ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С. А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «СТЕК И ОЧЕРЕДЬ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

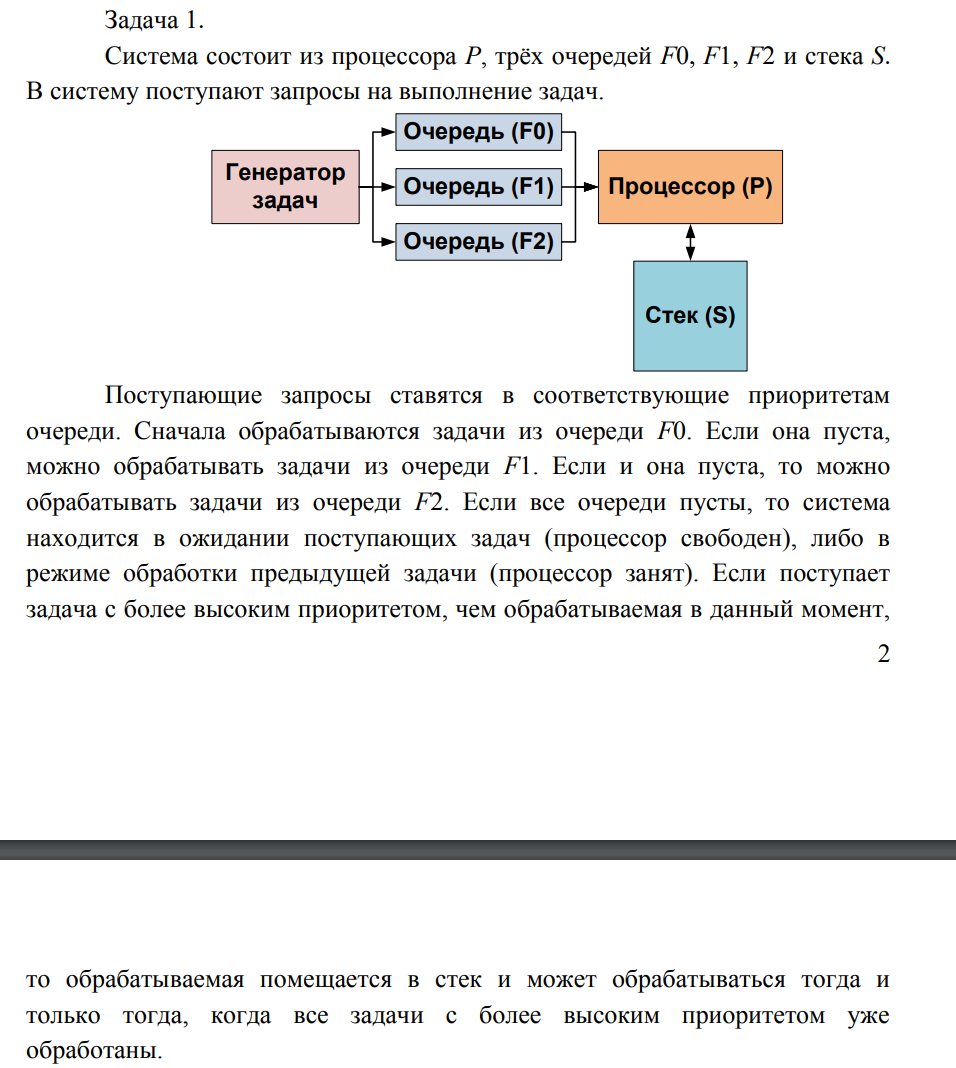
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4131 |  |  |  | Д. А. Кузнецов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Цель работы:** Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

**Вариант:** 4





**Листинг программы:**

**Processor.h**

1. #pragma once
2. #include <iostream>
3. #include <windows.h>
4. #include "TaskManager.h"
6. class Processor {
7. Task task;
8. bool isProcessing = false;
10. public:
11. bool isEmpty() {
12. return (isProcessing == 0 ? true : false);
13. }
15. void show() {
16. if (isProcessing) {
17. TaskManager::show(task);
18. }
19. else
20. cout << "Процессор свободен.\n";
21. }
23. Task getTask() {
24. return task;
25. }
27. void setTask(Task obj) {
28. SYSTEMTIME st;
29. GetSystemTime(&st);
31. obj.arrival\_time = (to\_string(st.wHour + 3) + ":" + to\_string(st.wMinute) +
32. ":" + to\_string(st.wSecond) + ":" + to\_string(st.wMilliseconds));
34. task = obj;
35. isProcessing = true;
36. }
38. void process() {
39. if (task.execution\_time > 0) {
40. task.execution\_time -= 1;
42. if (task.execution\_time == 0)
43. isProcessing = false;
44. }
45. else {
46. isProcessing = false;
47. task.priority = 1000;
48. }
49. }
50. };

**Queue.h**

1. #pragma once
3. #include <string>
4. #include <iostream>
5. #include "TaskManager.h";
7. using namespace std;
9. class Queue {
11. struct Node {
12. Task data;
13. Node\* next = nullptr;
14. Node\* previous = nullptr;
15. };
17. Node\* out = new Node;
18. Node\* in = out;
20. bool is\_empty = 1;
22. public:
23. void push(Task value);
24. Task read();
25. void clear();
26. void show();
27. bool isEmpty();
28. int getPriority();
30. ~Queue();
31. };

**Queue.cpp**

1. #include <iostream>
2. #include <string>
4. #include "Queue.h"
5. #include "TaskManager.h"
7. using namespace std;
9. void Queue::push(Task value) {
11. if (in->previous == nullptr && (in->data).arrival\_time == "-1") {
12. in->data = value;
13. }
14. else {
15. Node\* obj = new Node;
16. obj->data = value;
17. in->next = obj;
18. obj->previous = in;
19. in = obj;
20. }
22. is\_empty = 0;
23. }
25. Task Queue::read() {
27. Task temp\_data = out->data;
28. Node\* temp\_obj = out;
30. if (out->next == nullptr) {
31. out = new Node;
32. in = out;
33. is\_empty = 1;
34. }
35. else {
36. out = out->next;
37. }
39. delete temp\_obj;
40. return temp\_data;
41. }
43. void Queue::show() {
45. Node\* pointer = out;
47. if ((pointer->data).arrival\_time == "-1" && pointer->next == nullptr) {
48. cout << "Очередь пуста!\n\n";
49. return;
50. }
52. while (pointer != nullptr) {
53. TaskManager::show(pointer->data);
54. pointer = pointer->next;
55. }
56. cout << "\n";
57. }
59. void Queue::clear() {
60. Node\* pointer = out;
62. while (pointer != nullptr) {
63. Node\* temp = pointer;
64. pointer = pointer->next;
66. delete temp;
67. }
69. in = nullptr;
70. out = nullptr;
72. is\_empty = 1;
73. }
75. int Queue::getPriority() {
76. Task temp = out->data;
78. return (temp.priority);
79. }
81. bool Queue::isEmpty() {
82. return is\_empty;
83. }
85. Queue::~Queue() {};

**Stack.h**

1. #pragma once
3. #include <string>
4. #include <iostream>
5. #include "Task.h"
7. using namespace std;
9. class Stack {
11. struct Node {
12. Task data;
13. Node\* next = nullptr;
14. };
16. Node\* head = new Node;
18. bool is\_empty = 1;
20. public:
21. void push(Task value);
22. Task read();
23. int getPriority();
24. void clear();
25. void show();
26. bool isEmpty();
28. ~Stack();
29. };

**Stack.cpp**

1. #include <iostream>
2. #include <string>
4. #include "Stack.h"
5. #include "TaskManager.h"
7. using namespace std;
9. void Stack::push(Task value) {
10. if ((head->next == nullptr) && (head->data).arrival\_time == "-1") {
11. head->data = value;
12. }
13. else {
14. Node\* obj = new Node;
15. obj->data = value;
16. obj->next = head;
17. head = obj;
18. }
20. is\_empty = 0;
21. }
23. Task Stack::read() {
24. Task temp = head->data;
25. Node\* temp\_pointer = head;
27. if (head->next == nullptr) {
28. head = nullptr;
30. is\_empty = 1;
31. }
32. else {
33. head = head->next;
34. }
35. delete temp\_pointer;
36. return temp;
37. }
39. int Stack::getPriority() {
40. Task temp = head->data;
42. return (temp.priority);
43. }
45. void Stack::show() {
46. if (is\_empty) {
47. cout << "Стек пуст.\n";
48. return;
49. }
51. Node\* pointer = head;
53. while (pointer->next != nullptr) {
54. TaskManager::show(pointer->data);
56. pointer = pointer->next;
57. }
59. TaskManager::show(pointer->data);
60. cout << "\n\n";
61. }
63. void Stack::clear() {
64. Node\* pointer = head;
66. while (pointer != nullptr) {
67. Node\* temp = pointer;
68. pointer = pointer->next;
70. delete temp;
71. }
73. is\_empty = 1;
74. head = nullptr;
75. }
77. bool Stack::isEmpty() {
78. return is\_empty;
79. }
81. Stack::~Stack() {};

**Task.h**

1. #pragma once
2. #include <string>
4. struct Task {
5. int priority = 1000;
6. int execution\_time;
7. std::string arrival\_time = "-1";
8. };

**TaskManager.h**

1. #pragma once
2. #include "Task.h"
3. #include <chrono>
4. #include <string>
6. bool getRandom();
8. class TaskManager {
10. public:
11. static Task generateTask();
12. static Task createTask();
13. static void show(Task task);
14. };

**TaskManager.cpp**

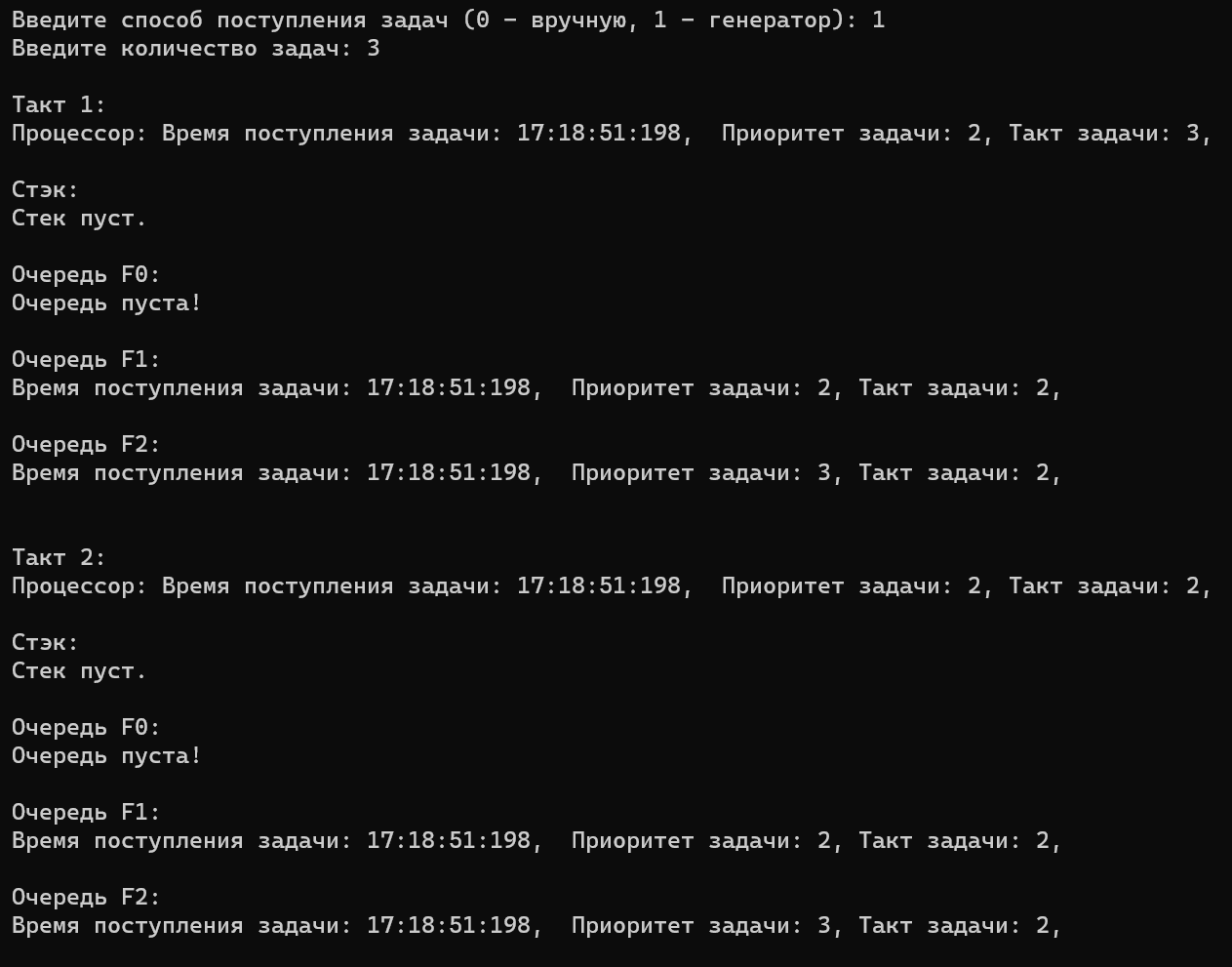
1. #include <iostream>
2. #include <windows.h>
3. #include <string>
4. #include "TaskManager.h"
6. using namespace std;
8. bool getRandom() {
9. double i;
10. i = rand() % 100;
12. return (i < 70 ? true : false);
13. }
15. string get\_Time() {
16. SYSTEMTIME st;
17. GetSystemTime(&st);
19. return ((to\_string(st.wHour + 3) + ":" + to\_string(st.wMinute) + ":" + to\_string(st.wSecond) + ":" + to\_string(st.wMilliseconds)));
20. }
22. Task TaskManager::generateTask() {
23. Task task;
25. task.execution\_time = 1 + rand() % 4;
27. int i;
28. i = 1 + rand() % 3;
30. switch (i)
31. {
32. case 1:
33. task.priority = 1;
34. break;
35. case 2:
36. task.priority = 2;
37. break;
38. case 3:
39. task.priority = 3;
40. break;
41. }
43. task.arrival\_time = get\_Time();
45. return task;
46. }
48. Task TaskManager::createTask() {
49. Task task;
51. int value;
53. while (true) {
54. cout << "Введите время исполнения задачи (число от 1 до 4): "; cin >> value; cout << endl;
55. task.execution\_time = value;
57. cout << "Введите приоритет задачи (число от 1 до 3): "; cin >> value; cout << endl;
58. task.priority = value;
60. if ((task.execution\_time > 4 || task.execution\_time < 1) ||
61. (task.priority > 3 || task.priority < 1)) {
62. cout << "Некорректные данные, повторите ввод!" << endl;
63. continue;
64. }
66. break;
67. }
69. task.arrival\_time = get\_Time();
71. return task;
72. }
74. void TaskManager::show(Task task) {
75. cout << "Время поступления задачи: " << task.arrival\_time << ", ";
76. cout << "Приоритет задачи: " << task.priority << ", ";
77. cout << "Такт задачи: " << task.execution\_time << ", \n";
78. }

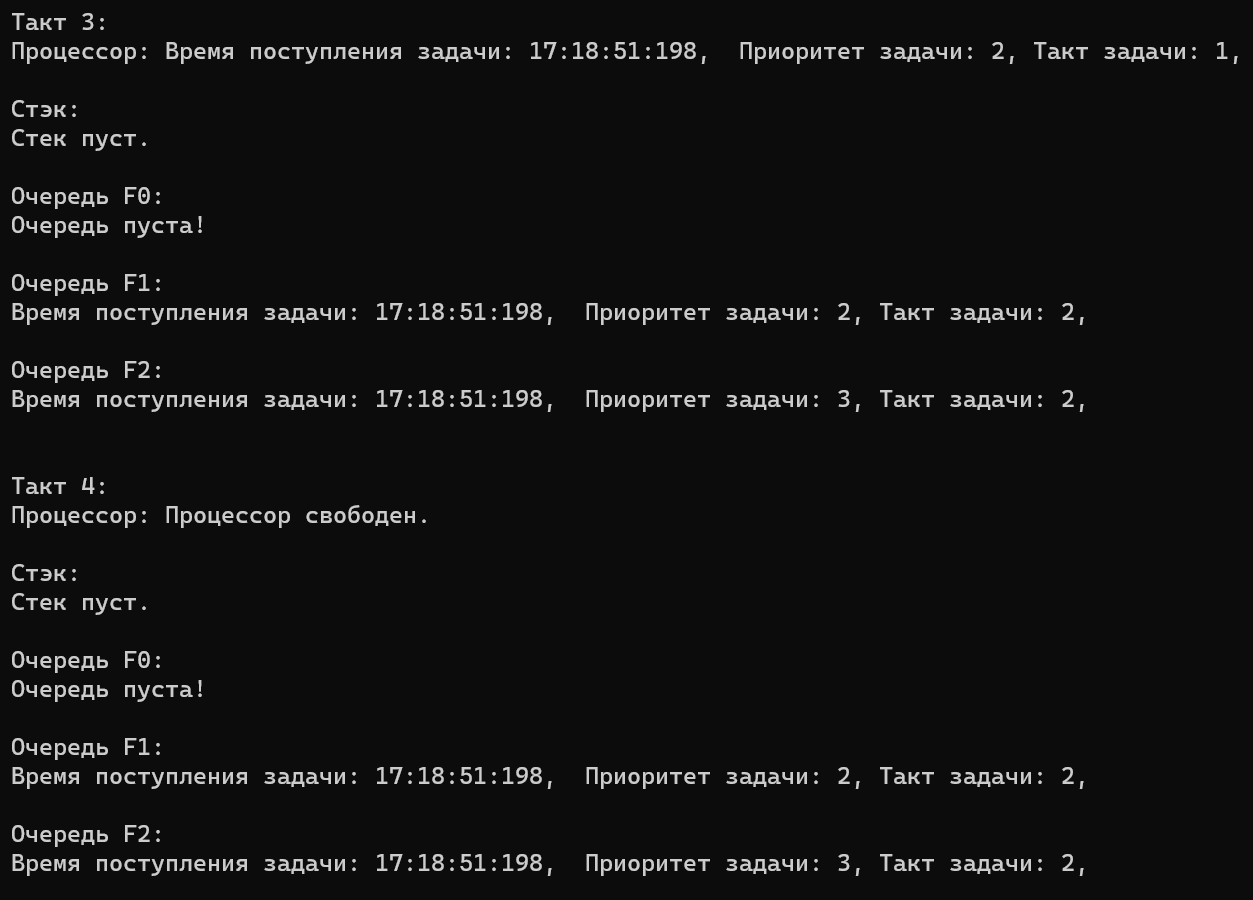
**Main.cpp**

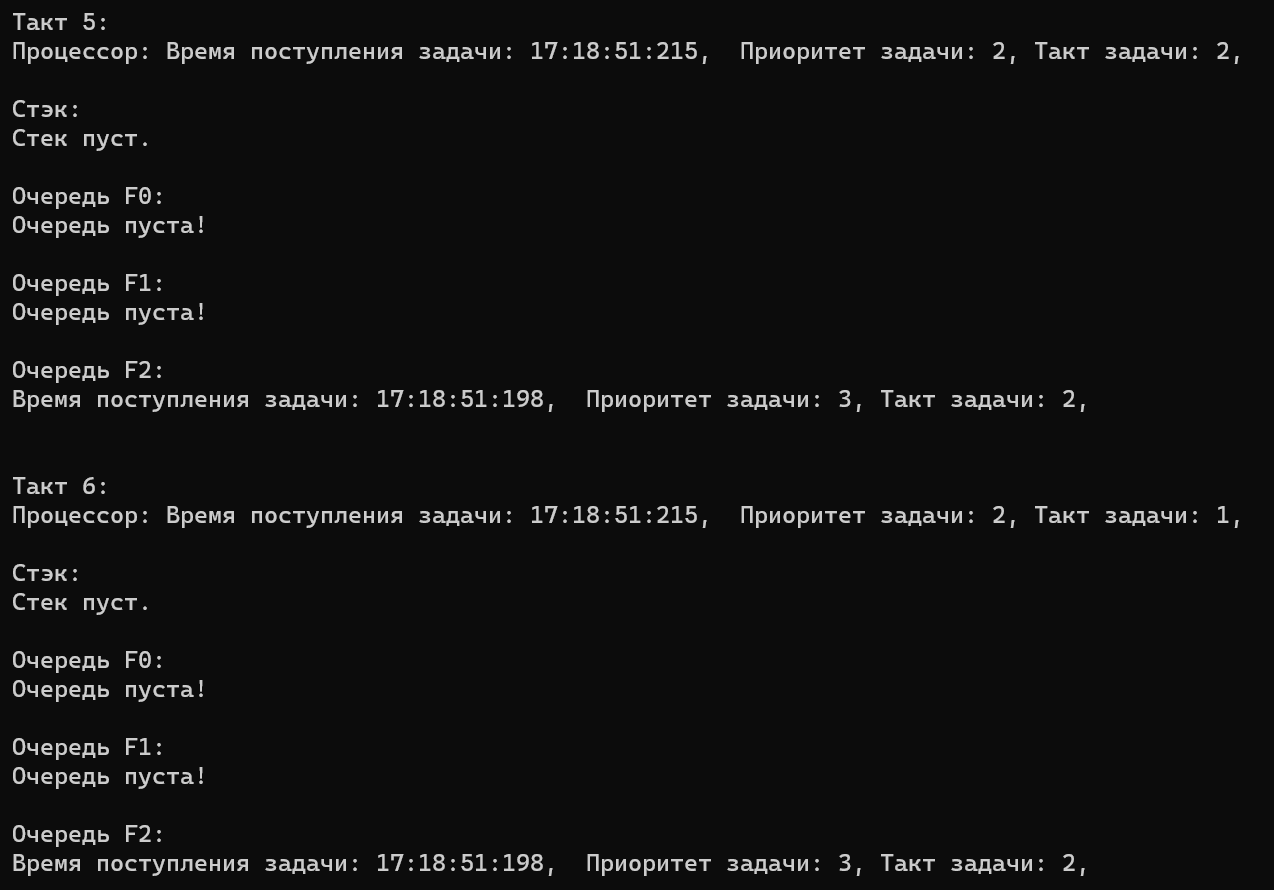
1. // Var 4
3. #define \_CRTDBG\_MAP\_ALLOC
4. #include <stdlib.h>
5. #include <crtdbg.h>
6. #include <chrono>
7. #include <iostream>
8. #include <string>
10. #include "Stack.h"
11. #include "Queue.h"
12. #include "Task.h"
13. #include "TaskManager.h"
14. #include "Processor.h"
16. using namespace std;
18. string get\_time();
20. int main() {
21. setlocale(LC\_ALL, "rus");
23. Processor processor;
24. Stack stack;
26. Queue F0;
27. Queue F1;
28. Queue F2;
30. Queue task\_list;
32. int checker;
34. // Время поступления и последний вывод
36. // Check input option (!!! Export to single method !!!)
37. while (true) {
38. cout << "Введите способ поступления задач (0 - вручную, 1 - генератор): ";
39. cin >> checker;
41. if (checker != 1 && checker != 0) {
42. cout << "\nНекорректные данные, повторите ввод.\n";
43. continue;
44. }
45. break;
46. }
48. if (checker == 0) {
49. int r;
50. cout << "Введите количество задач: "; cin >> r;
52. for (int i = 0; i < r; i++) {
53. Task task;
54. task = TaskManager::createTask();
55. task\_list.push(task);
56. }
57. }
58. else {
59. int r;
60. cout << "Введите количество задач: "; cin >> r;
62. for (int i = 0; i < r; i++) {
63. Task task;
64. task = TaskManager::generateTask();
65. task\_list.push(task);
66. }
67. }
68. // Check input option (!!! Export to single method !!!)

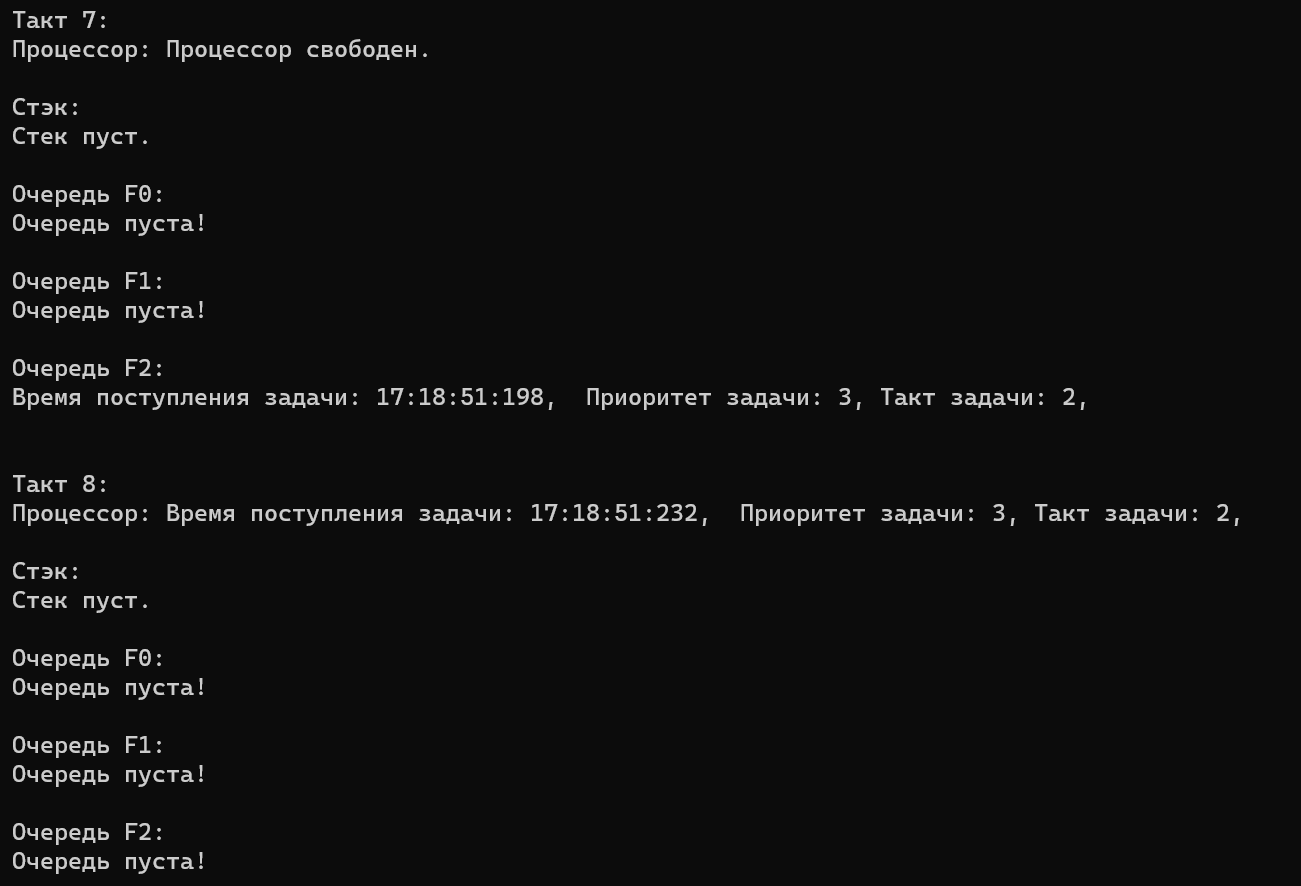

72. int iterator = 1;
74. // Start processing
75. while (true) {
77. cout << "\nТакт " + to\_string(iterator) + ":\n";
79. // Pushing random amount if new task from task list by 50% chance
80. // 1 + ( std::rand() % 4)
81. int changer = 1 + (std::rand() % 4);
83. for (int k = 0; k < changer; k++) {
84. if (task\_list.isEmpty())
85. break;
87. if (getRandom()) {
88. Task task = task\_list.read();
90. task.arrival\_time = get\_time();
92. switch (task.priority)
93. {
94. case 1:
95. F0.push(task);
96. break;
97. case 2:
98. F1.push(task);
99. break;
100. case 3:
101. F2.push(task);
102. break;
103. }
104. }
105. }
107. int stack\_active\_priority;
109. if (!stack.isEmpty()) {
110. stack\_active\_priority = stack.getPriority();
111. }
112. else {
113. stack\_active\_priority = 1000;
114. }
116. // Task distribution
117. Task active\_task = processor.getTask();
119. if (stack\_active\_priority == 1 && stack\_active\_priority < active\_task.priority) {
120. processor.setTask(stack.read());
122. if (active\_task.priority != 1000)
123. stack.push(active\_task);
124. }
125. else if (!F0.isEmpty() && (F0.getPriority() < active\_task.priority)) {
126. processor.setTask(F0.read());
128. if (active\_task.priority != 1000)
129. stack.push(active\_task);
130. }
131. else if (stack\_active\_priority == 2 && stack\_active\_priority < active\_task.priority) {
132. processor.setTask(stack.read());
134. if (active\_task.priority != 1000)
135. stack.push(active\_task);
136. }
137. else if (!F1.isEmpty() && (F1.getPriority() < active\_task.priority)) {
138. processor.setTask(F1.read());
140. if (active\_task.priority != 1000)
141. stack.push(active\_task);
142. }
143. else if (stack\_active\_priority == 3 && stack\_active\_priority < active\_task.priority) {
144. processor.setTask(stack.read());
146. if (active\_task.priority != 1000)
147. stack.push(active\_task);
148. }
149. else if (!F2.isEmpty() && (F2.getPriority() < active\_task.priority)) {
150. processor.setTask(F2.read());
152. if (active\_task.priority != 1000)
153. stack.push(active\_task);
154. }
156. cout << "Процессор: "; processor.show(); cout << "\n";
158. cout << "Стэк: \n";
159. stack.show();
161. cout << "\nОчередь F0: \n";
162. F0.show();
163. cout << "Очередь F1: \n";
164. F1.show();
165. cout << "Очередь F2: \n";
166. F2.show();
168. processor.process();
169. iterator += 1;
171. if (processor.isEmpty() && F0.isEmpty() && F1.isEmpty() && F2.isEmpty() &&
172. stack.isEmpty() && task\_list.isEmpty()) {
174. cout << "\nТакт " + to\_string(iterator) + ":\n";
175. cout << "Процессор: "; processor.show(); cout << "\n";
177. cout << "Стэк: \n";
178. stack.show();
180. cout << "\nОчередь F0: \n";
181. F0.show();
182. cout << "Очередь F1: \n";
183. F1.show();
184. cout << "Очередь F2: \n";
185. F2.show();
187. break;
188. }
190. }

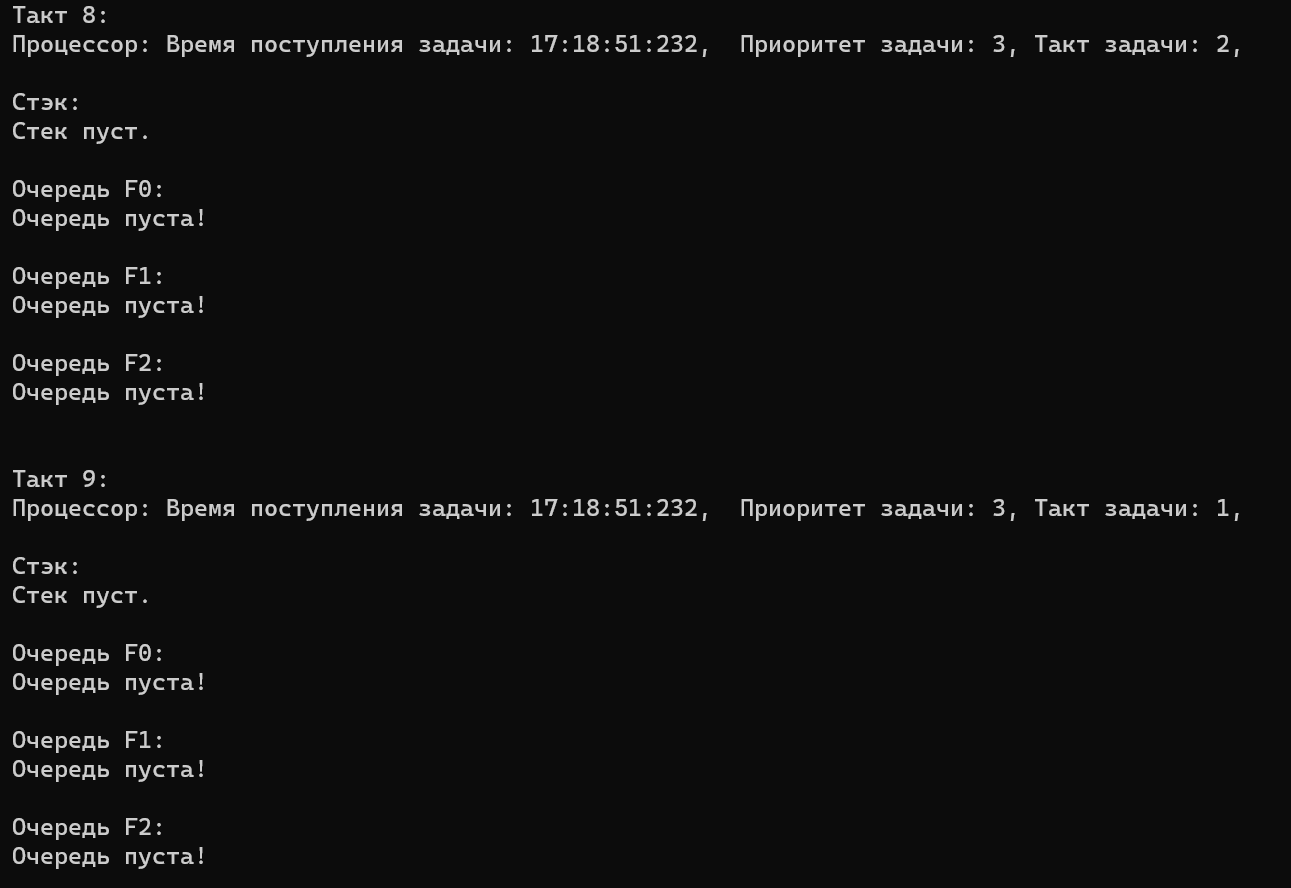
193. stack.clear();
194. F0.clear();
195. F1.clear();
196. F2.clear();
198. \_CrtDumpMemoryLeaks();
199. return 0;
200. }
202. string get\_time() {
203. SYSTEMTIME st;
204. GetSystemTime(&st);
206. return ((to\_string(st.wHour + 3) + ":" + to\_string(st.wMinute) + ":" + to\_string(st.wSecond) + ":" + to\_string(st.wMilliseconds)));
207. }

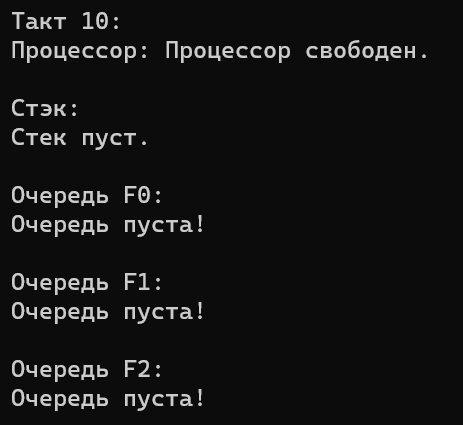
**Контрольные примеры:**

****

****

****

****

****

**Вывод:**

В ходе лабораторной работы я изучил структуры данных «стек» и «очередь», а также получил практические навыки их реализации.