

## Отчёт по лабораторной работе №7–8

- **Задание**

Целью лабораторной работы является изучение принципов работы со связанными списками, а также реализация класса на языке C++ для управления списком блоков управления процессами (Process Control Block, PCB).

- **Описание структуры программы**

Программа состоит из следующих основных компонентов:

1. Структура PCB, содержащая информацию о процессе.
2. Класс ListNode, представляющий узел связанного списка.
3. Класс ProcessList, реализующий операции управления списком процессов.

### **Структура PCB:**

- PCB() — конструктор структуры процесса

- processName: Название процесса (std::string).
- processStatus: Статус процесса (например, Running, Waiting, Stopped).
- commandCounter: Счетчик команд (int), указывающий на следующую исполняемую команду.
- cpuRegisters: Массив или структура для хранения состояния регистров CPU

### **Класс ListNode:**

- ListNode(const PCB& data) — конструктор узла списка
- ~ListNode() — деструктор узла (при необходимости)

### **Класс ProcessList:**

- ProcessList() — конструктор списка процессов
- ~ProcessList() — деструктор, освобождающий память
- bool insert(const PCB& newPCB) — вставка процесса в список
- bool remove(int processID) — удаление процесса по идентификатору

- void printList() const — вывод списка процессов

### Функция main():

- Создание объекта ProcessList
- Добавление процессов
- Вывод списка
- Удаление процессов
- Демонстрация обработки ошибок Описание работы программы

В функции main создаётся объект класса ProcessList. Далее в список добавляется несколько процессов с различными идентификаторами. После этого выполняется вывод списка для проверки корректности вставки.

Затем производится удаление одного или нескольких процессов, после чего список выводится повторно.

- Код :

#### main.cpp

```
#include <iostream>
#include "ProcessList.h"
#include <limits>
int vvod_and_durak()
{
    int a;
    while (true)
    {
        std::cin >> a;
        if (std::cin.fail() == true)
        {
            std::cin.clear();
            std::cin.ignore(1000, '\n');
            std::cout << "Wrong input. Please enter a
number: ";
        }
        else{
            std::cin.ignore(1000, '\n');
            return a;
        }
    }
}
int main()
```

```

{
    ProcessList function;
    int choice;

    while (true)
    {
        std::cout << "    PROCESS MANAGER    " << std::endl;
        std::cout << "1. Add process" << std::endl;
        std::cout << "2. Remove process" << std::endl;
        std::cout << "3. Show list" << std::endl;
        std::cout << "0. Exit" << std::endl;
        std::cout << "Your choice: " << std::endl;
        choice = vvod_and_durak();

        if (choice == 0)
        {
            std::cout << "Program was closed." <<
std::endl;
            break;
        }
        switch (choice)
        {
            case 1:
            {
                PCB newPCB;
                int statusIn;

                std::cout << "--- Add Process ---" <<
std::endl;
                std::cout << "Enter Id: ";
                newPCB.processID = vvod_and_durak();

                std::cout << "Enter Name: ";
                std::cin >> newPCB.processName;

                std::cout << "What Status(Run,Wait or Stop
(0,1,2) for each other): ";
                statusIn = vvod_and_durak();

                if (statusIn < 0 || statusIn > 2) {
                    std::cout << "    Error! Invalid status
they must be 0, 1 or 2" << std::endl;
                    break;
                }
                newPCB.status = (processStatus)statusIn;
            }
        }
    }
}

```

```
        std::cout << "Enter command counter  
(number): ";  
        newPCB.commandCounter = vvod_and_durak();  
  
        for(int i=0; i<4; i++)  
        {  
            newPCB.cpuRegisters[i] = 0;  
        }  
        if (function.insert(newPCB) == true)  
        {  
            std::cout << "Success, process added" <<  
std::endl;  
        } else  
        {  
            std::cout << "Error, process with this  
ID already exists" << std::endl;  
        }  
        break;  
    }  
    case 2:  
    {  
        int delId;  
        std::cout << "Remove Process" <<  
std::endl;  
        std::cout << "Enter ID for removing: ";  
        delId = vvod_and_durak();  
        if (function.remove(delId))  
        {  
            std::cout << "Process removed." <<  
std::endl;  
        } else  
        {  
            std::cout << "Error! Process with  
ID " << delId << " was not found" << std::endl;  
        }  
        break;  
    }  
    case 3:  
    {  
        function.printList();  
        break;  
    }  
    default:  
        std::cout << "Wrong command. Try again"  
<< std::endl;
```

```
        }
    }
    return 0;
}
```

ProcessList.cpp

```
#include "ProcessList.h"
#include <iostream>
```

```
ProcessList::ProcessList()
```

```
{    head = nullptr;
}
```

```
ProcessList::~ProcessList()
```

```
{    ListNode* current = head;
    while (current!=nullptr)
    {
        ListNode* del = current;
        current = current->next;
        delete del;
    }
}
```

```
bool ProcessList::insert(const PCB& newPCB)
```

```
{    ListNode* newOne = new ListNode(newPCB);
    ListNode* current = head;
    if (head != nullptr and head->data.processID ==
newPCB.processID)
    {
        return false;
    }
    if (head == nullptr or (head->data.processID) >
newPCB.processID)
    {
        newOne->next = head;
        head = newOne;
        return true;
    }
    while (current->next!=nullptr and current->next-
>data.processID < newPCB.processID)
    {

```

```

        current = current ->next;
    }

    if (current->next != nullptr and current->next-
>data.processID == newPCB.processID)
    {
        delete newOne;
        return false;
    }
    newOne->next = current -> next;
    current->next = newOne;

    return true;
}
bool ProcessList::remove(int id)
{
    if (head == nullptr)
    {
        return false;
    }

    if (head->data.processID == id)
    {
        ListNode* del = head;
        head = head -> next;
        delete del;
        return true;
    }

    ListNode* current = head;

    while (current->next and current->next-
>data.processID != id)
    {
        current = current->next;
    }
    if (current -> next == nullptr) return false;
    ListNode* del = current->next;
    current->next = del -> next;
    delete del;
    return true;
}

void ProcessList::printList()
{
    if (head == nullptr)

```

```

    {
        std::cout << "No process" << std::endl;
        return;
    }
    std::cout << "--- List of processes ---" << std::endl;
    ListNode* current = head;
    while (current != nullptr)
    {
        std::cout << "ID: " << current->data.processID <<
std::endl;
        std::cout << "Name: " << current->data.processName
<< std::endl;
        std::cout << "Status: " << current-
>data.whatStatusIs() << std::endl;
        std::cout << "Command: " << current-
>data.commandCounter << std::endl;
        current = current->next;
    }
}

```

ProcessList.h

```

#ifndef ProcessListh
#define ProcessListh
#include <iostream>
#include <string>

enum processStatus
{
    Running, //типо 0
    Waiting, // 1
    Stopped // 2
};
struct PCB {int processID;std::string processName ;
    processStatus status; int commandCounter;
    int cpuRegisters[4];

    std::string whatStatusIs() const
    {
        switch (status)
        {
            case Running : return "Running";
            case Waiting : return "Waiting";
            case Stopped : return "Stopped";
            default : return "HZ";
        }
    }
}

```

```
        }
    }
};

class ListNode
{
public :
    PCB data;
    ListNode* next;
    ListNode(PCB pcb)
    {
        data = pcb;
        next = nullptr;
    }
};

class ProcessList
{
private:
    ListNode* head;
public :
    ProcessList();
    ~ProcessList();

    bool insert(const PCB& newPCB);
    bool remove(int pid);
    void printList();
};

#endif
```

#### 4. Ссылка на репозиторий:

<https://github.com/nikgolich-ship-it/Nikita-Golev-SCB252.git>