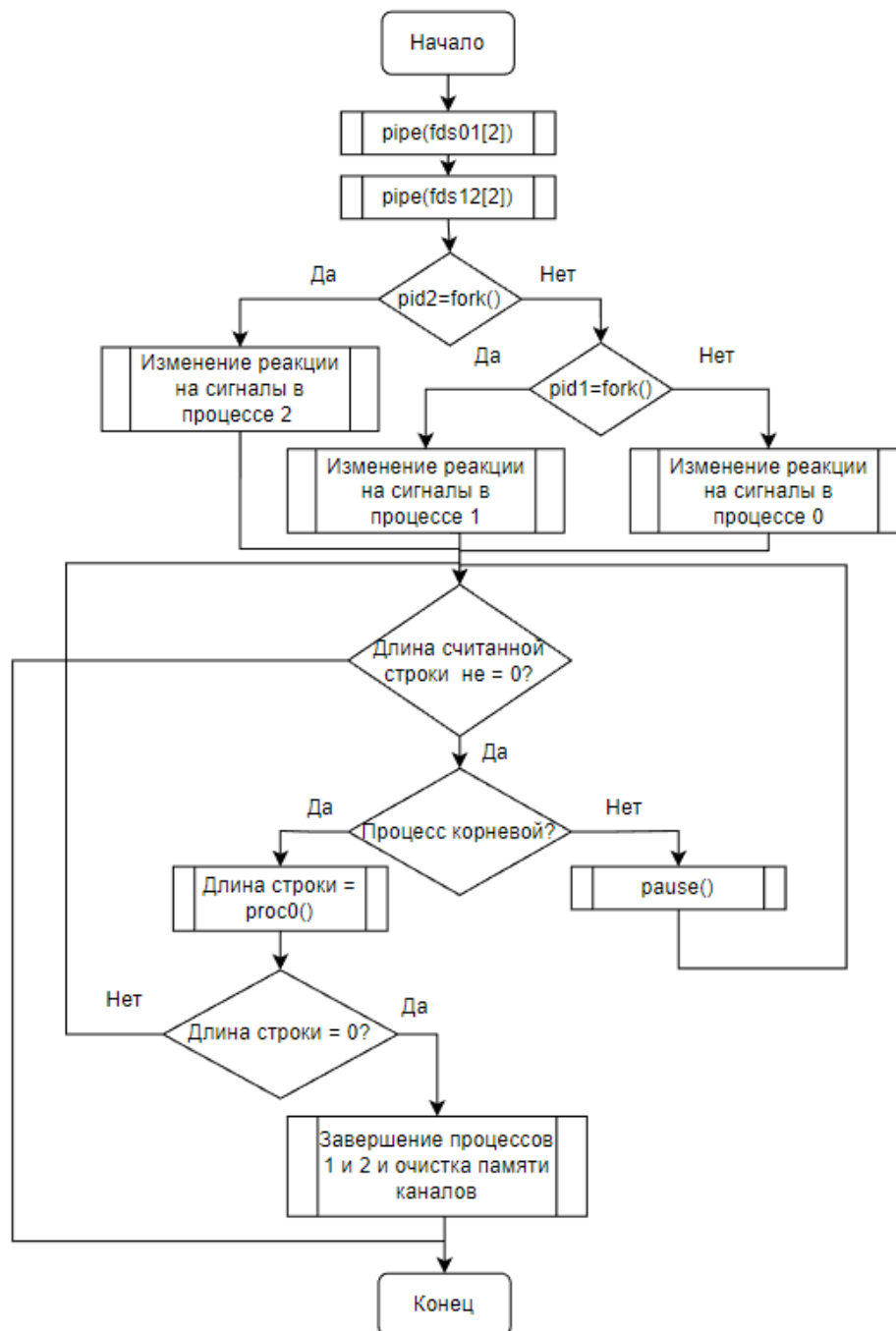


Рис. 2. Блок схема работы программы

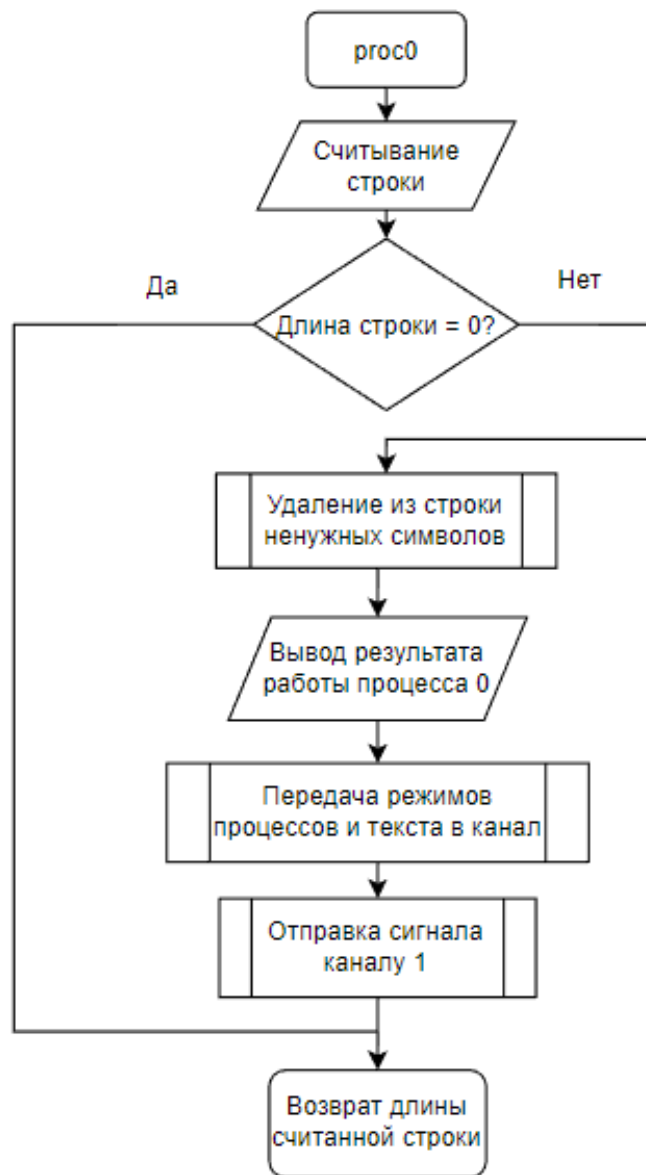


Рис. 3. Блок схема работы `rgos0()`



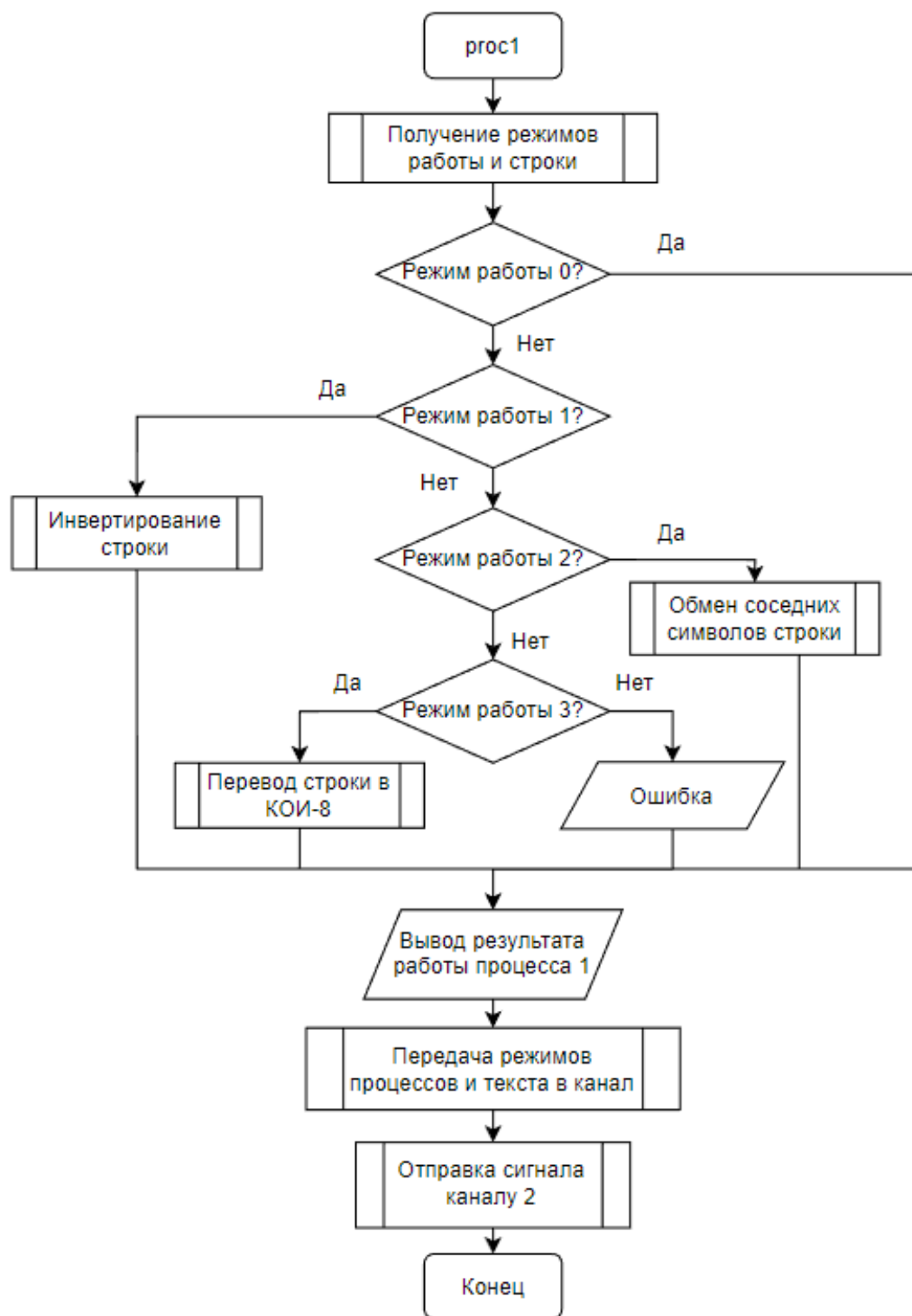


Рис. 4. Блок схема работы rpos1()

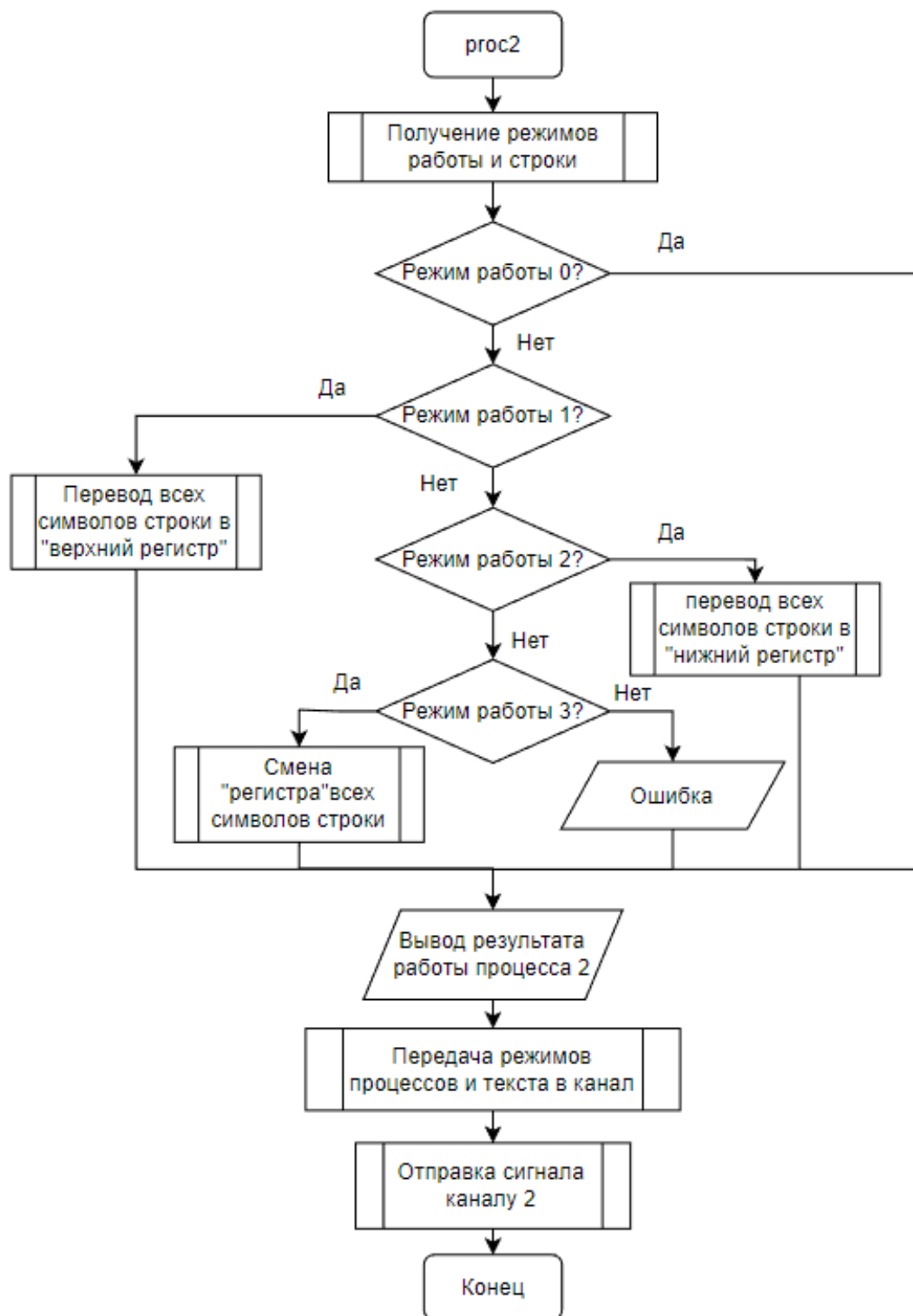
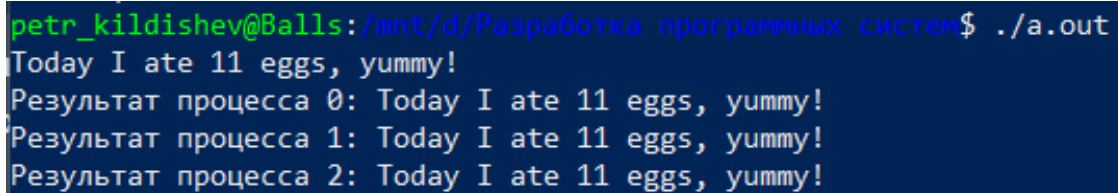


Рис. 5. Блок схема работы proc2()

## Примеры работы программы

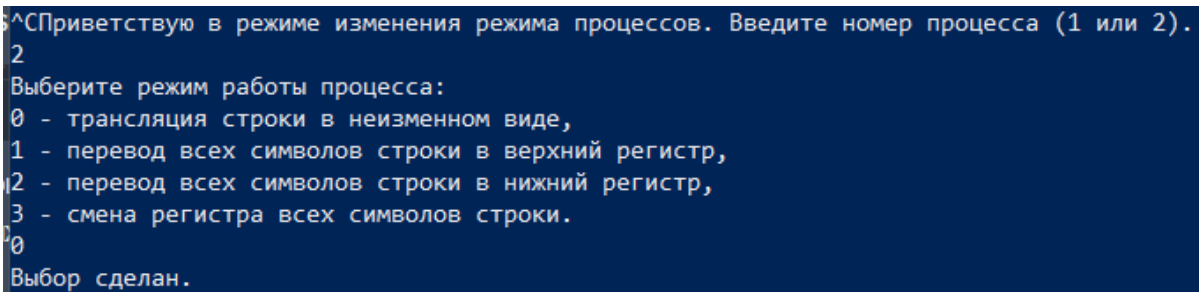
На рисунке 6 приведен пример корректной работы программы в режиме по умолчанию. На вход дается строка текста и ретранслируется в последующие процессы:



```
petr_kildishev@Balls:/mnt/d/Разработка программных систем$ ./a.out
Today I ate 11 eggs, yummy!
Результат процесса 0: Today I ate 11 eggs, yummy!
Результат процесса 1: Today I ate 11 eggs, yummy!
Результат процесса 2: Today I ate 11 eggs, yummy!
```

Рис. 6. Пример работы программы №1

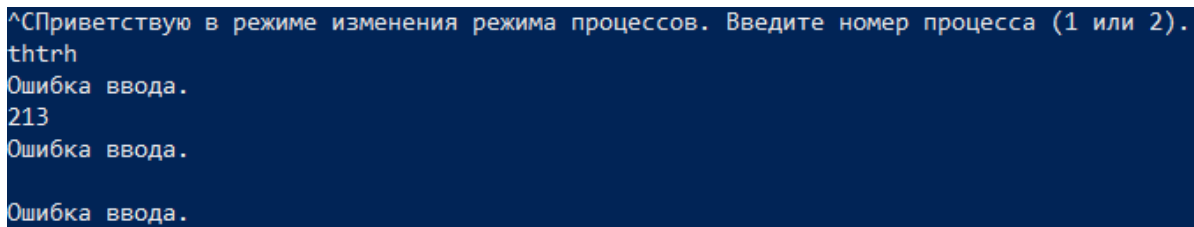
На рисунке 7 приведен пример корректного изменения режима процесса:



```
^СПриветствую в режиме изменения режима процессов. Введите номер процесса (1 или 2).
2
Выберите режим работы процесса:
0 - трансляция строки в неизменном виде,
1 - перевод всех символов строки в верхний регистр,
2 - перевод всех символов строки в нижний регистр,
3 - смена регистра всех символов строки.
0
Выбор сделан.
```

Рис. 7. Пример работы программы №2

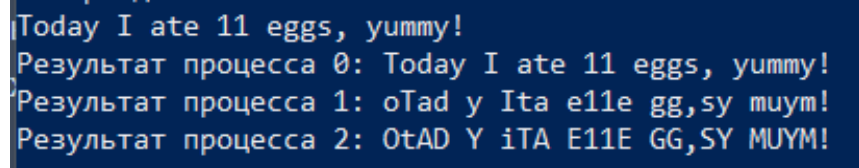
На рисунке 8 приведен пример попытки не корректного ввода при изменении режима процесса:



```
^СПриветствую в режиме изменения режима процессов. Введите номер процесса (1 или 2).
thtrh
Ошибка ввода.
213
Ошибка ввода.
Ошибка ввода.
```

Рис. 8. Пример работы программы №3

На рисунке 9 приведен пример корректной работы программы в с измененными режимами работы процессов. Режим процесса 1 - обмен соседних символов, процесса 2 - смена регистра всех символов строки.



```
Today I ate 11 eggs, yummy!  
Результат процесса 0: Today I ate 11 eggs, yummy!  
Результат процесса 1: oTad y Ita e11e gg,sy muym!  
Результат процесса 2: OtAD Y iTA E11E GG,SY MUYM!
```

Рис. 9. Пример работы программы №1

## Текст программы

Ниже в листинге 1 представлен текст программы.

Листинг 1. Листинг программы

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <signal.h>
5 #include <ctype.h>
6
7 const char* INPUT_ERROR = "Ошибка ввода.\n";
8 const int MAX_N = 10000;
9 char proc_mode[2][2];
10 pid_t pid1, pid2;
11 int fds01[2], fds12[2];
12
13 void set_mode(int pid1, int pid2)
14 {
15     char proc[2], mode[2], c;
16     const char* WELCOME = "Приветствую в режиме изменения режима процессов.
        Введите номер процесса (1 или 2).\n";
17     const char* CHOOSE_MODE_MES = "Выберите режим работы процесса:\n";
18     const char* PROC_MODES[2] = {
19 "0 — трансляция строки в неизменном виде, \n1 — инвертирование строки —
        первый символ становится последним и т.д., \n2 — обмен соседних символов —
        нечетный становится на место четного и наоборот, \n3 — перевод в КОИ-8 —
        установление в 1 старшего (8-го) бита каждого символа.\n",
20 "0 — трансляция строки в неизменном виде, \n1 —
        перевод всех символов строки в верхний регистр, \n2 —
        перевод всех символов строки в нижний регистр, \n3 — смена регистра всех символов строки
        .\n"};
21     const char* FINISH = "Выбор сделан.\n";
22
23     write(1, WELCOME, strlen(WELCOME));
24     while (read(0, &proc, MAX_N) != 2 (mode[0] != '0' && proc[0] != '1' && proc[0] !=
        '2') proc[1] != '\n')
25     {
26         write(1, INPUT_ERROR, strlen(INPUT_ERROR));
27     }
28     write(1, CHOOSE_MODE_MES, strlen(CHOOSE_MODE_MES));
29     write(1, PROC_MODES[proc[0] - '1'], strlen(PROC_MODES[proc[0] - '1']));
30
31     while (read(0, &mode, MAX_N) != 2 (mode[0] != '0' && mode[0] != '1' && mode[0]
        != '2' && mode[0] != '3') mode[1] != '\n')
```

```

32     {
33         write(1, INPUT_ERROR, strlen(INPUT_ERROR));
34     }
35     proc_mode[proc[0] - '1'][0] = mode[0];
36
37     write(1, FINISH, strlen(FINISH));
38 }
39
40 int isprep(char c)
41 {
42     char prep[29] = {'!', '%', '^', '&', '*', '(', ')', '-', '+',
43 '=, '{', '}', 124, '~', '[', ']', 92, 59, 39, ':', 34, '<', '>', '?', ',', '.', '/', '#', ' '};
44     for (int i = 0; i < 29; i++)
45         if (c == prep[i])
46             return 1;
47     return 0;
48 }
49
50 int proc0()
51 {
52     char str[MAX_N], clear_str[MAX_N], mode_str[MAX_N];
53     const char *MSG = "Результат процесса0: ";
54
55     int str_len = 0, clear_str_len = 0;
56
57     for (int i = strlen(str); i > -1; i--)
58         str[i] = '\0';
59     for (int i = strlen(mode_str); i > -1; i--)
60         mode_str[i] = '\0';
61
62     if ((str_len = read(0, &str, MAX_N)) == 0)
63         return 0;
64
65     for (int i = 0; i < str_len; i++)
66     {
67         if (isprep(str[i]) str[i] == '\n' isalnum(str[i]))
68             clear_str[clear_str_len++] = str[i];
69     }
70
71     write(1, MSG, strlen(MSG));
72     write(1, clear_str, clear_str_len);
73
74     strcat(mode_str, proc_mode[0]);
75     strcat(mode_str, proc_mode[1]);
76     strcat(mode_str, clear_str);
77

```

```

78     write(fds01[1], mode_str, str_len + 2);
79
80     kill(pid1, SIGUSR1);
81
82     return str_len;
83 }
84
85 void proc1()
86 {
87     const char *MSG = "Результат процесса1: ";
88     char buf[MAX_N];
89
90     for (int i = strlen(buf); i > -1; i--)
91         buf[i] = '\0';
92
93     read(fds01[0], &proc_mode[0], 1);
94     read(fds01[0], &proc_mode[1], 1);
95     int len_buf = read(fds01[0], &buf, MAX_N);
96
97     char new_buf[len_buf], new_buf_mode[len_buf + 1];
98
99     for (int i = strlen(new_buf); i > -1; i--)
100         new_buf[i] = '\0';
101     for (int i = strlen(new_buf_mode); i > -1; i--)
102         new_buf_mode[i] = '\0';
103
104     switch (proc_mode[0][0])
105     {
106     case '0':
107         strcpy(new_buf, buf);
108         break;
109     case '1':
110         for (int i = 0; i < len_buf - 1; i++)
111             new_buf[i] = buf[len_buf - 2 - i];
112         new_buf[len_buf - 1] = '\n';
113         break;
114     case '2':
115         for (int i = 0; i < len_buf; i++)
116             new_buf[i] = buf[i + (-1) * (i % 2 == 1) * (i != (len_buf - 1)) + (i
117 % 2 == 0) * (i < (len_buf - 2))];
118         break;
119     case '3':
120         for (int i = 0; i < len_buf; i++)
121             // new_buf[i] = buf[i] | 256;
122             new_buf[i] = buf[i] + ((buf[i] & 256) == 0) * 256; //Версия,
123                                     компилируемая в Latex

```

```

122         break;
123     default:
124         write(1, "Error1\n", 7);
125         break;
126     }
127     new_buf_mode[0] = proc_mode[1][0];
128     strcat(new_buf_mode, new_buf);
129
130     write(1, MSG, strlen(MSG));
131     write(1, new_buf, len_buf);
132
133     write(fds12[1], new_buf_mode, len_buf + 1);
134     kill(pid2, SIGUSR2);
135 }
136
137 void proc2()
138 {
139     const char *MSG = "Результат процесса2: ";
140     char buf[MAX_N];
141
142     for (int i = strlen(buf); i > -1; i--)
143         buf[i] = '\0';
144
145     read(fds12[0], &proc_mode[1], 1);
146     int len_buf = read(fds12[0], &buf, MAX_N);
147
148     char new_buf[len_buf];
149     switch (proc_mode[1][0])
150     {
151     case '0':
152         strcpy(new_buf, buf);
153         break;
154     case '1':
155         for (int i = 0; i < len_buf; i++)
156             new_buf[i] = toupper(buf[i]);
157         break;
158     case '2':
159         for (int i = 0; i < len_buf; i++)
160             new_buf[i] = tolower(buf[i]);
161         break;
162     case '3':
163         for (int i = 0; i < len_buf; i++)
164             if (isupper(buf[i]))
165                 new_buf[i] = tolower(buf[i]);
166             else
167                 new_buf[i] = toupper(buf[i]);

```



```

168         break;
169     default:
170         write(1, "Error2\n", 7);
171         break;
172     }
173     write(1, MSG, strlen(MSG));
174     write(1, new_buf, len_buf);
175 }
176
177 int main(int argc, char** argv)
178 {
179     int str_len = 1;
180
181     pipe(fds01);
182     pipe(fds12);
183
184     proc_mode[0][0] = '0';
185     proc_mode[0][1] = '\0';
186     proc_mode[1][0] = '0';
187     proc_mode[1][1] = '\0';
188
189     if ((pid2 = fork()) == 0)
190     {
191         signal(SIGUSR2, proc2);
192         signal(SIGINT, SIG_IGN);
193     }
194     else if ((pid1 = fork()) == 0)
195     {
196         signal(SIGUSR1, proc1);
197         signal(SIGINT, SIG_IGN);
198     }
199     else if (pid1 != 0 && pid2 != 0)
200         signal(SIGINT, set_mode);
201
202     while (str_len != 0)
203     {
204         if (pid1 && pid2)
205         {
206             str_len = proc0();
207             if (str_len == 0)
208             {
209                 kill(pid1, SIGTERM);
210                 kill(pid2, SIGTERM);
211                 close(fds01[1]);
212                 close(fds01[0]);
213                 close(fds12[1]);

```

```
214         close(fds12[0]);
215         exit(0);
216     }
217 }
218 else
219     pause();
220 }
221 exit(0);
222 }
```

---