



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 №2
Тема:	Многопоточное программирование
Вариант:	10

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 потоков . . . . .	4
Описание основных используемых структур данных . . . . .	5
Блок-схемы . . . . .	7
Примеры работы программы . . . . .	11
Текст программы . . . . .	13

## Задание

Разработать многопоточный вариант программы моделирования лифтовой системы (см. задание 8 из лаб. работы 1). При этом модели БН, ЛП и ЛГ выполняются в рамках отдельных потоков программы.

Задание 8 из лабораторной работы 1:

Разработать программу, моделирующую в реальном времени работу лифтовой системы десятиэтажного дома, состоящей из блока управления БУ, пассажирского ЛП и грузового ЛГ лифтов. БУ отслеживает текущее состояние лифтов, принимает вызовы пассажиров на этажах, диспетчеризует движение лифтов (выбирает "ближайший" направляет к пассажирам, тормозит на промежуточных этажах и т.п.), индицирует состояние лифтов. Лифты исполняют команды находящихся в них пассажиров и команды БУ. Скорость ЛП в два раза выше скорости ЛГ. Работа каждого элемента системы (БН, ЛП и ЛГ) моделируется отдельным процессом. Клавиши "1"..."0" используются для имитации клавиш вызова лифта на этажах здания. Следующий ряд клавиш "q"..."r" имитирует кнопки внутри ЛП, а ряд "a"..."": внутри ЛГ. Для обработки нажатий клавиш использовать неканонический режим. Визуализация перемещения лифтов може выглядеть, например, след. образом.

ЛП - - - - 2 - - - - -

ЛГ - - - - - 0 -

Здесь цифры индицируют количество пассажиров в лифте.

## Описание структуры программы и реализованных способов взаимодействия потоков

Для выполнения поставленной задачи реализовано разделение на 4 потока: Поток БУ, поток ЛП, поток ЛГ и поток визуализатора.

Для моделирования работы лифтов реализована пользовательская структура **Lift**. У данной структуры есть 2 объекта - **lp** для ЛП и **lg** для ЛГ.

Для ЛП и ЛГ реализованы семафоры для сокращения накладных расходов, блокирующие соответствующий им лифт при отсутствии для него команд на выполнение. Для синхронизации потоков реализована атомарная переменная *done*, которой присваивается не нулевое значение при завершении основного цикла БУ. Изменение *done* сообщает дочерним потокам о прекращении их внутренних циклов выполнения и последующем завершении.

Схема взаимодействия потоков представлена на рисунке 1.

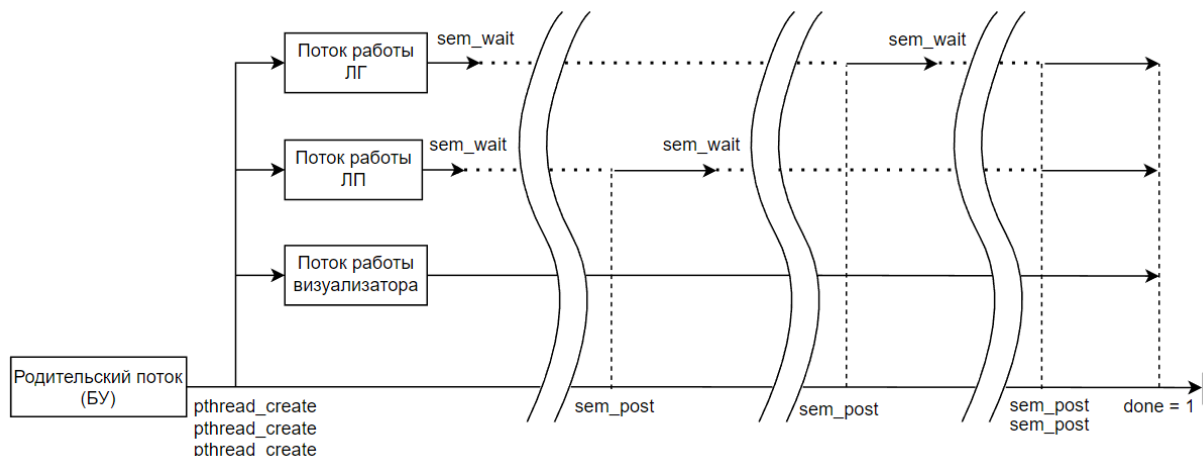


Рис. 1. Схема взаимодействия потоков

Поток БУ является главным и создает остальные потоки. После этого поток переводит ввод в неканонический режим и циклически считывает введенные символы по одному. При получении символа завершения файла или символа, не соответствующего одной из команд в задании (проверка функцией **right\_command**), цикл ввода прекращается. БУ определяет, к какому типу (команда на посадку в лифт, команда

на высадку из ЛП, команда на высадку из ЛГ) принадлежит полученная команда и совершает соответствующий блок действий. При считывании команды на посадку БУ высчитывает с помощью функции **get\_queue\_time**, какой из лифтов потенциально выполнит запрос на посадку быстрее, и добавляет поступивший запрос в его очередь выполнения запросов командой **to\_queue**. При получении команды на высадку из ЛП или ЛГ у соответствующего лифта проверяется, есть ли в нем люди, так как иначе на кнопку вызова нельзя было бы нажать. Если люди есть, с помощью функции **to\_queue** запрос на высадку отправляется в очередь запросов соответствующего лифта. При получении любой команды лифтом происходит вызов **sem\_post** семафора соответствующего лифта, таким образом он разблокируется и начинает движение. Когда лифт выполнил запрос, происходит вызов **sem\_wait**, с целью перевода лифта в блокировку при отсутствии запросов. По завершении цикла происходит присваивание глобальной переменной **done** в 1, сигнализирующей другим потокам о завершении работы. Происходит вызов **sem\_post** семафоров потоков обоих лифтов для их разблокировки, чтобы они могли самостоятельно завершиться.

В потоках, реализующих работу ЛП и ЛГ сперва происходит инициализация соответствующего потоку семафора. В цикле, выполняемом пока **done** равно 0, происходит вызов функции **move** соответствующего лифта. Функция **move** имитирует действия лифта, занимающее время: посадка или высадка людей и перемещение к следующему запросу в очереди.

В потоке, реализующем работу визуализатора, в цикле, пока **done** равно 0, происходит вызов функции **display** и **usleep(100000)** для задержки вывода. Функция **display** выводит в одну строку местоположение лифтов в формате, указанном в задании, и производит перевод ввода обратно в начало строки.

## Описание основных используемых структур данных

Реализована пользовательская структура лифта **Lift**, состоящая из следующих информационных полей:

- **int speed** - Коэффициент скорости движения лифта между этажами, где 1 зна-

чение скорости эквивалентно 1 секунде;

- **int occupancy** - Заполненность лифта на данный момент;
- **int cur\_floor** - Этаж лифта на данный момент;
- **capacity** - Максимальная заполненность лифта;
- **sem\_t sem** - Семафор лифта, отвечающий за блокировку лифта, когда отсутствуют запросы на вызов;
- **char queue[FLOOR\_COUNT \* 2 + 1]** - Порядок остановки лифта на этажах, где FLOOR\_COUNT - количество этажей в доме;
- **char occupancy\_mask[FLOOR\_COUNT \* 2 + 1]** - Показатель, сколько на каждом вызове хочет зайти или выйти людей.

В структуре данных **Lift** также реализованы следующие методы:

- **void to\_queue(int floor, int occupancy\_change)** - Добавление в очередь запроса на вызов;
- **void move()** - Передвижение лифта ближе к этажу следующего вызова за определенное скоростью время;
- **void insert(int dest, char num, int occupancy\_change)** - Вставка элемента по заданному индексу в массив порядка остановки;
- **void del(int dest)** - Удаление по заданному индексу элемента из массива порядка остановки;
- **int get\_queue\_time(int floor)** - Получение времени, требующегося лифту для добирания до заданного этажа;
- **Lift(int \_speed, int \_capacity)** - Конструктор, задающий коэффициент скорости и максимальный объем лифта.

## Блок-схемы

На рисунке 2 представлена блок-схема функции **main**. На рисунке 3 представлена схема работы основной функции БУ. На рисунке 4 представлена блок-схема работы функций лифтов. На рисунке 5 представлена блок-схема работы визуализатора.

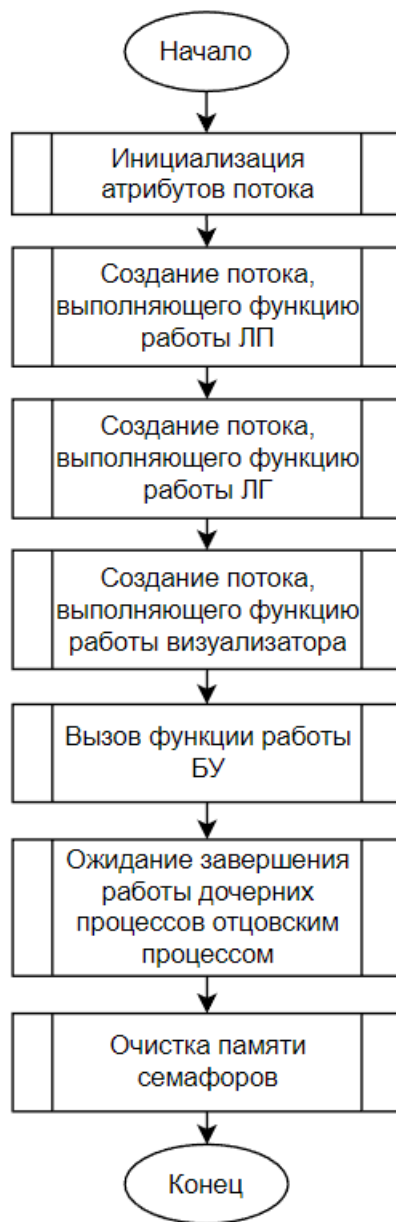


Рис. 2. Блок схема функции main

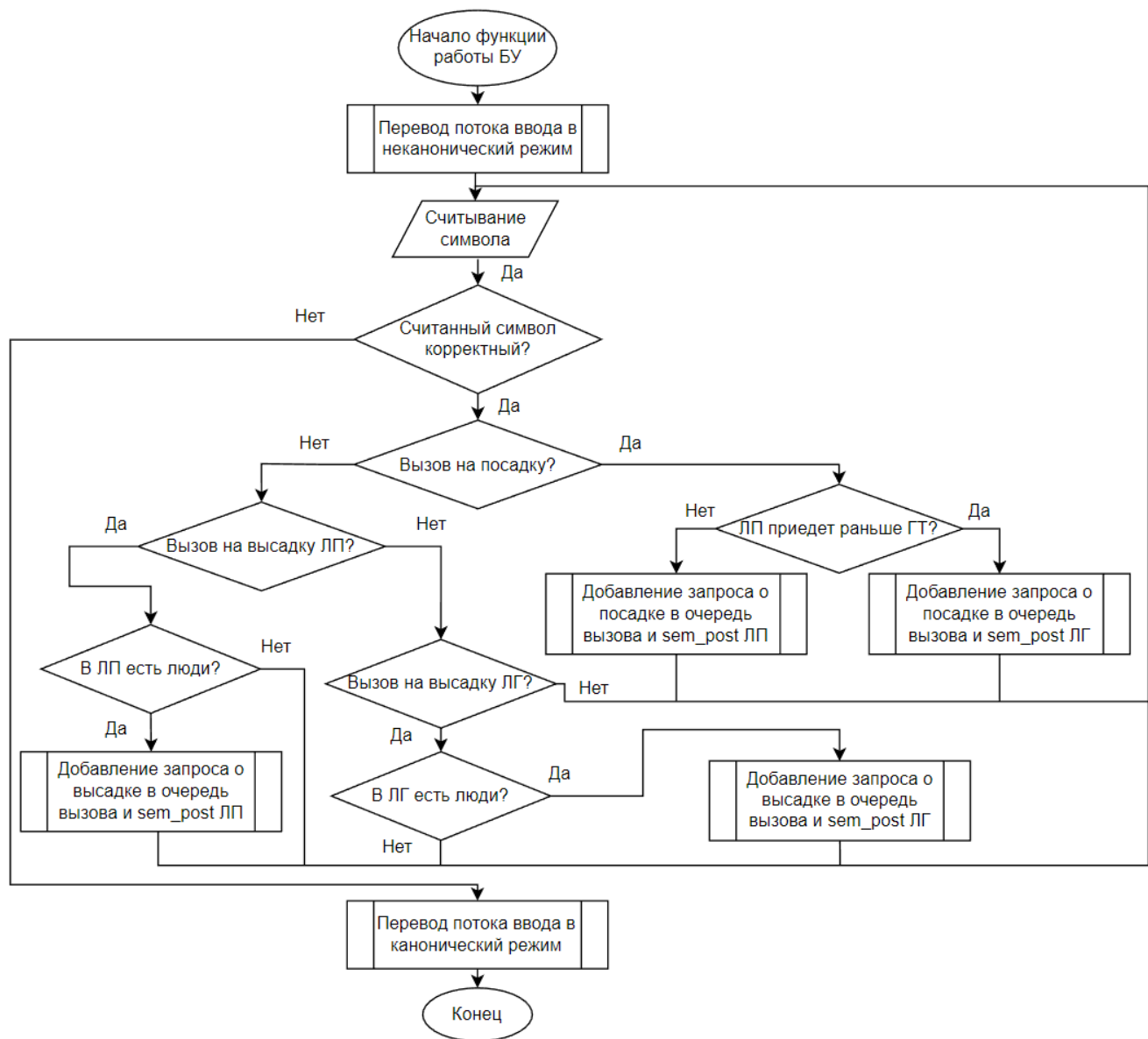


Рис. 3. Блок схема работы основной функции БУ



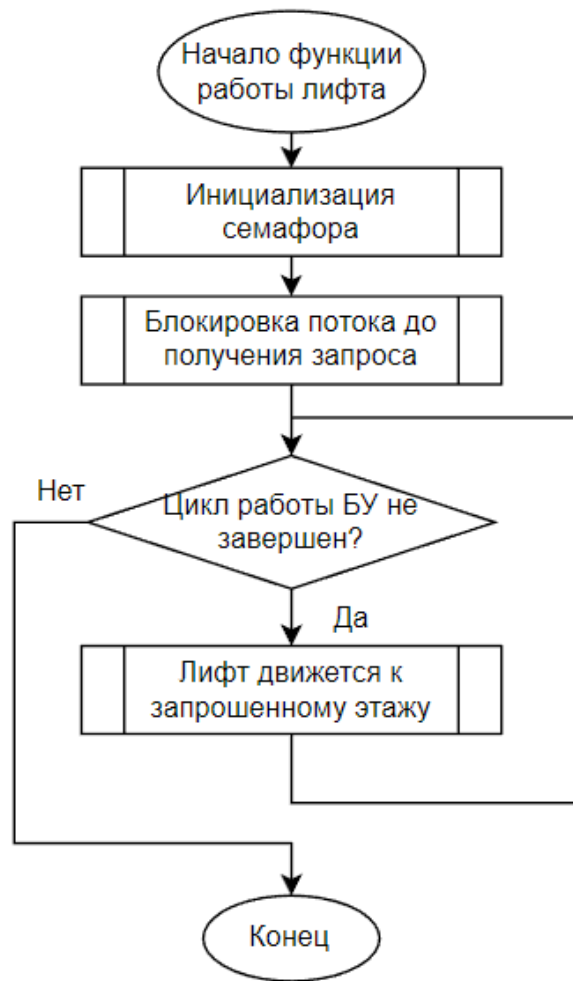


Рис. 4. Блок схема работы функций лифтов

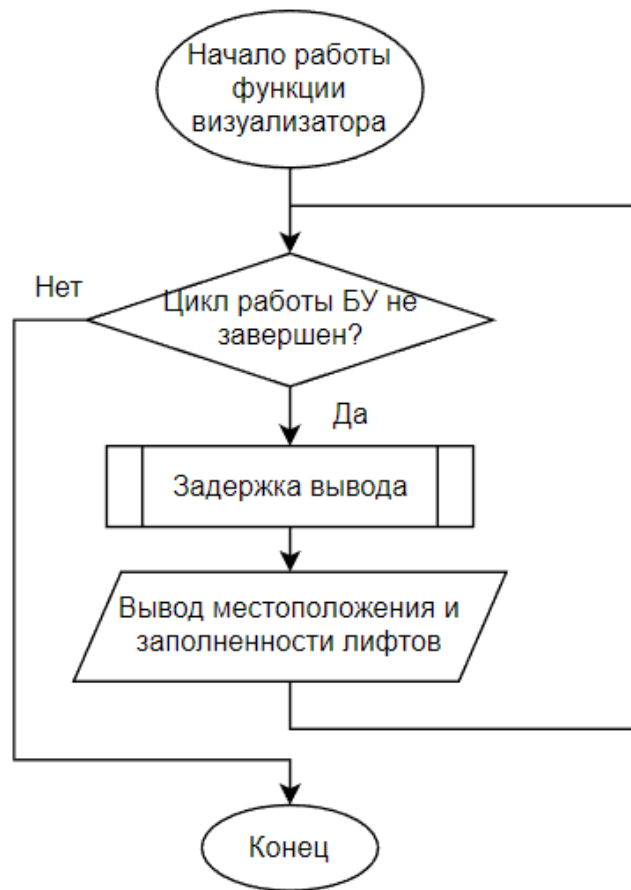
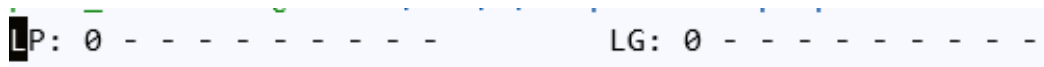


Рис. 5. Блок схема работы визуализатора

## Примеры работы программы

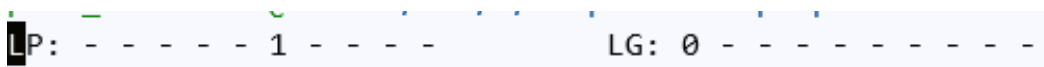
На рисунке 6 приведено состояние лифтов по умолчанию:

A horizontal bar representing the status of elevators. On the left, there is a small black square icon. To its right, the text 'LP: 0' is followed by a series of dashes. Further to the right, the text 'LG: 0' is followed by another series of dashes. Above the bar, there are several small colored dots (green, blue, red) indicating specific floor levels or calls.

LP: 0 - - - - - LG: 0 - - - - -

Рис. 6. Пример работы программы №1

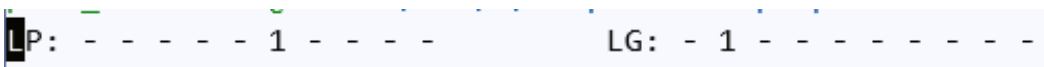
На рисунке 7 приведен пример последующего вызова одного из лифтов на посадку на 6 этаж. В данной ситуации ЛП приехал раньше.

A horizontal bar representing the status of elevators. On the left, there is a small black square icon. To its right, the text 'LP: - - - - - 1 - - - - -' is shown, where the '1' is in a larger font. Further to the right, the text 'LG: 0 - - - - -' is shown. Above the bar, there are several small colored dots (green, blue, red) indicating specific floor levels or calls.

LP: - - - - - 1 - - - - - LG: 0 - - - - -

Рис. 7. Пример работы программы №2

На рисунке 8 приведен пример последующего вызова лифта на посадку, при котором ближе ехать было ЛГ:

A horizontal bar representing the status of elevators. On the left, there is a small black square icon. To its right, the text 'LP: - - - - - 1 - - - - -' is shown, where the '1' is in a larger font. Further to the right, the text 'LG: - 1 - - - - -' is shown. Above the bar, there are several small colored dots (green, blue, red) indicating specific floor levels or calls.

LP: - - - - - 1 - - - - - LG: - 1 - - - - -

Рис. 8. Пример работы программы №3

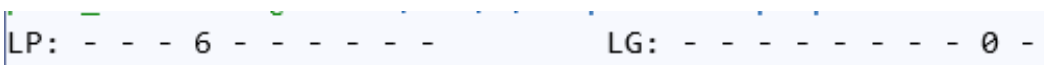
На рисунке 9 приведен пример последующей высадки пассажира из ЛГ на 9 этаже:

A horizontal bar representing the status of elevators. On the left, there is a small black square icon. To its right, the text 'LP: - - - - - 1 - - - - -' is shown, where the '1' is in a larger font. Further to the right, the text 'LG: - - - - - 0 -' is shown. Above the bar, there are several small colored dots (green, blue, red) indicating specific floor levels or calls.

LP: - - - - - 1 - - - - - LG: - - - - - 0 -

Рис. 9. Пример работы программы №4

На рисунке 10 приведен пример достижения ЛП максимальной грузоподъемности, теперь при выполнении запроса на посадку количество людей в нем не увеличивается:

A horizontal bar representing the status of elevators. On the left, there is a small black square icon. To its right, the text 'LP: - - - 6 - - - - -' is shown, where the '6' is in a larger font. Further to the right, the text 'LG: - - - - - 0 -' is shown. Above the bar, there are several small colored dots (green, blue, red) indicating specific floor levels or calls.

LP: - - - 6 - - - - - LG: - - - - - 0 -

Рис. 10. Пример работы программы №5

На рисунке 11 приведен пример результата ввода не верной символьной команды, приводящей к завершению выполнения программы:

```
LP: - - - 6 - - - - - LG: - - - - - - - - 0 -  
petr_kildishev@Balls:/mnt/d/Разработка программных систем/lab2_var10/lab2_var10$
```

Рис. 11. Пример работы программы №6

## Текст программы

Ниже в листинге 1 представлен текст программы.

Листинг 1. Листинг программы

```
1 #include <fcntl.h>
2 #include <termios.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <ctype.h>
6 #include <unistd.h>
7 #include <cstring>
8 #include <pthread.h>
9 #include <stdlib.h>
10 #include <semaphore.h>
11
12 const int FLOOR_COUNT = 10; // Количество этажей
13
14 struct Lift // Структура лифта
15 {
16     int speed; // Коэффициент скорости движения лифта, где 1 = 100000 мс
17     int occupancy; // Заполненность лифта
18     int cur_floor; // Нынешний этаж лифта
19     int capacity; // Максимальный объем лифта
20     sem_t sem; // Семафор лифта, отвечающий за блокировку лифта, когда в нем нет
        людей и нет сигнала о вызове
21     char queue[FLOOR_COUNT * 2 + 1]; // Порядок остановки лифта на этажах
22     char occupancy_mask[FLOOR_COUNT * 2 + 1]; // Показатель, сколько на каждом
        вызове хочет зайти или выйти людей
23
24     Lift(int _speed, int _capacity): speed(_speed), occupancy(0), cur_floor(0),
        capacity(_capacity) //
        Конструктор
25     {
26         for (int i = 0; i < FLOOR_COUNT * 2 + 1; i++) // Больше, чем 2 * кол-во
            этажей вызовов быть не может, т. к. иначе вызовы идут в маску
27         {
28             queue[i] = '\0'; // Пустые элементы queue заполняются пустыми байтами для
                работы strlen(queue)
29             occupancy_mask[i] = 0;
30         }
31     }
32
33     void to_queue(int floor, int occupancy_change); // Добавление в очередь запроса на
        вызов
34     void move(); // Передвижение лифта ближе к этажу следующего вызова за
        определенное скоростью время
35     void insert(int dest, char num, int occupancy_change); // Вставка элемента по
```

```

36     заданному индексу в массив порядка остановки
    void del(int dest); // Удаление по заданному индексу элемента из массива порядка
        остановки
37
38     int get_queue_time(int floor); // Получение времени, требующегося лифту для
        добирания до заданного этажа
39 };
40
41 // dest - индекс вставки, num - какой этаж вставляем, occupancy_change - изменение
    кол-ва людей в лифте на этом этаже
42 void Lift::insert(int dest, char num, int occupancy_change)
43 {
44     for (int i = strlen(queue); i > dest; i--) // Просто сдвиг вправо
45     {
46         queue[i] = queue[i - 1];
47         occupancy_mask[i] = occupancy_mask[i - 1];
48     }
49     queue[dest] = num;
50     occupancy_mask[dest] = occupancy_change;
51 }
52
53 void Lift::del(int dest) // dest - индекс удаления
54 {
55     for (int i = dest; i < strlen(queue); i++) // Сдвиг влево
56     {
57         queue[i] = queue[i + 1];
58         occupancy_mask[i] = occupancy_mask[i + 1];
59     }
60     queue[strlen(queue)] = '\0';
61     occupancy_mask[strlen(queue)] = 0;
62 }
63
64 void Lift::move()
65 {
66     int TICK = 100000;
67     if (strlen(queue) == 0)
68     {
69         //Тут он в состоянии блокировки
70         //usleep(TICK);
71     }
72     else if (queue[0] == cur_floor + '0') // Если пришли в следующий по очереди этаж -
        люди входят или выходят
73     {
74         if (occupancy_mask[0] >= 0) // Внутри происходит проверка на вход не больше
            чем максимального и выход не больше чем минимального кол-ва людей в
            лифте
75         occupancy += (occupancy_mask[0] > capacity - occupancy ? capacity -
            occupancy : occupancy_mask[0]);

```

```

76     else
77         occupancy += (occupancy_mask[0] < occupancy * (-1) ? occupancy * (-1) :
                        occupancy_mask[0]);
78     del(0);
79     usleep(TICK); // Люди входят или выходят TICK времени просто потому что я
                  так решил
80     sem_wait(&sem); // Запрос выполнен, семафор -= 1. Если все запросы
                  выполнились, поток блокируется
81 }
82 else if (queue[0] > cur_floor + '0') // Поднимается к вызову
83 {
84     cur_floor += 1;
85     usleep(TICK * speed); // Ждет время передвижения, зависящее от скорости
86 }
87 else if (queue[0] < cur_floor + '0') // Опускается к вызову
88 {
89     cur_floor -= 1;
90     usleep(TICK * speed); // Ждет время передвижения, зависящее от скорости
91 }
92 }
93
94 void Lift::to_queue(int floor, int occupancy_change) // floor - добавляемый этаж,
                  occupancy_change - изменение кол-ва людей в лифте
95 {
96     // Добавление сделано так, что лифт сначала поднимается до самого высокого этажа
          из очереди вызвавших, потом саускается до самого низкого
97     // Таким образом не происходит "прыганий"вверх-вниз от этажа к этажу
98     if (strlen(queue) == 0) // Если в лифте нет людей - просто добавляет
99     {
100         insert(0, floor + '0', occupancy_change);
101     }
102     else if (floor >= cur_floor) //Если добавляемый этаж выше нынешнего, то добавляем
          перед первым в очереди большим, чем добавляемый
103     {
104         for (int i = 0; i < strlen(queue); i++)
105             if (queue[i] >= floor + '0')
106             {
107                 if (queue[i] > floor + '0')
108                     insert(i, floor + '0', occupancy_change);
109                 else
110                     occupancy_mask[i] += occupancy_change;
111                 return;
112             }
113         insert(strlen(queue), floor + '0', occupancy_change);
114     }
115     else //Если добавляемый этаж ниже нынешнего, то добавляем после первого в
          очереди большего с конца
116     {

```

```

117     for (int i = strlen(queue); i > 0; i--)
118         if (queue[i - 1] >= floor + '0')
119             {
120                 if (queue[i - 1] > floor + '0')
121                     insert(i, floor + '0', occupancy_change);
122                 else
123                     occupancy_mask[i] += occupancy_change;
124                 return;
125             }
126     insert(0, floor + '0', occupancy_change);
127 }
128 }
129
130 int Lift::get_queue_time(int floor) // floor - до какого этажа хотим получить время
131 {
132     // Ищет место для вставки требуемого этажа аналогичным образом с добавлением в
133     // очередь запросов,
134     // но дополнительно считает время на перемещение между этажами, учитывает
135     // время на высадку-посадку пассажиров на этажах
136     int sum = 0, curr_floor = cur_floor; // sum - суммарное время, curr_floor -
137     // "нынешний" этаж при подсчете
138     if (floor >= cur_floor)
139     {
140         for (int i = 0; i < strlen(queue); i++)
141         {
142             if (queue[i] >= floor + '0') // Ищет место для "вставки" этажа
143                 return (sum + abs(curr_floor - floor)) * speed + strlen(queue) - 1 -
144                     i; // Возвращает сумму + время на высадку/посадку и доезд до
145                     // заданного этажа
146             sum += abs(curr_floor - queue[i]); // Добавляет время перемещения до
147             // следующего запроса
148             curr_floor = queue[i]; // Меняет "нынешний" этаж
149         }
150         return abs(curr_floor - floor) * speed; // Добавил бы floor в конец очереди
151     }
152     else
153     {
154         int i_insert = 0;
155         for (int i = strlen(queue); i > 0; i--) // Ищет индекс, куда вставил бы запрос с
156         // конца очереди запросов
157             if (queue[i - 1] >= floor + '0')
158             {
159                 i_insert = i;
160                 break;
161             }
162         for (int i = 0; i < i_insert; i++) // Считает сумму времени на перемещение
163         // между этажами в запросах до полученного индекса
164         {
165             sum += abs(curr_floor - queue[i]);
166         }
167     }
168 }

```



```

158         curr_floor = queue[i];
159     }
160     return (abs(curr_floor - floor) + sum) * speed + i_insert; // Возвращает сумму +
        // время на посадку/высадку + время до заданного этажа
161 }
162 }
163
164 Lift lp(1, 6), lg(2, 9); // Создание лифтов, в грузовом максимум 9 человек, в
        // пассажирском 6
165 int done = 0;
166
167 void display() // Отображение нынешнего местонахождения лифтов
168 {
169     char occupancy_lp[2], occupancy_lg[2]; // Сколько человек в лифтах, в виде строки
        // чтобы было легче выводить с помощью write
170     occupancy_lp[0] = lp.occupancy + '0'; // Переводит количество людей в лифтах в
        // строки просто прибавляя '0', встает ограничение в макс. объеме людей в 9
171     occupancy_lg[0] = lg.occupancy + '0';
172
173     fflush(stdin); // Чистит поток вывода, потому что в неканоническом режиме
        // выводится какая-то чепуха
174
175     // Выводит сообщение сначала для пассажирского, потом для грузового лифта
176     write(1, "LP: ", 4);
177     for (int i = 0; i < FLOOR_COUNT; i++) // Проходит по этажам
178     {
179         if (i == lp.cur_floor) // Если на этом этаже лифт - выводит сколько в нем людей
180         {
181             write(1, occupancy_lp, 2);
182             write(1, " ", 1);
183         }
184         else // Иначе пишет черточку
185             write(1, "—", 2);
186     }
187     write(1, "\tLG: ", 5);
188     for (int i = 0; i < FLOOR_COUNT; i++) // Аналогично пассажирскому лифту
189     {
190         if (i == lg.cur_floor)
191         {
192             write(1, occupancy_lg, 2);
193             write(1, " ", 1);
194         }
195         else
196             write(1, "—", 2);
197     }
198     for (int i = 0; i < 4 * FLOOR_COUNT + 18; i++) // Возвращает вывод в начало
        // строки
199     write(1, "\b", 1);

```

```

200 int right_command(char command, int symbols_used[]) // Проверка на то, что полученная
    команда правильная
201 {
202     for (int i = 0; i < 30; i++)
203         if (command == symbols_used[i])
204             return 1;
205     return 0;
206 }
207
208 void* lp_controller(void* arg_p) // Функция потока пассажирского лифта
209 {
210     sem_init(&(lp.sem), 1, 0); // Инициализация семафора пассажирского лифта
211     sem_wait(&(lp.sem)); // Перевод лифта в состояние ожидание запроса
212     while (!done) // Цикл до завершения цикла основной программы, отмечающимся
        done = 1
213     {
214         lp.move(); // Движения лифта, занимающие время
215     }
216     pthread_exit(0); // Завершение выполнения потока
217 }
218
219 void* lg_controller(void* arg_p) // Все аналогично функции потока пассажирского
    лифта, но с грузовым лифтом
220 {
221     sem_init(&(lg.sem), 1, 0);
222     sem_wait(&(lg.sem));
223     while (!done)
224     {
225         lg.move();
226     }
227     pthread_exit(0);
228 }
229
230 void* visualizer(void* arg_p) // Функция потока-визуализатора, который выводит на
    экран нынешнее положение лифтов
231 {
232     while (!done) // Пока не завершится цикл основной программы
233     {
234         usleep(100000); // Время ожидания между выводами
235         display(); // Функция вывода
236     }
237     write(1, "\n", 1); // Перевод строки после окончания, остается последнее положение
        лифтов
238     pthread_exit(0); // Завершение выполнения потока
239 }
240
241 void dispatcher() // Основная функция потока контрольного блока лифтов

```

```

242 {
243     char command;
244
245     int symbols_used[30] = { // Разные команды для лифтов
246         '1', '2', '3', '4', '5', '6', '7', '8', '9', '0',
247         'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p',
248         'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';'
249     };
250     int floor_converter[121]; // Массив для перевода команд из символа в определенный
251     // этаж
252     for (int i = 0; i < 30; i++) // Заполняет массив перевода, при этом если в десятке 0 -
253         // вызов на посадку, 1 - на высадку в ЛП, 2 - высадку в ЛГ
254         floor_converter[symbols_used[i]] = i;
255
256     // Блок перевода ввода в неканонический режим
257     struct termios savetty;
258     struct termios tty;
259
260     if (!isatty(0)) { // Проверка на то, что ввод из терминала
261         fprintf(stderr, "stdin not terminal\n");
262         exit(1);
263     }
264
265     tcgetattr(0, &tty);
266     savetty = tty;
267     tty.c_lflag &= ~(ICANON ECHO ISIG);
268     tty.c_cc[VMIN] = 1;
269     tcsetattr(0, TCSAFLUSH, &tty);
270     // Конец блока перевода ввода в неканонический режим
271
272     while (read(0, &command, 1)) // Пока может считать команду - идет по циклу
273     {
274         if (right_command(command, symbols_used)) // Если получили не верную
275             // команду - конец ввода
276             command = floor_converter[command];
277         else
278             break;
279         switch (command / 10) // Определяет какого рода команду получили
280         {
281             case 0: // Команда на вызов на посадку
282                 if (lp.get_queue_time(command % 10) <= lg.get_queue_time(command %
283                     10)) // Если ЛП придет раньше или за то же время, что ЛГ, то едет
284                     // ЛП
285                 {
286                     sem_post(&(lp.sem)); // При получении команды, семафор ЛП += 1.
287                     // Если до этого он был заблокирован - начинает работать
288                     lp.to_queue(command % 10, 1); // Добавление нового запроса на
289                     // перемещение к этажу
290                 }
291             }
292         }
293     }

```

```

283     }
284     else
285     {
286         sem_post(&(lg.sem)); // Аналогично с ЛП
287         lg.to_queue(command % 10, 1);
288     }
289     break;
290 case 1: // Команда на вызов на высадку из ЛП
291     if (lp.occupancy > 0) // Если людей в ЛП нет, то вызов не мог произвестись
292     {
293         sem_post(&(lp.sem)); // Новый запрос => семафор += 1
294         lp.to_queue(command % 10, -1); // Добавляет запрос в очередь
295     }
296     break;
297 case 2: // Команда на вызов на высадку из ЛГ, все аналогично с ЛП
298     if (lg.occupancy > 0)
299     {
300         sem_post(&(lg.sem));
301         lg.to_queue(command % 10, -1);
302     }
303     break;
304 }
305 }
306
307 done = 1; // Если цикл завершился - сообщает об этом через глобальную атомарную
           переменную done
308 sem_post(&(lp.sem)); // Разблокирует поток ЛП
309 sem_post(&(lg.sem)); // Разблокирует поток ЛГ
310
311 tcsetattr(0, TCSAFLUSH, &savetty); // Переводит ввод в канонический режим
312 }
313
314 int main(int argc, char** argv)
315 {
316     pthread_t tidp, tidg, tidv; // tidp - поток ЛП, tidg - поток ЛГ, tidv - поток
           визуализатора
317     pthread_attr_t pattr;
318
319     // Инициализирует атрибуты потоков
320     pthread_attr_init(&pattr);
321     pthread_attr_setscope(&pattr, PTHREAD_SCOPE_SYSTEM);
322     pthread_attr_setdetachstate(&pattr, PTHREAD_CREATE_JOINABLE);
323
324     // Создает потоки
325     pthread_create(&tidp, &pattr, lp_controller, NULL);
326     pthread_create(&tidg, &pattr, lg_controller, NULL);
327     pthread_create(&tidv, &pattr, visualizer, NULL);

```

```
328
329     dispatcher(); // Вызов функции потока блока контроля (родительского потока)
330
331     // Родительский поток ждет завершения дочерних потоков
332     pthread_join(tidp, NULL);
333     pthread_join(tidg, NULL);
334     pthread_join(tidv, NULL);
335
336     // Очистка памяти семафоров
337     sem_destroy(&(lg.sem));
338     sem_destroy(&(lg.sem));
339
340     return 0; // Завершение программы
341 }
```

---