## Описание задания (задание 5, вариант 21):

21. Задача об инвентаризации по рядам. После нового года в библиотеке университета обнаружилась пропажа каталога. После поиска и наказания виноватых, ректор дал указание восстановить каталог силами студентов. Фонд библиотека представляет собой прямоугольное помещение, в котором находится M рядов по N шкафов по K книг в каждом шкафу. Требуется создать многопоточное приложение, составляющее каталог. При решении задачи использовать метод «портфель задач», причем в качестве отдельной задачи задается составление каталога одним студентом для одного ряда.

## Исполнитель: Сизов Владислав Игоревич БПИ202

Разработать консольное приложение с использованием библиотеки POSIX Threads языка программирования C или стандартной библиотеки языка программирования C++.

Главная сущность в программе – библиотека. Она определяется количеством рядов, шкафов в ряду и количеством книг в шкафу. На каждой полке в шкафу помещается 10 книг.

Количество книг в библиотеке = количество рядов * количество шкафов * количество книг в шкафу.

Это число не должно превышать 10^9, иначе программа будет хранится очень долго.

Цель программы – составить каталог всех книг в библиотека, для каждой указать номер ряда, номер шкафа и номер полки на которой она хранится.

В качестве модели построения многопоточного приложения используется портфель задач.

Также в программе можно выбрать время, которое тратится на составление каталога для книг из одного ряда для наглядной демонстрации пользы потоков (ускорение времени работы).

## Модель приложения

Взаимодействующие равные – модель, в которой исключен не занимающийся непосредственными вычислениями управляющий поток.

Распределение работ в таком приложении либо фиксировано заранее, либо динамически определяется во время выполнения. Одним из распространенных способов динамического распределения работ является «портфель задач». Портфель задач, как правило, реализуется с помощью разделяемой переменной, доступ к которой в один момент времени имеет только один процесс.

Вычислительная задача делится на конечное число подзадач. Как правило, каждая подзадача должна выполнить однотипные действия над разными данными. Подзадачи нумеруются, и каждому номеру определяется функция, которая однозначно отражает номер задачи на соответствующий ему набор данных. Создается переменная, которую следует выполнять следующей. Каждый поток сначала обращается к портфелю задач для выяснения текущего номера задачи, после этого увеличивает его, потом берет соответствующие данные и выполняет задачу, затем обращается к портфелю задач для выяснения следующего номера задачи.

**Составление библиотеки случайными книгами:**

Inventory – разделяемая переменная. Каждый поток получает ряд, который должен формировать и генерирует его случайным образом.

**Составление каталога:**

Result – разделяемая переменная. Каждый поток получает ряд, из которого должен составить каталог.

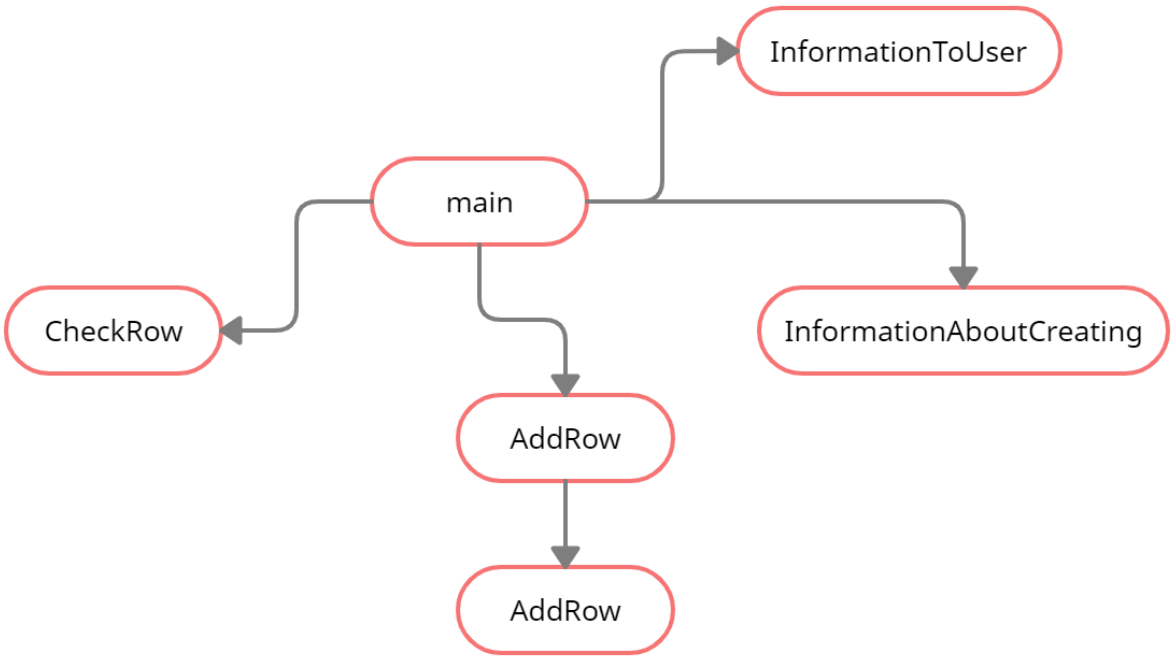
**СТРУКТУРНАЯ СХЕМА АРХИТЕКТУРЫ ВС С ПРОГРАММОЙ:**

Программа разработана в 64 битной системе (дистрибутив Ubuntu на ярде Linux).

Таблица типов

| Название | Размер |
|---|---|
| int | 4 |
| Class Inventory:<br><br>rowsCount: int<br>boxesCount: int<br>`booksCount`: int<br>`curRow`: int<br>`timeToSleep`: int<br>`result`: **int<br>`inventory`: ***Box<br>mtx:Mutex | 20 + 4*rowsCount*boxesCount*booksCount + rowsCount*boxesCount*(16 + 4*totalBooks)<br>4<br>4<br>4<br>4<br>4<br>4*rowsCount*boxesCount*booksCount<br>rowsCount*boxesCount*(16 + 4*totalBooks) |
| Class Box:<br>`countOfBooks`: int<br>`sizeOfShelf`: int<br>`countOfShelf`: int<br>`totalBooks`: int<br>`books`: **int | 16 + 4*totalBooks<br>4<br>4<br>4<br>4<br>4 * totalBooks |

**Блок схема возможного стека, в результате работы функции main (с глубиной в 1 шаг):**



**Описание работы для функции FillBox в рамках архитектуры:**

| Stack |
|---|
| booksLeft |
| numOfBook |
| countOfBooks |
| countOfShelf |
| sizeofShelf |
| totalBooks |
| books |

**Память программы**

```
25
26  void Box::FillBox() {
27      int booksLeft = this->countOfBooks;
28      for (int i = 0; i < this->countOfShelf; ++i) {
29          for (int j = 0; j < this->sizeOfShelf; ++j) {
30              if (booksLeft > 0) {
31                  int numOfBook = std::rand() % this->totalBooks + 1;
32                  books[i][j] = numOfBook;
33                  booksLeft--;
34              } else {
35                  books[i][j] = -1;
36              }
37          }
38      }
39  }
```

| Heap |
|---|
| |
| |
| |

**Описание работы для функции CheckRow в рамках архитектуры:**

| Stack |
|---|
| rowsPos |
| boxesCount |
| countOfShelf |
| sizeOfShelf |
| books |
| timeToSleep |

**Память программы**

```
44  void Inventory::CheckRow(int rowsPos) {
45      for (int i = 0; i < this->boxesCount; ++i) {
46          for (int j = 0; j < inventory[rowsPos][i]->countOfShelf; ++j) {
47              for (int k = 0; k < inventory[rowsPos][i]->sizeOfShelf; ++k) {
48                  int indexOfBook = inventory[rowsPos][i]->books[j][k] - 1;
49                  if (indexOfBook == -2) {
50                      continue;
51                  }
52                  this->result[indexOfBook][0] = rowsPos;
53                  this->result[indexOfBook][1] = i;
54                  this->result[indexOfBook][2] = j;
55              }
56          }
57      }
58      std::this_thread::sleep_for( rtime: std::chrono::milliseconds (this->timeToSleep));
59      mtx.lock();
60      std::cout << "\nСтудент составил каталог книг из ряда номер: " << rowsPos + 1 << ";";
61      mtx.unlock();
62  }
```

| Heap |
|---|
| mtx |
| |

**Описание работы потоков в рамках архитектуры:**

Сначала программа запускает потоки для создания библиотеки, затем дожидается выполнения всех потоков и начинает составлять каталог.

| **Stack** |
| --- |
| rowStarted |
| |
| |
| |
| |
| |

| **Память программы** |
| --- |
|  |
|  |
| |

| **Heap** |
| --- |
| Threads[] |
| |

# Основные характеристики программы:

**Число заголовочных файлов** – 2

**Число модулей реализации** – 3

**Общий размер исходных текстов** – 265 строки

**Время работы программы:**

1 тест (1 1 1) – 0.001 секунды

2 тест (5 4 3) - 0.001 секунды

3 тест (10 10 10) - 0.003 секунды

4 тест (100 100 100) - 0.813 секунды

5 тест (1000 1000 100) – 86.982 секунды

**Размер кода:** 8.144 килобайт