Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5
за IV семестр
по дисциплине: "Операционные системы и системное программирование"
Тема: "Ввод/вывод"

Выполнил:
студент 2 курса
факультета ЭИС
группы ПО-4 (1)
Галанин П. И. (зачётка №190333)

Проверил:
ст. преподаватель
Давидюк Ю. И.

Брест 2021

Лабораторная работа №5

Тема: "Ввод/вывод".

Цель:

Ход работы:

Часть II Вариант 5, 15

Условие: Написать программу, которая получает со стандартного потока ввода список активных процессов, и выводит в стандартный поток вывода процессы только с четными PID, добавив к имени процессов любое число. Протестировать с использованием конвейеров в различных комбинациях вашей программы и команд ps, sort, head, tail.

Решение:

Листинг: Makefile

```
all:
    mkdir -p bin
    cd bin; gcc ./../src/main.c -o app

clean:
    rm -rfv bin

run:
    cd bin; ./app

test1:
    cd bin; ps | ./app

test2:
    cd bin; ps | sort | ./app

test3:
    cd bin; ps | head -n 4 | ./app

test4:
    cd bin; ps | tail -n 4 | ./app
```

Листинг: main.c

```c
// библиотеки

#include <stdio.h>  //printf()
#include <unistd.h> //read()
#include <stdlib.h> //calloc(), free(), realloc(), rand()
#include <string.h> //strcmp(), strlen(), atoi(), strcat()

// прототипы

struct WordNode
{
    char* word;
    struct WordNode* left;
    struct WordNode* right;
};
struct WordNode* WordNode__Constructor(struct WordNode* object, char* word);
struct WordNode* WordNode__Destructor(struct WordNode* object);
struct WordNode* WordNode__add_top(struct WordNode* object, char* word);
struct WordNode* WordNode__get_bottom(struct WordNode* object);
void WordNode__set_word(struct WordNode* object, char* word);
int get_depth_long_int(long int N);
char* get_string_with_number_hash(char* str, long int number);

// главная программа

int main()
{
    struct WordNode* string_list = NULL;
    struct WordNode* result = NULL;

    int file_descriptor = 0;
    const int buffer_size = 1;
    char buffer[buffer_size];

    int str_size = 0;
    char* str = (char*) calloc(str_size, sizeof(char*));
    if (str == NULL) printf("Память не выделилась\n");
    while(read(file_descriptor, buffer, buffer_size))
    {
        char character = buffer[0];
        if (character == '\n')
        {
            string_list = WordNode__add_top(string_list, str);

            str_size = 0;
            str = (char*) realloc(str, str_size * sizeof(char));

            continue;
        }
        str_size += 1;
        str = (char*) realloc(str, str_size * sizeof(char));
        str[str_size - 1] = character;
    }
    free(str);

    printf(" ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~\n");
    for(struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
    temp->right)
```

```
58      {
59          printf("%s\n", temp->word);
60      }
61
62      for(struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
        temp->right)
63      {
64          int str_size = strlen(temp->word);
65          char* str = (char*) calloc(str_size, sizeof(char));
66          if (str == NULL) printf("Память не выделилась\n");
67          strcpy(str, temp->word);
68
69          int a = 0, b = 1234567890;
70          int rand_number = rand() % (b - a + 1) + a;
71          str = get_string_with_number_hash(str, rand_number);
72
73          WordNode__set_word(temp, str);
74          free(str);
75      }
76
77      for (struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
        temp->right)
78      {
79          str_size = 0;
80          str = (char*) calloc(str_size, sizeof(char));
81          if (str == NULL) printf("Память не выделилась\n");
82
83          struct WordNode* words_list = NULL;
84          for (int i = 0; ; i++)
85          {
86              char ch = temp->word[i];
87              if (ch == '\0') break;
88
89              if (ch == ' ')
90              {
91                  words_list = WordNode__add_top(words_list, str);
92                  str_size = 0;
93                  str = (char*) realloc(str, str_size * sizeof(char));
94                  continue;
95              }
96              str_size += 1;
97              str = (char*) realloc(str, str_size * sizeof(char));
98              str[str_size - 1] = ch;
99          }
100
101         int number = atoi( WordNode__get_bottom(words_list)->word );
102         if( number != 0 && number % 2 == 0 )
103         {
104             result = WordNode__add_top(result, temp->word);
105         }
106         words_list = WordNode__Destructor(words_list);
107         free(str);
108     }
109
110     string_list = WordNode__Destructor(string_list);
111
112     printf(" ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~\n");
113     for(struct WordNode* temp = WordNode__get_bottom(result); temp != NULL; temp = temp->right)
114     {
115         int file_descriptor = 1;
```

```
116         char* buf = temp->word;
117         int buf_size = strlen(buf);
118         write(file_descriptor, buf, buf_size);
119         write(file_descriptor, "\n", 1);
120     }
121
122     result = WordNode__Destructor(result);
123
124     return 0;
125 }
126
127 // реализация прототипов
128
129 struct WordNode* WordNode__Constructor(struct WordNode* object, char* word)
130 {
131     object = (struct WordNode*) malloc(sizeof(struct WordNode));
132     if (object == NULL) printf("Память не выделилась\n");
133     //printf("%p Constructor\n", object);
134     if (object == NULL)
135     {
136         printf("Не выделилась память\n");
137     }
138
139     object->left = NULL;
140     object->right = NULL;
141
142     object->word = (char*) calloc(strlen(word), sizeof(char));
143     if (object->word == NULL) printf("Память не выделилась\n");
144     if (object->word == NULL)
145     {
146         printf("Не выделилась память\n");
147     }
148     strcpy(object->word, word);
149
150     return object;
151 }
152
153 struct WordNode* WordNode__Destructor(struct WordNode* object)
154 {
155     for (struct WordNode* temp = object; temp != NULL; )
156     {
157         free(temp->word);
158         object = temp;
159         //printf("%p Destructor\n", object);
160         temp = temp->left;
161         free(object);
162     }
163     return NULL;
164 }
165
166 struct WordNode* WordNode__add_top(struct WordNode* object, char* word)
167 {
168     if (word == NULL)
169     {
170         return object;
171     }
172
173     if (word[0] == '\0')
174     {
175         return object;
```

```
    }

    struct WordNode* new_node = WordNode__Constructor(new_node, word);
    new_node->left = object;
    if (object == NULL)
    {
        return new_node;
    }

    object->right = new_node;
    return new_node;
}

struct WordNode* WordNode__get_bottom(struct WordNode* object)
{
    if (object == NULL)
    {
        return NULL;
    }
    struct WordNode* temp = object;
    while(temp->left != NULL)
    {
        temp = temp->left;
    }
    return temp;
}

void WordNode__set_word(struct WordNode* object, char* word)
{
    free(object->word);
    object->word = (char*) calloc(strlen(word), sizeof(char));
    if ( object->word == NULL )
    {
        printf("Память не выделилась\n");
        return;
    }
    strcpy(object->word, word);
}

int get_depth_long_int(long int N)
{
    int i = 0;
    for( ; N>0; N = N / 10)
    {
        ++i;
    }
    return i;
}

char* get_string_with_number_hash(char* str, long int number)
{
    int len = get_depth_long_int(number);
    char* substr = (char*) calloc(len, sizeof(char));
    if (substr == NULL) printf("Память не выделилась\n");
    long int N = number;
    for (int i = 0; i < len; i += 1)
    {
        char ch;
        switch(N % 10)
        {
```

```
            case 0: ch = '0'; break;
            case 1: ch = '1'; break;
            case 2: ch = '2'; break;
            case 3: ch = '3'; break;
            case 4: ch = '4'; break;
            case 5: ch = '5'; break;
            case 6: ch = '6'; break;
            case 7: ch = '7'; break;
            case 8: ch = '8'; break;
            case 9: ch = '9'; break;
            default: ch = '_';
        }
        N = N / 10;
        substr[i] = ch;
    }

    char* result = (char*) calloc(len, sizeof(char));
    if (result == NULL) printf("Память не выделилась\n");
    for (int i = 0; i < len; i++)
    {
        result[i] = substr[(len - 1) - i];
    }
    free(substr);

    str = (char*) realloc(str, (5 + len) * sizeof(char));
    strcat(str, result);
    free(result);

    return str;
}
```

Компиляция

Листинг: Terminal

```
make
```

Листинг: Out

```
mkdir -p bin
cd bin; gcc ./../src/main.c -o app
```

Тест 1 (ps)

Листинг: Terminal

```
make test1
```

Листинг: Out

```
cd bin; ps | ./app
 ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
  PID TTY          TIME CMD
 1745 pts/2    00:00:00 bash
 6516 pts/2    00:00:00 make
 6517 pts/2    00:00:00 sh
 6518 pts/2    00:00:00 ps
 6519 pts/2    00:00:00 app
 ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
 6516 pts/2    00:00:00 make447124886
 6518 pts/2    00:00:00 ps723179902
```

Тест 2 (ps | sort)

Листинг: Terminal

```
make test2
```

Листинг: Out

```
cd bin; ps | sort | ./app
 ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
 1745 pts/2    00:00:00 bash
 6524 pts/2    00:00:00 make
 6525 pts/2    00:00:00 sh
 6526 pts/2    00:00:00 ps
 6527 pts/2    00:00:00 sort
 6528 pts/2    00:00:00 app
  PID TTY          TIME CMD
 ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
 6524 pts/2    00:00:00 make846930886
 6526 pts/2    00:00:00 ps480069024
 6528 pts/2    00:00:00 app424238335
```

Тест 3 (ps | head -n 4)

Листинг: Terminal

```
make test3
```

Листинг: Out

```
cd bin; ps | head -n 4 | ./app
 ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
  PID TTY          TIME CMD
 1745 pts/2    00:00:00 bash
 6534 pts/2    00:00:00 make
 6535 pts/2    00:00:00 sh
 ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
 6534 pts/2    00:00:00 make447124886
```

Тест 4 (ps | tail -n 4)

Листинг: Terminal

```
make test4
```

Листинг: Out

```
cd bin; ps | tail -n 4 | ./app
 ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
 6544 pts/2    00:00:00 sh
 6545 pts/2    00:00:00 ps
 6546 pts/2    00:00:00 tail
 6547 pts/2    00:00:00 app
 ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
 6544 pts/2    00:00:00 sh569721492
 6546 pts/2    00:00:00 tail447124886
```

Очистка проекта

Листинг: Terminal

```
make clean
```

Листинг: Out

```
rm -rfv bin
removed 'bin/app'
removed directory 'bin'
```

Часть III Вариант 5, 15

Откройте любой текстовый файл и добавляйте в стандартном выводе к имени процессов не число, а очередное слово из этого файла.

Решение:

Листинг: main.c

```c
//библиотеки

#include <unistd.h> //read()
#include <stdio.h>  //printf()
#include <stdlib.h> //calloc(), realloc(), free()
#include <string.h> //strcpy(), strlen(), strcat()

//прототипы

struct WordNode
{
    char* word;
    struct WordNode* left;
    struct WordNode* right;
};
struct WordNode* WordNode__Constructor(struct WordNode* object, char* word);
struct WordNode* WordNode__Destructor(struct WordNode* object);
struct WordNode* WordNode__add_top(struct WordNode* object, char* word);
struct WordNode* WordNode__get_bottom(struct WordNode* object);
struct WordNode* WordNode__get_by_index(struct WordNode* object, int index);
void WordNode__set_word(struct WordNode* object, char* word);
struct WordNode* get_words_file_list(const char path[]);
char* copy_strg1_to_strg2(char* string1, char* string2);

//главная функция

int main()
{
    struct WordNode* list_file_words = get_words_file_list("../lorem100.html");
    struct WordNode* string_list = NULL;
    struct WordNode* result = NULL;

    const int buffer_size = 1;
    char buffer[buffer_size];

    int str_size = 0;
    char* str = (char*) calloc(str_size, sizeof(char*));
    while(read(0, buffer, buffer_size))
    {
        char character = buffer[0];
        if (character == '\n')
        {
            string_list = WordNode__add_top(string_list, str);

            str_size = 0;
            str = (char*) realloc(str, str_size * sizeof(char));

            continue;
        }
        str_size += 1;
        str = (char*) realloc(str, str_size * sizeof(char));
        str[str_size - 1] = character;
```

```
    }
    free(str);

    printf(" ~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~\n");
    for (struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
    temp->right)
    {
        printf("%s\n", temp->word);
    }

    int index_file_word = 0;
    for (struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
    temp->right)
    {
        char* string1 = temp->word;
        char* string2 = WordNode__get_by_index(list_file_words, index_file_word)->word;
        char* strng = copy_strg1_to_strg2(string1, string2);

        WordNode__set_word(temp, strng);
        free(strng);

        index_file_word += 1;
    }

    for (struct WordNode* temp = WordNode__get_bottom(string_list); temp != NULL; temp =
    temp->right)
    {
        str_size = 0;
        str = (char*) calloc(str_size, sizeof(char));
        if (str == NULL) printf("Память не выделилась\n");

        struct WordNode* words_list = NULL;
        for (int i = 0; ; i++)
        {
            char ch = temp->word[i];
            if (ch == '\0') break;

            if (ch == ' ')
            {
                words_list = WordNode__add_top(words_list, str);
                str_size = 0;
                str = (char*) realloc(str, str_size * sizeof(char));
                continue;
            }
            str_size += 1;
            str = (char*) realloc(str, str_size * sizeof(char));
            if (str == NULL) printf("Память не выделилась\n");
            str[str_size - 1] = ch;
        }

        int number = atoi( WordNode__get_bottom(words_list)->word );
        if( number != 0 && number % 2 == 0 )
        {
            result = WordNode__add_top(result, temp->word);
        }
        words_list = WordNode__Destructor(words_list);
        free(str);
    }

    string_list = WordNode__Destructor(string_list);
```

```
110
111     printf(" ~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~\n");
112     for(struct WordNode* temp = WordNode__get_bottom(result); temp != NULL ; temp = temp->right)
113     {
114         int file_descriptor = 1;
115         char* buf = temp->word;
116         int buf_size = strlen(buf);
117         write(file_descriptor, buf, buf_size);
118         write(file_descriptor, "\n", 1);
119     }
120     result = WordNode__Destructor(result);
121
122     list_file_words = WordNode__Destructor(list_file_words);
123
124     return 0;
125 }
126
127 //реализация прототипов
128
129 struct WordNode* WordNode__Constructor(struct WordNode* object, char* word)
130 {
131     object = (struct WordNode*) malloc(sizeof(struct WordNode));
132     if (object == NULL) printf("Память не выделилась\n");
133     //printf("%p Constructor\n", object);
134
135     object->left = NULL;
136     object->right = NULL;
137
138     object->word = (char*) calloc(strlen(word), sizeof(char));
139     if (object->word == NULL) printf("Память не выделилась\n");
140     strcpy(object->word, word);
141
142     return object;
143 }
144
145 struct WordNode* WordNode__Destructor(struct WordNode* object)
146 {
147     for (struct WordNode* temp = object; temp != NULL; )
148     {
149         free(temp->word);
150         object = temp;
151         //printf("%p Destructor\n", object);
152         temp = temp->left;
153         free(object);
154     }
155     return NULL;
156 }
157
158 struct WordNode* WordNode__add_top(struct WordNode* object, char* word)
159 {
160     if (word == NULL)
161     {
162         return object;
163     }
164
165     if (word[0] == '\0')
166     {
167         return object;
168     }
169
```

```
170     struct WordNode* new_node = WordNode__Constructor(new_node, word);
171     new_node->left = object;
172     if (object == NULL)
173     {
174         return new_node;
175     }
176 
177     object->right = new_node;
178     return new_node;
179 }
180 
181 struct WordNode* WordNode__get_bottom(struct WordNode* object)
182 {
183     if (object == NULL)
184     {
185         return NULL;
186     }
187     struct WordNode* temp = object;
188     while(temp->left != NULL)
189     {
190         temp = temp->left;
191     }
192     return temp;
193 }
194 
195 struct WordNode* WordNode__get_by_index(struct WordNode* object, int index)
196 {
197     struct WordNode* temp = WordNode__get_bottom(object);
198     for(int i = 0; temp != NULL && i != index; temp = temp->right, i += 1)
199     {
200 
201     }
202     return temp;
203 }
204 
205 void WordNode__set_word(struct WordNode* object, char* word)
206 {
207     free(object->word);
208     object->word = (char*) calloc(strlen(word), sizeof(char));
209     if ( object->word == NULL )
210     {
211         printf("Память не выделилась\n");
212         return;
213     }
214     strcpy(object->word, word);
215 }
216 
217 struct WordNode* get_words_file_list(const char path[])
218 {
219     FILE* file_pointer = fopen(path, "r");
220     if (file_pointer == NULL)
221     {
222         printf("File %s not opened\n", path);
223         return 0;
224     }
225 
226     int word_size = 0;
227     char* word = (char*) calloc(word_size, sizeof(char));
228     if (word == NULL) printf("Память не выделилась\n");
229     struct WordNode* list_file_words = NULL;
```

```
230
231     while(!feof(file_pointer))
232     {
233         char character = fgetc(file_pointer);
234         if(
235             character == '␣'
236             || character == ','
237             || character == '.'
238             || character == '!'
239         )
240         {
241             if (word_size == 0)
242             {
243                 continue;
244             }
245             list_file_words = WordNode__add_top(list_file_words, word);
246             word_size = 0;
247             word = (char*) realloc(word, word_size * sizeof(char));
248             continue;
249         }
250         word_size += 1;
251         word = (char*) realloc(word, word_size * sizeof(char));
252         word[word_size - 1] = character;
253     }
254
255     if (word != NULL)
256     {
257         free(word);
258     }
259     fclose(file_pointer);
260
261     return list_file_words;
262 }
263
264 char* copy_strg1_to_strg2(char* string1, char* string2)
265 {
266     int string1_size = strlen(string1);
267     int string2_size = strlen(string2);
268
269     int result_string_size = string1_size + string2_size;
270     char* result_string = (char*) calloc(result_string_size, sizeof(char));
271     if (result_string == NULL) printf("Память не выделилась\n");
272     strcpy(result_string, string1);
273     strcat(result_string, string2);
274
275     return result_string;
276 }
```

Листинг: lorem100.html

```
1 Lorem ipsum, dolor sit amet consectetur adipisicing elit. Tempora, et quisquam! Magni veritatis,
      impedit aspernatur earum quos iure explicabo dolor commodi, enim optio sapiente. Natus aliquam
      non deleniti, ipsam molestiae corporis culpa. Soluta labore deleniti at ipsum ab doloribus
      quidem optio natus. Iusto, numquam nam ipsum dolorem, corrupti veniam esse repellat quo
      obcaecati molestiae voluptate commodi. Molestias, dolore dolorum? Veniam cumque sapiente nobis
      enim numquam, eos ratione voluptas voluptatem fugit libero. Quisquam repellat ex culpa
      accusantium tempora amet quod facere unde, temporibus, veniam delectus earum. Consequuntur
      doloremque nisi sequi asperiores ipsum earum facilis natus! Dolorum consequatur nam
      perferendis nesciunt labore.
```

Листинг: Makefile

```
all:
    mkdir -p bin
    cd bin; gcc ./../src/main.c -o app

clean:
    rm -rfv bin

run:
    cd bin; ./app

test1:
    cd bin; ps | ./app

test2:
    cd bin; ps | sort | ./app

test3:
    cd bin; ps | head -n 4 | ./app

test4:
    cd bin; ps | tail -n 4 | ./app
```

Компиляция

Листинг: Terminal

```
make
```

Листинг: Out

```
mkdir -p bin
cd bin; gcc ./../src/main.c -o app
```

Test 1 (ps)

Листинг: Terminal

```
make test1
```

Листинг: Out

```
cd bin; ps | ./app
~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
    PID TTY          TIME CMD
1745 pts/2    00:00:00 bash
2905 pts/2    00:00:00 make
2906 pts/2    00:00:00 sh
2907 pts/2    00:00:00 ps
2908 pts/2    00:00:00 app
~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
2906 pts/2    00:00:00 shsit
2908 pts/2    00:00:00 appconsectetur
```

Test 2 (ps | sort)

Листинг: Terminal

```
make test2
```

Листинг: Out

```
cd bin; ps | sort | ./app
~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
1745 pts/2    00:00:00 bash
2924 pts/2    00:00:00 make
2925 pts/2    00:00:00 sh
2926 pts/2    00:00:00 ps
2927 pts/2    00:00:00 sh
2928 pts/2    00:00:00 app
    PID TTY          TIME CMD
~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
2924 pts/2    00:00:00 makeipsum
2926 pts/2    00:00:00 pssit
2928 pts/2    00:00:00 appconsectetur
```

Test 3 (ps | head -n 4)

Листинг: Terminal

```
make test3
```

Листинг: Out

```
cd bin; ps | head -n 4 | ./app
~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
    PID TTY          TIME CMD
1745 pts/2    00:00:00 bash
2938 pts/2    00:00:00 make
2939 pts/2    00:00:00 sh
~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
2938 pts/2    00:00:00 makedolor
```

Test 4 (ps | tail -n 4)

Листинг: Terminal

```
make test4
```

Листинг: Out

```
cd bin; ps | tail -n 4 | ./app
~ ~ ~ ~ ~ file descriptor = 0 ~ ~ ~ ~ ~
2954 pts/2    00:00:00 sh
2955 pts/2    00:00:00 ps
2956 pts/2    00:00:00 sh
2957 pts/2    00:00:00 app
~ ~ ~ ~ ~ file descriptor = 1 ~ ~ ~ ~ ~
2954 pts/2    00:00:00 shLorem
2956 pts/2    00:00:00 shdolor
```