ИДЗ-4 Кокорев Артём
Вариант 21

Файлы расположены в публичном репозитории: https://github.com/w1sq/ABC/

Задание:
Задача о нелюдимых садовниках. Имеется пустой участок земли (двумерный массив размером M × N) и план сада, разбитого на отдельные квадраты. От 10 до 30 процентов (задается случайно) площади сада заняты прудами или камнями. То есть недоступны для ухаживания. Эти квадраты располагаются на плане произвольным (случайным) образом. Ухаживание за садом выполняют два садовника, которые не хотят встречаться друг другом (то есть, одновременно появляться в одном и том же квадрате). Первый садовник начинает работу с верхнего левого угла сада и перемещается слева направо, сделав ряд, он спускается вниз и идет в обратном направлении, пропуская обработанные участки. Второй садовник начинает работу с нижнего правого угла сада и перемещается снизу вверх, сделав ряд, он перемещается влево и также идет в обратную сторону. Если садовник видит, что участок сада уже обработан другим садовником или является необрабатываемым, он идет дальше. Если по пути какой-то участок занят другим садовником, то садовник ожидает когда участок освободится, чтобы пройти дальше на доступный ему необработанный участок. Садовники должны работать одновременно со скоростями, определяемыми как параметры задачи. Прохождение через любой квадрат занимает некоторое время, которое задается константой, меньшей чем времена обработки и принимается за единицу времени. Создать многопоточное приложение, моделирующее работу садовников. Каждый садовник — это отдельный поток.

Код на 8 баллов:
main.cpp

```cpp
#include <mutex>
#include <vector>
#include <thread>
#include <random>
#include <fstream>
#include <sstream>
#include <iostream>
#include <condition_variable>

std::mutex garden_mutex, start_mutex;
std::condition_variable cv, start_cond;

bool ready_to_start = false;
```

```
// Function to generate a random number in a given range
int generateRandomNumber(int min, int max)
{
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<> dis(min, max);
return dis(gen);
}

// Function to fill the garden with rocks
void fillGardenWithRocks(int garden_height, int garden_width,
std::vector<std::vector<int>> &garden)
{
int rockCount = generateRandomNumber(10, 30);
while (rockCount > 0)
{
int row = generateRandomNumber(0, garden_height - 1);
int col = generateRandomNumber(0, garden_width - 1);
if (garden[row][col] == 0)
{
garden[row][col] = -1; // Place a rock
rockCount--;
}
}
}

class Gardener
{
public:
int id;
int garden_height;
int garden_width;
int walking_speed;
int working_speed;
std::vector<std::vector<int>> &garden;

Gardener(int id, int garden_height, int garden_width, int
walking_speed, int working_speed, std::vector<std::vector<int>>
&garden) : id(id), garden_height(garden_height),
garden_width(garden_width), walking_speed(walking_speed),
working_speed(working_speed), garden(garden) {}
```

```
void operator()()
{
// Wait for the start
{
std::unique_lock<std::mutex> lk(start_mutex);
start_cond.wait(lk, []
{ return ready_to_start; });
}

{
std::unique_lock<std::mutex> lk(garden_mutex);
cv.wait(lk, []
{ return ready_to_start; });
}

if (id == 1)
{
processGarden(0, garden_height, 1); // First gardener starts from the
top
}
else
{
processGarden(garden_height - 1, -1, -1); // Second gardener starts
from the bottom
}
}

private:
void processGarden(int start, int end, int step)
{
bool reverse = false;
if (id == 1) // First gardener moves horizontally
{
for (int i = start; i != end; i += step)
{
int j_start = reverse ? garden_width - 1 : 0;
int j_end = reverse ? -1 : garden_width;
int j_step = reverse ? -1 : 1;

for (int j = j_start; j != j_end; j += j_step)
{
waitAndProcess(i, j);
}
```

```
reverse = !reverse;
}
}
else // Second gardener moves vertically
{
for (int j = garden_width - 1; j >= 0; j--)
{
int i_start = reverse ? 0 : garden_height - 1;
int i_end = reverse ? garden_height : -1;
int i_step = reverse ? 1 : -1;

for (int i = i_start; i != i_end; i += i_step)
{
waitAndProcess(i, j);
}
reverse = !reverse;
}
}
}

void waitAndProcess(int i, int j)
{
while (true)
{
std::unique_lock<std::mutex> lock(garden_mutex);
if (garden[i][j] == 0) // Empty square
{
garden[i][j] = id;
lock.unlock();
std::cout << "Gardener " << id << " processing square at (" << i << ",
" << j << ")\n";
std::this_thread::sleep_for(std::chrono::milliseconds(working_speed));
return;
}
else if (garden[i][j] == -1 || garden[i][j] > 0) // Rock or processed
square
{
std::this_thread::sleep_for(std::chrono::milliseconds(walking_speed));
return;
}
// If the square is temporarily occupied by another gardener, wait
cv.wait(lock);
}
```

```
cv.notify_all();
}
};

void printGarden(std::ofstream &ofs, std::vector<std::vector<int>>
&garden)
{
for (const auto &row : garden)
{
for (const auto &cell : row)
{
ofs << (cell == -1 ? "X" : std::to_string(cell)) << " "; // X for
rocks
}
ofs << "\n";
}
}

int main(int argc, char *argv[])
{
int garden_height;
int garden_width;
int walking_speed;
int first_gardener_working_speed;
int second_gardener_working_speed;
std::string input_file;
std::string output_file;

if (argc == 7)
{
// Read all parameters from the command line
garden_height = std::stoi(argv[1]);
garden_width = std::stoi(argv[2]);
walking_speed = std::stoi(argv[3]);
first_gardener_working_speed = std::stoi(argv[4]);
second_gardener_working_speed = std::stoi(argv[5]);
output_file = std::string(argv[6]);
}
else if (argc == 2)
{
input_file = argv[1];
std::ifstream ifs(input_file);
if (!ifs.is_open())
```

```
{
std::cerr << "Error opening file for reading." << std::endl;
return 1;
}

// Check if the file content is valid
if (!(ifs >> garden_height >> garden_width >> walking_speed >>
first_gardener_working_speed >> second_gardener_working_speed >>
output_file))
{
std::cerr << "Error: Invalid file content. Expected 5 digits and
output file name." << std::endl;
ifs.close();
return 1;
}
}
else
{
std::cerr << "Usage: " << argv[0]
<< " <garden_height> <garden_width> <walking_speed> "
<< "<first_gardener_working_speed> <second_gardener_working_speed>
<output_file>"
<< "\nOR\n"
<< argv[0] << " <input_file>" << std::endl;
return 1;
}

std::vector<std::vector<int>> garden(garden_height,
std::vector<int>(garden_width, 0)); // 0 - empty, 1 - gardener 1, 2 -
gardener 2, -1 - rock

// Fill the garden with rocks
fillGardenWithRocks(garden_height, garden_width, garden);

// Initialize the output file
std::ofstream ofs(output_file);
if (!ofs.is_open())
{
std::cerr << "Error opening file for writing." << std::endl;
return 1;
}

// Write the initial state of the garden
```

```
ofs << "Initial garden:\n";
printGarden(ofs, garden);

// Create and start gardeners
Gardener gardener1(1, garden_height, garden_width, walking_speed,
first_gardener_working_speed, garden); // Gardener 1
Gardener gardener2(2, garden_height, garden_width, walking_speed,
second_gardener_working_speed, garden); // Gardener 2

std::thread t1(std::ref(gardener1));
std::thread t2(std::ref(gardener2));

// Start the gardeners
{
std::lock_guard<std::mutex> lk(start_mutex);
ready_to_start = true;
start_cond.notify_all();
}

t1.join();
t2.join();

ofs << "Processed garden:\n";
printGarden(ofs, garden);
ofs.close();

return 0;
}
```

Примерный вывод, для первого набора тестовых данных(остальные имеют такую же логику, но слишком большие для отчёта)

[w1sq@archlinux IDZ4]$ ./a.out input1
Initial garden:
0 X 0 0 X 0 0 0 X 0
0 0 0 0 0 0 0 0 0 0
X 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
X 0 X 0 0 0 0 X 0 0
X 0 0 0 0 0 0 0 0 0

Gardener 2 processing square at (9, 9)
Gardener 1 processing square at (0, 0)
Gardener 2 processing square at (8, 9)
Gardener 1 processing square at (0, 2)
Gardener 2 processing square at (7, 9)
Gardener 1 processing square at (0, 3)
Gardener 2 processing square at (6, 9)
Gardener 1 processing square at (0, 5)
Gardener 2 processing square at (4, 9)
Gardener 1 processing square at (0, 6)
Gardener 2 processing square at (2, 9)
Gardener 1 processing square at (0, 7)
Gardener 1 processing square at (0, 9)
Gardener 2 processing square at (1, 9)
Gardener 1 processing square at (1, 8)
Gardener 2 processing square at (2, 8)
Gardener 1 processing square at (1, 7)
Gardener 2 processing square at (3, 8)
Gardener 1 processing square at (1, 6)
Gardener 2 processing square at (5, 8)
Gardener 1 processing square at (1, 5)
Gardener 2 processing square at (6, 8)
Gardener 1 processing square at (1, 4)
Gardener 2 processing square at (7, 8)
Gardener 1 processing square at (1, 3)
Gardener 2 processing square at (8, 8)
Gardener 1 processing square at (1, 2)
Gardener 2 processing square at (9, 8)
Gardener 1 processing square at (1, 1)
Gardener 2 processing square at (9, 7)
Gardener 1 processing square at (1, 0)
Gardener 2 processing square at (7, 7)
Gardener 1 processing square at (2, 1)
Gardener 2 processing square at (6, 7)
Gardener 1 processing square at (2, 2)
Gardener 2 processing square at (5, 7)
Gardener 1 processing square at (2, 3)
Gardener 2 processing square at (4, 7)
Gardener 1 processing square at (2, 5)
Gardener 2 processing square at (3, 7)
Gardener 1 processing square at (2, 6)
Gardener 2 processing square at (2, 7)
Gardener 1 processing square at (3, 6)
Gardener 2 processing square at (4, 6)
Gardener 1 processing square at (3, 5)
Gardener 2 processing square at (5, 6)
Gardener 1 processing square at (3, 4)
Gardener 2 processing square at (6, 6)
Gardener 1 processing square at (3, 3)

Gardener 2 processing square at (7, 6)
Gardener 1 processing square at (3, 2)
Gardener 2 processing square at (8, 6)
Gardener 1 processing square at (3, 1)
Gardener 2 processing square at (9, 6)
Gardener 1 processing square at (3, 0)
Gardener 2 processing square at (9, 5)
Gardener 1 processing square at (4, 0)
Gardener 2 processing square at (8, 5)
Gardener 1 processing square at (4, 1)
Gardener 2 processing square at (7, 5)
Gardener 1 processing square at (4, 2)
Gardener 2 processing square at (6, 5)
Gardener 1 processing square at (4, 3)
Gardener 2 processing square at (5, 5)
Gardener 1 processing square at (4, 4)
Gardener 2 processing square at (4, 5)
Gardener 1 processing square at (5, 4)
Gardener 2 processing square at (6, 4)
Gardener 1 processing square at (5, 3)
Gardener 2 processing square at (7, 4)
Gardener 1 processing square at (5, 2)
Gardener 2 processing square at (8, 4)
Gardener 1 processing square at (5, 1)
Gardener 2 processing square at (9, 4)
Gardener 1 processing square at (5, 0)
Gardener 2 processing square at (9, 3)
Gardener 1 processing square at (6, 0)
Gardener 2 processing square at (8, 3)
Gardener 1 processing square at (6, 1)
Gardener 2 processing square at (7, 3)
Gardener 1 processing square at (6, 2)
Gardener 2 processing square at (6, 3)
Gardener 1 processing square at (7, 2)
Gardener 2 processing square at (9, 2)
Gardener 1 processing square at (7, 1)
Gardener 2 processing square at (9, 1)
Gardener 1 processing square at (7, 0)
Gardener 2 processing square at (8, 1)
Processed garden:
1 X 1 1 X 1 1 1 X 1
1 1 1 1 1 1 1 1 1 2
X 1 1 1 X 1 1 2 2 2
1 1 1 1 1 1 1 2 2 X
1 1 1 1 1 2 2 2 X 2
1 1 1 1 1 2 2 2 2 X
1 1 1 2 2 2 2 2 2 2
1 1 1 2 2 2 2 2 2 2
X 2 X 2 2 2 2 X 2 2

X 2 2 2 2 2 2 2 2 2

Сценарий поведения сущностей

В саду прямоугольной формы работают два садовника, которые должны обработать все свободные участки. На участках сада случайным образом расположены камни, которые нельзя обработать. Садовники имеют следующие характеристики и правила поведения:

- Первый садовник начинает работу с верхней части сада и движется горизонтально (змейкой).
- Второй садовник начинает с нижней части сада и движется вертикально (змейкой).
- Каждый садовник имеет свою скорость передвижения и скорость обработки участков.
- Садовники не могут обрабатывать один и тот же участок одновременно.
- При встрече с камнем или уже обработанным участком садовник просто проходит мимо.

Модель параллельных вычислений

В программе используется модель параллельных вычислений на основе потоков (threads) с общей памятью. Основные компоненты:

- Потоки: каждый садовник реализован как отдельный поток выполнения
- Мьютексы: используются для синхронизации доступа к общим ресурсам (саду)
- Условные переменные: применяются для координации начала работы и взаимодействия садовников
- Общая память: сад представлен как разделяемая матрица

Входные данные программы

Программа принимает следующие параметры:

1. Размеры сада:
   - Высота (garden_height): положительное целое число
   - Ширина (garden_width): положительное целое число

2. Временные параметры:
   - Скорость передвижения (walking_speed): в миллисекундах
   - Скорость обработки для первого садовника (first_gardener_working_speed): в миллисекундах
   - Скорость обработки для второго садовника (second_gardener_working_speed): в миллисекундах

Имя выходного файла для сохранения результатов

Параметры могут быть заданы:
- Непосредственно через аргументы командной строки
- Через входной файл с параметрами

Генераторы случайных чисел

В программе используется генератор случайных чисел для размещения камней:

1. Количество камней
   - Диапазон: от 10 до 30 камней
   - Генератор: std::mt19937 с равномерным распределением
   - Интерпретация: определяет общее количество препятствий в саду

2. Размещение камней
   - Диапазон: от 0 до (размер_сада - 1) для обеих координат
   - Генератор: std::mt19937 с равномерным распределением
   - Интерпретация: определяет случайные координаты для размещения камней

5. Обобщенный алгоритм реализации

Инициализация:
1. Создание пустого сада (матрица)
2. Случайное размещение камней
3. Создание двух потоков-садовников

Реализация садовников (класс Gardener):
1. Каждый садовник реализован как объект класса с собственными параметрами:
   - Идентификатор (id)
   - Скорость передвижения (walking_speed)
   - Скорость работы (working_speed)

2. Алгоритм работы садовника:
   а. Ожидание сигнала начала работы
   б. Движение по заданной траектории:
      - Первый садовник: горизонтальное движение змейкой сверху вниз
      - Второй садовник: вертикальное движение змейкой справа налево
   в. Для каждого участка:
      - Проверка доступности участка
      - Обработка если участок свободен
      - Пропуск если встречен камень или обработанный участок

Синхронизация:
- Использование мьютекса для доступа к саду
- Условные переменные для координации начала работы
- Ожидание завершения работы обоих садовников

Вывод результатов:
- Запись начального состояния сада
- Запись конечного состояния сада
- Вывод промежуточных сообщений о работе садовников