ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| 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. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

* 1. Цель работы

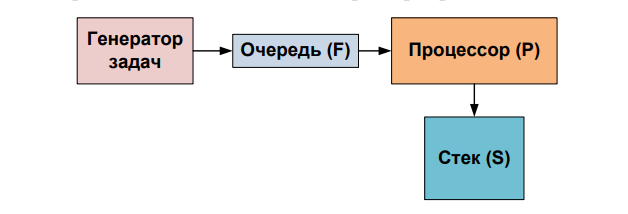
Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

* 1. Задание на лабораторную работу Реализовать структуры данных «стек» и «очередь» в соответствии с заданным вариантом. Дополнительно программа должна удовлетворять следующим требованиям: 1) Вывод на экран состояния моделируемой системы на каждой итерации работы (содержимое стека(ов), очереди(ей), процессора(ов)); 2) Для каждой задачи из списка входных задач должно быть определено время поступления; 3) Необходимо наличие, как автоматического генератора задач, так и возможность ручного добавления задач, с указанием их параметров (в зависимости от задания); 4) Необходимо обработать ситуации, при которых какая-либо структура данных может быть переполнена.

**Вариант 3**



Система состоит из процессора P, очереди F и стека S. В систему поступают запросы на выполнение задач трех приоритетов.



Поступающие запросы ставятся в очередь. Если очередь пуста, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны Генератор задач должен формировать запросы на выполнение задач. В общем случае эти запросы на выполнение задач со случайным приоритетом и случайной длительности должны поступать в случайные моменты времени. Однако, для демонстрации всех свойств системы, необходимо определенная комбинация задач с заданными характеристиками. Поэтому при сдаче лабораторной студенту будет необходимо продемонстрировать работу системы на заранее запланированных примерах.

**Ход выполнения**

**Листинг**

#include <iostream>

class stack

{

public:

stack();

~stack();

void add(int tick);

int get\_tick();

void del();

int is\_empty();

void show();

class element

{

public:

int tick;

element\* next;

element(int tick=0, element\* ptr = nullptr) { this->tick = tick; next = ptr; };

~element();

void set(int tick) { this->tick = tick; };

};

private:

element\* top = nullptr;

};

void stack::del()

{

element\* current = top;

for (int i = 0; i < 10; i++)

{

current->tick = current->next->tick;

current = current->next;

}

current->tick = 0;

}

int stack::is\_empty()

{

if (top->tick==0)

{

return 1;

}

else

{

return 0;

}

}

void stack::show()

{

{

element\* current = top;

int i = 1;

while (current->next != nullptr and i<10)

{

std::cout << "stack Tasks ticks " << current->tick << ", priority: " << i<<"\n";

i++;

current=current->next;

}

}

}

int stack::get\_tick() {

if (top->tick==0)

{

return 0;

}

element\* current = top;

return current->tick;

}

stack::stack()

{

for (int i = 0; i < 10; i++)

{

this->add(0);

}

}

stack::~stack()

{

}

void stack::add(int tick)

{

element\* el = new element(tick);

element\* current = top;

if (top != nullptr)

{

el->next = top;

top = el;

}

else

{

top = el;

}

}

class queue

{

public:

queue(int arr[10]);

~queue();

void add(int tick);

void del();

int get\_tick();

void solve();

int is\_empty();

void show();

class element

{

public:

element(int tick =0, element\* ptr = nullptr) { this->tick = tick; next = ptr; };

~element();

void set(int tick) { this->tick = tick; };

int tick;

element\* next;

};

private:

element\* now = nullptr;

};

queue::queue(int arr[10])

{

for (int i = 0; i < 10; i++)

{

add(arr[i]);

}

}

queue::~queue()

{

}

void queue::add(int tick)

{

if (now==nullptr)

{

element\* el = new element(tick);

now = el;

}

else

{

element\* cur = now;

while (cur->next !=nullptr)

{

cur = cur->next;

}

element\* el = new element(tick);

cur->next = el;

}

}

void queue::del()

{

element\* cur = now;

cur = cur->next;

now = cur;

}

int queue::get\_tick()

{

if (now==nullptr)

{

return 0;

}

return now->tick;

}

void queue::solve()

{

now->tick = now->tick - 1;

}

int queue::is\_empty()

{

if (now==nullptr)

{

return 1;

}

else

{

return 0;

}

}

void queue::show()

{

element\* current = now;

int i = 1;

if (current == nullptr) return;

while (current->next!=nullptr)

{

std::cout << "queue Tasks ticks " << current->tick << ", priority: " << i<<"\n";

i++;

current = current->next;

}

}

class processor

{

public:

processor(int num) { name = num; };

~processor();

void do\_tick();

void add\_task(int time);

int get\_status();

int get\_task() { return task; };

int havedone() { return done; };

private:

int task;

int tick=0;

bool free=1;

int name;

int done;

};

processor::~processor()

{

}

void processor::do\_tick()

{

if (free)

{

std::cout << "PC" << name << " is free" << std::endl;

done = 0;

}

else

{

std::cout << "PC" << name << " Is doing task. Ticks befor complete: " << tick << std::endl;

tick = tick - 1;

done = 0;

}

if (tick<=0 && free==0)

{

std::cout <<"PC"<<name << " Task is Done\n";

free = 1;

done = 1;

}

}

void processor::add\_task(int time)

{

task = time;

tick = time;

free = 0;

std::cout << "PC" << name << " recieved task. Ticks befor complete: " << tick<<'\n';

}

int processor::get\_status()

{

return free;

}

int main()

{

int tasks[10];

tasks[0] = 10;

tasks[1] = 2;

for (int i = 2; i < 10; i++)

{

tasks[i] = 1;

}

std::cout << "Do you want to set your own tasks y/n\n";

char inp;

std::cin >> inp;

if (inp=='y')

{

std::cout << "enter 10 values: ";

for (int i = 0; i < 10; i++)

{

std:: cin >> tasks[i];

}

}

for (int i = 0; i < 10; i++)

{

std::cout << tasks[i] << "\t";

}

std::cout<< "\n";

queue hosp(tasks);

stack done;

processor PC0(0);

processor PC1(1);

int timer = 1;

while (true)

{

try

{

std::cout << "-------tick number " << timer <<"-------" << '\n';

hosp.show();

done.show();

if (PC0.havedone() == 1)

{

done.add(PC0.get\_task());

std::cout << "Added new task " << PC0.get\_task() << " in stack. time of adjust " << timer << "\n";

}

if (PC0.get\_status() == 1)

{

if (hosp.get\_tick() != 0)

{

int task = hosp.get\_tick();

PC0.add\_task(task);

hosp.del();

}

else

{

std::cout << "Hosp is empty\n";

}

}

if (PC1.get\_status() == 1)

{

if (done.get\_tick() > 0)

{

int task = done.get\_tick();

PC1.add\_task(task);

done.del();

}

std::cout << "Done is empty\n";

}

PC0.do\_tick();

PC1.do\_tick();

timer++;

std::cout << "\n";

std::cin.get();

}

catch (const std::exception&)

{

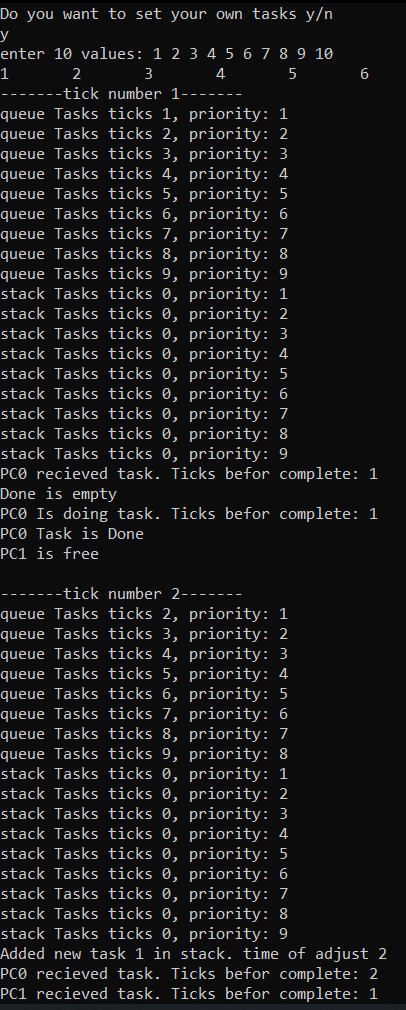
continue;

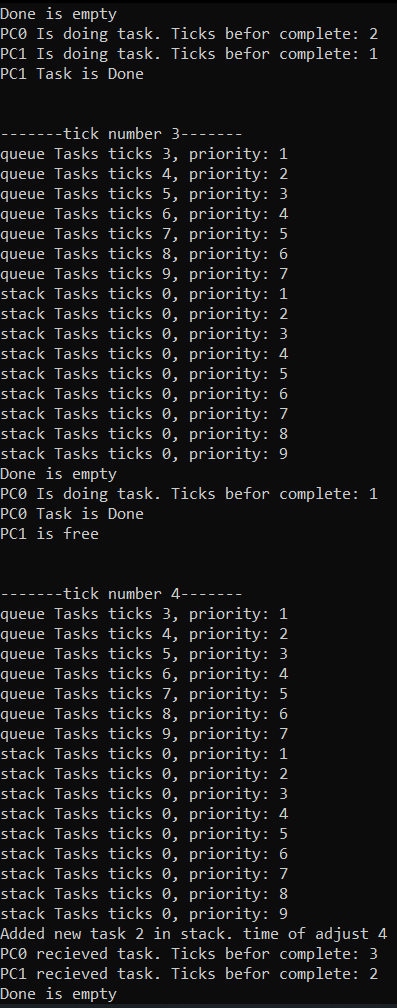
}

}

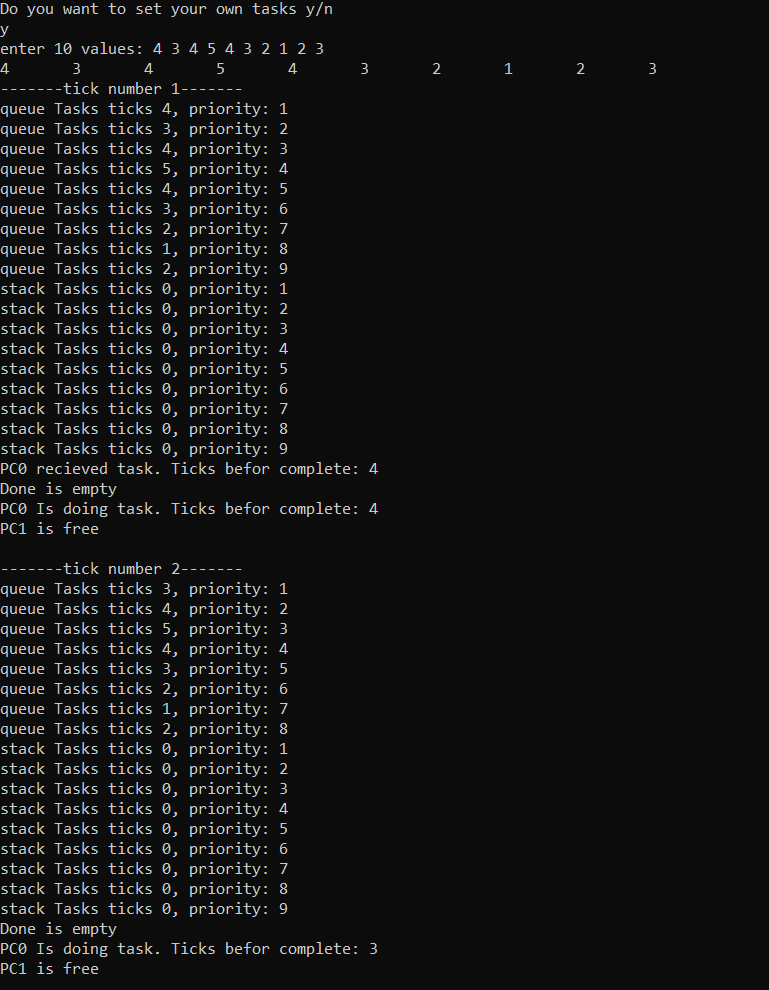
}

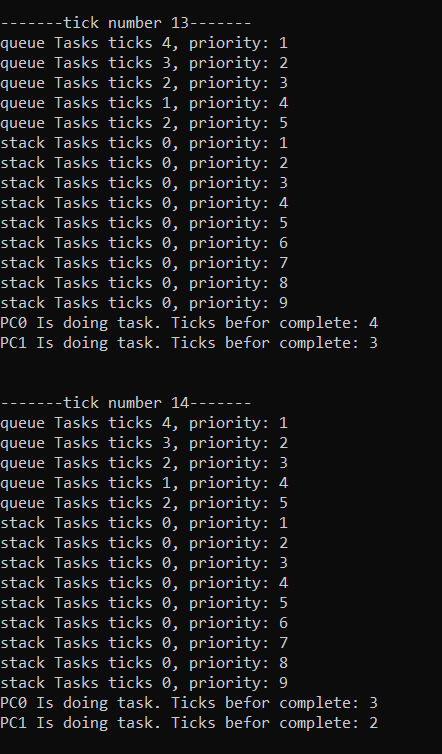
**Результат работы**



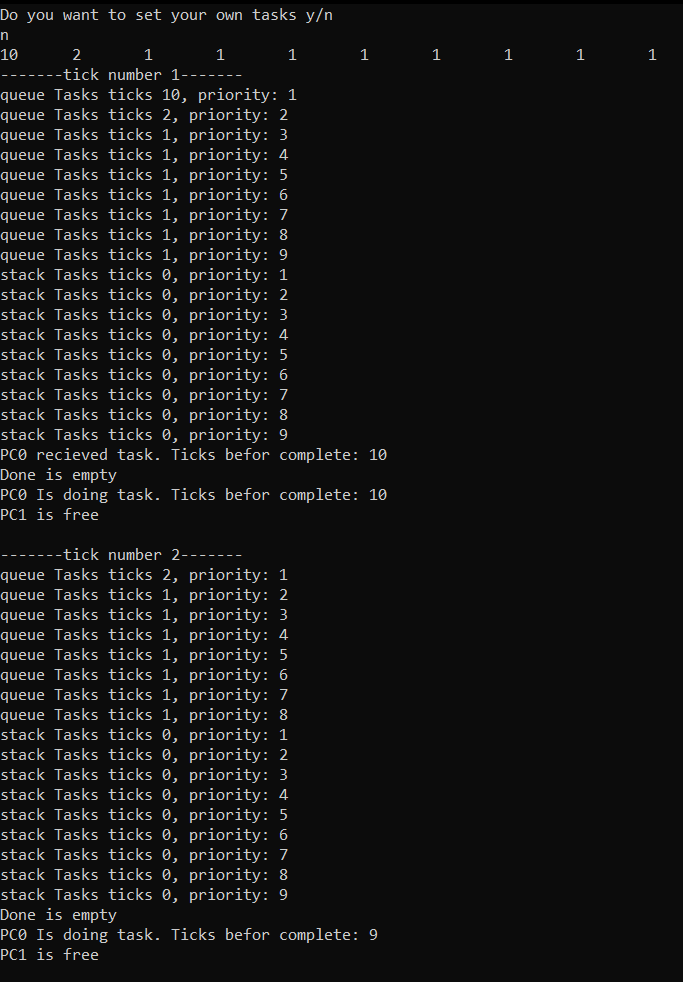


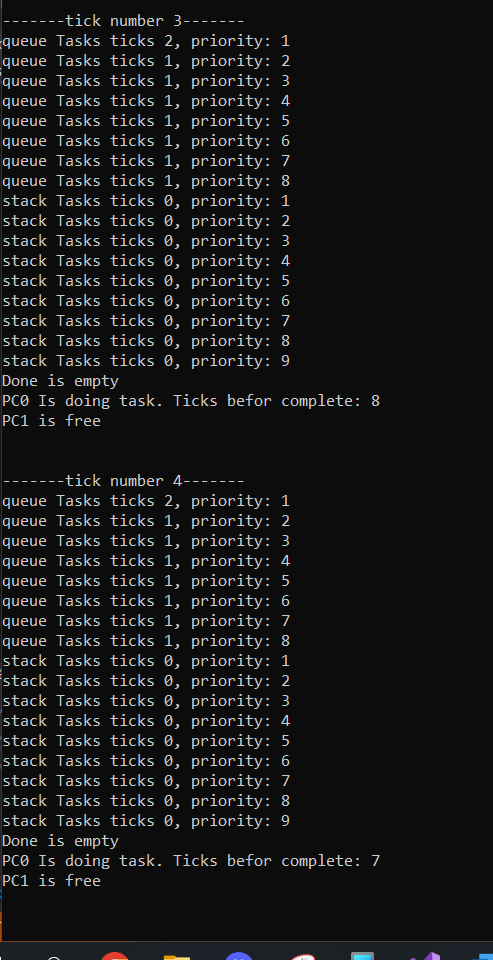
**Результат работы 2**





**Результат работы 3 с генератором**





**Вывод я изучил работу со стеком и очередью и разработал свою систему обработки задач**