Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Операционные системы реального времени

Лабораторная работа №3

Электростанция

Выполнил:

студент гр.ИП-213 Дмитриев Антон Александрович

ФИО студента

«__» _____________ 2025 г.

Проверил:

Преподаватель

ФИО преподавателя

«__» _____________ 2025 г. Оценка__________________

Новосибирск 2025 г.

1. Электростанция состоит из следующих элементов: хранилище топлива (1 шт.), транспортное средство (1 шт.), котлы (4 шт.). Элементы станции работают параллельно, каждый по своей программе (что может быть реализовано с помощью нитей). Транспортное средство доставляет топливо из хранилища к котлам. Топливо имеет различные марки (от 1 до 10). Топливо марки 10 горит в котле 10 с (условно), в то время как топливо марки 1 горит всего 1 с. Необходимо написать программу, моделирующую работу электростанции и показывающую на экране процесс ее функционирования.

```
#include <vingraph.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <cmath>

class Boiler;

struct BurnData {
    Boiler* boiler;
    std::vector<int> fuelBlocks;
    BurnData(Boiler* b, const std::vector<int>& fb) : boiler(b), fuelBlocks(fb) {}
};

class Storage {
public:
    Storage(int x, int y) : x(x), y(y) {
        pthread_mutex_init(&mtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(0, 100, 200));
        fill = Rect(x+2, y+2, 96, 76, 0, RGB(0, 200, 100));
        text = Text(x + 15, y + 35, "FUEL", RGB(255, 255, 255));
        Show(rect);
        Show(fill);
        Show(text);
    }
    int getFuelType() {
        pthread_mutex_lock(&mtx);
        int fuelType = rand() % 10 + 1;
        printf("[Storage] issued fuel type %d\n", fuelType);
        pthread_mutex_unlock(&mtx);
        return fuelType;
    }
    int getX() const { return x; }
    int getY() const { return y; }
private:
    int x, y;
    int rect, fill, text;
    pthread_mutex_t mtx;
};

class Boiler {
public:
    Boiler(int id, int x, int y, int capacity) : id(id), x(x), y(y), capacity(capacity),
currentFuel(0), burnThread(0) {
        pthread_mutex_init(&burnMtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(200, 150, 0));
        statusRect = Rect(x+2, y+2, 96, 76, 0, RGB(255, 255, 100));

        char title[20];
        sprintf(title, "BOILER %d", id);
        text1 = Text(x + 25, y + 20, title, RGB(0, 0, 0));
        text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));

        Show(rect);
        Show(statusRect);
        Show(text1);
        Show(text2);
    }

    ~Boiler() {
        if (burnThread) {
            pthread_join(burnThread, NULL);
        }
        pthread_mutex_destroy(&burnMtx);
    }

    void fillAndBurn(int fuelType) {
        pthread_mutex_lock(&burnMtx);
        if (currentFuel == 0) {
            currentFuel = capacity;
            currentFuelType = fuelType;

            Delete(text2);
            text2 = Text(x + 25, y + 50, "BURNING", RGB(255, 100, 0));
            Show(text2);

            std::vector<int> fuelBlocks;
            for (int i = 0; i < capacity; i++) {
                int block = Rect(x + 80, y + 10 + (i * 8), 15, 6, 0, RGB(200, 50, 50));
                Show(block);
                fuelBlocks.push_back(block);
                usleep(200000);
            }

            usleep(500000);

            BurnData* data = new BurnData(this, fuelBlocks);
            pthread_create(&burnThread, NULL, burnHelper, data);
            Delete(text2);
            text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
            Show(text2);
        } else {
            Delete(text2);
            text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
            Show(text2);
        }
        pthread_mutex_unlock(&burnMtx);
    }

    static void* burnHelper(void* arg) {
        BurnData* data = static_cast<BurnData*>(arg);
        data->boiler->burnInternally(data->fuelBlocks);
        delete data;
        return NULL;
    }

    void burnInternally(const std::vector<int>& fuelBlocks) {
        pthread_mutex_lock(&burnMtx);
        for (int i = fuelBlocks.size() - 1; i >= 0; i--) {
            Delete(fuelBlocks[i]);
```

```
            usleep(currentFuelType * 100000);
        }
        Delete(text2);
        text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));
        currentFuel = 0;
        burnThread = 0;
        pthread_mutex_unlock(&burnMtx);
    }

    bool isEmpty() const {
        return currentFuel == 0;
    }

    int getX() const { return x; }
    int getY() const { return y; }

private:
    int id;
    int x, y;
    int rect, statusRect;
    int text1;
    int text2;
    int capacity;
    int currentFuel;
    int currentFuelType;
    pthread_t burnThread;
    pthread_mutex_t burnMtx;
};

class Vehicle {
```

```
public:
   Vehicle(Storage* storage, std::vector<Boiler*>& boilers, int x, int y, int plantRect)
      : storage(storage), boilers(boilers), x(x), y(y), plantRect(plantRect) {
      pthread_mutex_init(&moveMtx, NULL);
   }
   ~Vehicle() {
      pthread_mutex_destroy(&moveMtx);
   }

   static void* runHelper(void* arg) { return ((Vehicle*)arg)->runThread(); }

   void* runThread() {
      int i = 0;
      while (1) {
         int fuelType = storage->getFuelType();
         if (boilers[i]->isEmpty()) {
            moveToBoiler(storage->getX(), storage->getY() - 50, boilers[i]->getX() + 20,
boilers[i]->getY() - 50);
            boilers[i]->fillAndBurn(fuelType);
            moveToStorage(boilers[i]->getX(), boilers[i]->getY() - 50, storage->getX() +
20, storage->getY() - 50);
            usleep(1000000);
         }
         i = (i + 1) % boilers.size();
         usleep(10000);
      }
      return NULL;
   }

private:
   void moveToBoiler(int startX, int startY, int endX, int endY) {
      pthread_mutex_lock(&moveMtx);
      int steps = 30;
      int dx = (endX - startX) / steps;
      int dy = (endY - startY) / steps;
      MoveTo(startX, startY, plantRect);
      for (int step = 1; step <= steps; step++) {
         Move(plantRect, dx, dy);


         Fill(plantRect, RGB(50, 100, 200));
         usleep(15000);
      }
      pthread_mutex_unlock(&moveMtx);
   }

   void moveToStorage(int startX, int startY, int endX, int endY) {
      pthread_mutex_lock(&moveMtx);
      int steps = 30;
      int dx = (endX - startX) / steps;
      int dy = (endY - startY) / steps;
      MoveTo(startX, startY, plantRect);
      for (int step = 1; step <= steps; step++) {
         Move(plantRect, dx, dy);


         Fill(plantRect, RGB(150, 50, 200));
         usleep(15000);
      }
      pthread_mutex_unlock(&moveMtx);
   }

   Storage* storage;
```

```
    std::vector<Boiler*>& boilers;
    int x, y;
    int plantRect;
    pthread_mutex_t moveMtx;
};

int main() {
    ConnectGraph();



    int titleBg = Rect(250, 20, 200, 40, 1, RGB(100, 100, 200));
    int electricPlantText = Text(280, 40, "ELECTRIC PLANT", RGB(255, 255, 255));
    Show(titleBg);
    Show(electricPlantText);



    Storage storage(50, 200);
    Boiler b1(1, 200, 200, 8);
    Boiler b2(2, 350, 200, 8);
    Boiler b3(3, 500, 200, 8);
    Boiler b4(4, 650, 200, 8);

    std::vector<Boiler*> boilers;
    boilers.push_back(&b1);
    boilers.push_back(&b2);
    boilers.push_back(&b3);
    boilers.push_back(&b4);

    int vehicleRect = Rect(20, 20, 25, 15, 0, RGB(50, 100, 200));
    Show(vehicleRect);
    Vehicle vehicle(&storage, boilers, 20, 20, vehicleRect);
    pthread_t vehicleThread;
    pthread_create(&vehicleThread, NULL, Vehicle::runHelper, &vehicle);



    int infoText = Text(50, 350, "Press ESC to exit", RGB(150, 150, 150));
    Show(infoText);

    int key = 0;
    bool running = true;
    while (running) {
        key = InputChar();
        if (key == 27) {
            CloseGraph();
            return 0;
        }
    }

    pthread_join(vehicleThread, NULL);

    CloseGraph();
    return 0;
}
```

2. А теперь добавьте второе транспортное средство.

```
#include <vingraph.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <cmath>

class Boiler;

struct BurnData {
   Boiler* boiler;
   std::vector<int> fuelBlocks;
   BurnData(Boiler* b, const std::vector<int>& fb) : boiler(b), fuelBlocks(fb) {}
};

class Storage {
public:
   Storage(int x, int y) : x(x), y(y) {
      pthread_mutex_init(&mtx, NULL);
      rect = Rect(x, y, 100, 80, 1, RGB(0, 100, 200));
      fill = Rect(x+2, y+2, 96, 76, 0, RGB(0, 200, 100));
      text = Text(x + 15, y + 35, "FUEL", RGB(255, 255, 255));
      Show(rect);
```

```
        Show(fill);
        Show(text);
    }
    int getFuelType() {
        pthread_mutex_lock(&mtx);
        int fuelType = rand() % 10 + 1;
        printf("[Storage] issued fuel type %d\n", fuelType);
        pthread_mutex_unlock(&mtx);
        return fuelType;
    }
    int getX() const { return x; }
    int getY() const { return y; }
private:
    int x, y;
    int rect, fill, text;
    pthread_mutex_t mtx;
};

class Boiler {
public:
    Boiler(int id, int x, int y, int capacity) : id(id), x(x), y(y), capacity(capacity),
currentFuel(0), burnThread(0) {
        pthread_mutex_init(&burnMtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(200, 150, 0));
        statusRect = Rect(x+2, y+2, 96, 76, 0, RGB(255, 255, 100));
    char title[20];

    sprintf(title, "BOILER %d", id);

    text1 = Text(x + 25, y + 20, title, RGB(0, 0, 0));

    text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));


    Show(rect);

    Show(statusRect);

    Show(text1);

    Show(text2);

}


~Boiler() {

    if (burnThread) {

        pthread_join(burnThread, NULL);

    }

    pthread_mutex_destroy(&burnMtx);

}


void fillAndBurn(int fuelType) {

    pthread_mutex_lock(&burnMtx);

    if (currentFuel == 0) {

        currentFuel = capacity;

        currentFuelType = fuelType;
        Delete(text2);
        text2 = Text(x + 25, y + 50, "BURNING", RGB(255, 100, 0));
        Show(text2);

        std::vector<int> fuelBlocks;
        for (int i = 0; i < capacity; i++) {
            int block = Rect(x + 80, y + 10 + (i * 8), 15, 6, 0, RGB(200, 50, 50));
            Show(block);
            fuelBlocks.push_back(block);
            usleep(200000);
        }

        usleep(500000);

        BurnData* data = new BurnData(this, fuelBlocks);
        pthread_create(&burnThread, NULL, burnHelper, data);
        Delete(text2);
        text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
        Show(text2);
    } else {
        Delete(text2);
        text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
        Show(text2);
    }
    pthread_mutex_unlock(&burnMtx);
```

```
}

static void* burnHelper(void* arg) {
   BurnData* data = static_cast<BurnData*>(arg);
   data->boiler->burnInternally(data->fuelBlocks);
   delete data;
   return NULL;
}

void burnInternally(const std::vector<int>& fuelBlocks) {
   pthread_mutex_lock(&burnMtx);
   for (int i = fuelBlocks.size() - 1; i >= 0; i--) {
      Delete(fuelBlocks[i]);
      usleep(currentFuelType * 100000);
   }
   Delete(text2);
   text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));
   Show(text2);
   currentFuel = 0;
   burnThread = 0;
   pthread_mutex_unlock(&burnMtx);
}

bool isEmpty() const {
   return currentFuel == 0;
```

```
    }


    int getX() const { return x; }

    int getY() const { return y; }


private:

    int id;

    int x, y;

    int rect, statusRect;

    int text1;

    int text2;

    int capacity;

    int currentFuel;

    int currentFuelType;

    pthread_t burnThread;

    pthread_mutex_t burnMtx;

};


class Vehicle {

public:

    Vehicle(Storage* storage, std::vector<Boiler*>& boilers, int x, int y, int plantRect,
int vehicleId)

        : storage(storage), boilers(boilers), x(x), y(y), plantRect(plantRect),
vehicleId(vehicleId) {

        pthread_mutex_init(&moveMtx, NULL);

    }
    ~Vehicle() {
        pthread_mutex_destroy(&moveMtx);
    }

    static void* runHelper(void* arg) { return ((Vehicle*)arg)->runThread(); }

    void* runThread() {
        int i = vehicleId;
        while (1) {
            int fuelType = storage->getFuelType();
            printf("[Vehicle %d] carrying fuel type %d to the boiler %d\n", vehicleId + 1,
fuelType, i + 1);
            if (boilers[i]->isEmpty()) {
                moveToBoiler(storage->getX(), storage->getY() - 50, boilers[i]->getX() +
20, boilers[i]->getY() - 50);
                boilers[i]->fillAndBurn(fuelType);
                moveToStorage(boilers[i]->getX(), boilers[i]->getY() - 50, storage->getX()
+ 20, storage->getY() - 50);
                usleep(1000000);
            } else {
                printf("[Vehicle %d] Boiler %d is not empty, skipping\n", vehicleId + 1, i +
1);
            }
            i = (i + 2) % boilers.size();
            usleep(10000);
        }
        return NULL;
    }

private:
    void moveToBoiler(int startX, int startY, int endX, int endY) {
        pthread_mutex_lock(&moveMtx);
        int steps = 30;
        int dx = (endX - startX) / steps;
        int dy = (endY - startY) / steps;
        MoveTo(startX, startY, plantRect);
        for (int step = 1; step <= steps; step++) {
            Move(plantRect, dx, dy);
            Fill(plantRect, vehicleId == 0 ? RGB(50, 100, 200) : RGB(200, 100, 50));
            usleep(15000);
        }
        pthread_mutex_unlock(&moveMtx);
    }

    void moveToStorage(int startX, int startY, int endX, int endY) {
        pthread_mutex_lock(&moveMtx);
        int steps = 30;
        int dx = (endX - startX) / steps;
        int dy = (endY - startY) / steps;
        MoveTo(startX, startY, plantRect);
        for (int step = 1; step <= steps; step++) {
            Move(plantRect, dx, dy);
            Fill(plantRect, vehicleId == 0 ? RGB(150, 50, 200) : RGB(200, 50, 150));
            usleep(15000);
        }
        pthread_mutex_unlock(&moveMtx);
    }

    Storage* storage;
    std::vector<Boiler*>& boilers;
    int x, y;
    int plantRect;
    int vehicleId;
    pthread_mutex_t moveMtx;
};

int main() {
    ConnectGraph();

    int titleBg = Rect(250, 20, 200, 40, 1, RGB(100, 100, 200));
    int electricPlantText = Text(280, 40, "ELECTRIC PLANT", RGB(255, 255, 255));
    Show(titleBg);
    Show(electricPlantText);

    Storage storage(50, 200);
    Boiler b1(1, 200, 200, 8);
    Boiler b2(2, 350, 200, 8);
    Boiler b3(3, 500, 200, 8);
    Boiler b4(4, 650, 200, 8);

    std::vector<Boiler*> boilers;
    boilers.push_back(&b1);
    boilers.push_back(&b2);
    boilers.push_back(&b3);
    boilers.push_back(&b4);

    int vehicleRect1 = Rect(20, 20, 25, 15, 0, RGB(50, 100, 200));
    int vehicleRect2 = Rect(60, 20, 25, 15, 0, RGB(200, 100, 50));
    Show(vehicleRect1);
    Show(vehicleRect2);

    Vehicle vehicle1(&storage, boilers, 20, 20, vehicleRect1, 0);
    Vehicle vehicle2(&storage, boilers, 20, 20, vehicleRect2, 1);
    pthread_t vehicleThread1, vehicleThread2;
    pthread_create(&vehicleThread1, NULL, Vehicle::runHelper, &vehicle1);
    pthread_create(&vehicleThread2, NULL, Vehicle::runHelper, &vehicle2);

    int infoText = Text(50, 350, "Press ESC to exit", RGB(150, 150, 150));
    Show(infoText);

    int key = 0;
    bool running = true;
```

```
    while (running) {
        key = InputChar();
        if (key == 27) {
            CloseGraph();
            return 0;
        }
    }

    pthread_join(vehicleThread1, NULL);
    pthread_join(vehicleThread2, NULL);

    CloseGraph();
    return 0;
}
```

3. (использование импульсов) Регулируя скорости работы элементов электростанции, вы можете создать ситуацию, когда котлы будут простаивать из-за низкой скорости подвоза топлива. Создайте такую ситуацию. Теперь сделайте так, чтобы топливо подвозилось к котлам заранее, до момента их полной остановки. Это можно реализовать, если котлы будут сообщать о том, что топливо скоро кончится (например, его осталось на 2 с работы). Ясно, что котлы могут это сделать с помощью импульса, т.к. обычное сообщение их заблокировало бы, в то время как они должны продолжать работать

```
#include <vingraph.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>

class Boiler;

struct BurnData {
   Boiler* boiler;
   std::vector<int> fuelBlocks;
   BurnData(Boiler* b, const std::vector<int>& fb) : boiler(b), fuelBlocks(fb) {}
};

class Storage {
public:
   Storage(int x, int y) : x(x), y(y) {
        pthread_mutex_init(&mtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(0, 100, 200));
        fill = Rect(x+2, y+2, 96, 76, 0, RGB(0, 200, 100));
        text = Text(x + 15, y + 35, "FUEL", RGB(255, 255, 255));
        Show(rect);
        Show(fill);
        Show(text);
    }
    int getFuelType() {
        pthread_mutex_lock(&mtx);
        int fuelType = rand() % 10 + 1;
        printf("[Storage] issued fuel type %d\n", fuelType);
        pthread_mutex_unlock(&mtx);
        return fuelType;
    }
    int getX() const { return x; }
    int getY() const { return y; }
private:
    int x, y;
    int rect, fill, text;
    pthread_mutex_t mtx;
};

class Boiler {
public:
    Boiler(int id, int x, int y, int capacity) : id(id), x(x), y(y), capacity(capacity), currentFuel(0),
burnThread(0), lowFuel(false) {
        pthread_mutex_init(&burnMtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(200, 150, 0));
        statusRect = Rect(x+2, y+2, 96, 76, 0, RGB(255, 255, 100));
    char title[20];
    sprintf(title, "BOILER %d", id);
    text1 = Text(x + 25, y + 20, title, RGB(0, 0, 0));
    text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));

    Show(rect);
    Show(statusRect);
    Show(text1);
    Show(text2);
}

~Boiler() {
    if (burnThread) {
        pthread_join(burnThread, NULL);
    }
    pthread_mutex_destroy(&burnMtx);
}

void fillAndBurn(int fuelType) {
    pthread_mutex_lock(&burnMtx);
    if (currentFuel == 0) {
        currentFuel = capacity;
        currentFuelType = fuelType;
        printf("[Boiler %d] Filled with %d liters of fuel type %d\n", id, currentFuel, fuelType);

        Delete(text2);
        text2 = Text(x + 25, y + 50, "BURNING", RGB(255, 100, 0));
        Show(text2);

        std::vector<int> fuelBlocks;
        for (int i = 0; i < capacity; i++) {
            int block = Rect(x + 80, y + 10 + (i * 8), 15, 6, 0, RGB(200, 50, 50));
            Show(block);
            fuelBlocks.push_back(block);
            usleep(200000);
        }

        usleep(500000);

        BurnData* data = new BurnData(this, fuelBlocks);
        pthread_create(&burnThread, NULL, burnHelper, data);
        Delete(text2);
        text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
        Show(text2);
    } else {
        Delete(text2);
        text2 = Text(x + 25, y + 50, "WORKING", RGB(0, 150, 0));
        Show(text2);
        printf("[Boiler %d] Already filled with %d liters, skipping fill\n", id, currentFuel);
    }
    pthread_mutex_unlock(&burnMtx);
}

static void* burnHelper(void* arg) {
    BurnData* data = static_cast<BurnData*>(arg);
    data->boiler->burnInternally(data->fuelBlocks);
    delete data;
    return NULL;
}
```

```
void burnInternally(const std::vector<int>& fuelBlocks) {
    pthread_mutex_lock(&burnMtx);
    for (int i = fuelBlocks.size() - 1; i >= 0; i--) {
        if (i == 4) {
            lowFuel = true;
            printf("[Boiler %d] Low fuel signal set (remaining %d blocks)\n", id, i + 1);
        }
        Delete(fuelBlocks[i]);
        usleep(currentFuelType * 100000);
    }
    printf("[Boiler %d] has finished burning %d liters\n", id, currentFuel);
    Delete(text2);
    text2 = Text(x + 25, y + 50, "WAITING", RGB(255, 0, 0));
    Show(text2);
    currentFuel = 0;
    burnThread = 0;
    lowFuel = false;
    pthread_mutex_unlock(&burnMtx);
}

bool isEmpty() const {
    return currentFuel == 0;
}

bool isLowFuel() const {
    pthread_mutex_lock(const_cast<pthread_mutex_t*>(&burnMtx));
    bool result = lowFuel;
    pthread_mutex_unlock(const_cast<pthread_mutex_t*>(&burnMtx));
    return result;
}
```

```
    int getX() const { return x; }
    int getY() const { return y; }

private:
    int id;
    int x, y;
    int rect, statusRect;
    int text1;
    int text2;
    int capacity;
    int currentFuel;
    int currentFuelType;
    pthread_t burnThread;
    pthread_mutex_t burnMtx;
    volatile bool lowFuel;
};

class Vehicle {
public:
    Vehicle(Storage* storage, std::vector<Boiler*>& boilers, int x, int y, int plantRect, int
vehicleId)
        : storage(storage), boilers(boilers), x(x), y(y), plantRect(plantRect), vehicleId(vehicleId) {
        pthread_mutex_init(&moveMtx, NULL);
    }
    ~Vehicle() {
        pthread_mutex_destroy(&moveMtx);
    }

    static void* runHelper(void* arg) { return ((Vehicle*)arg)->runThread(); }
    void* runThread() {
        int i = vehicleId;
        while (1) {
            int fuelType = storage->getFuelType();
            printf("[Vehicle %d] carrying fuel type %d to the boiler %d\n", vehicleId + 1, fuelType, i
+ 1);
            if (boilers[i]->isEmpty() || boilers[i]->isLowFuel()) {
                moveToBoiler(storage->getX(), storage->getY() - 50, boilers[i]->getX() + 20,
boilers[i]->getY() - 50);
                boilers[i]->fillAndBurn(fuelType);
                moveToStorage(boilers[i]->getX(), boilers[i]->getY() - 50, storage->getX() + 20,
storage->getY() - 50);
                usleep(10000);
            } else {
                printf("[Vehicle %d] Boiler %d is not empty, skipping\n", vehicleId + 1, i + 1);
            }
            i = (i + 2) % boilers.size();
            usleep(2000000);
        }
        return NULL;
    }

private:
    void moveToBoiler(int startX, int startY, int endX, int endY) {
        pthread_mutex_lock(&moveMtx);
        int steps = 30;
        int dx = (endX - startX) / steps;
        int dy = (endY - startY) / steps;
        MoveTo(startX, startY, plantRect);
        for (int step = 1; step <= steps; step++) {
            Move(plantRect, dx, dy);
            Fill(plantRect, vehicleId == 0 ? RGB(50, 100, 200) : RGB(200, 100, 50));
            usleep(15000);
        }
        pthread_mutex_unlock(&moveMtx);
    }

    void moveToStorage(int startX, int startY, int endX, int endY) {
        pthread_mutex_lock(&moveMtx);
        int steps = 30;
        int dx = (endX - startX) / steps;
        int dy = (endY - startY) / steps;
        MoveTo(startX, startY, plantRect);
        for (int step = 1; step <= steps; step++) {
            Move(plantRect, dx, dy);
            Fill(plantRect, vehicleId == 0 ? RGB(150, 50, 200) : RGB(200, 50, 150));
            usleep(15000);
        }
        pthread_mutex_unlock(&moveMtx);
    }

    Storage* storage;
    std::vector<Boiler*>& boilers;
    int x, y;
    int plantRect;
    int vehicleId;
    pthread_mutex_t moveMtx;
};

int main() {
    ConnectGraph();
```

```
int titleBg = Rect(250, 20, 200, 40, 1, RGB(100, 100, 200));
int electricPlantText = Text(280, 40, "ELECTRIC PLANT", RGB(255, 255, 255));
Show(titleBg);
Show(electricPlantText);

Storage storage(50, 200);
Boiler b1(1, 200, 200, 8);
Boiler b2(2, 350, 200, 8);
Boiler b3(3, 500, 200, 8);
Boiler b4(4, 650, 200, 8);

std::vector<Boiler*> boilers;
boilers.push_back(&b1);
boilers.push_back(&b2);
boilers.push_back(&b3);
boilers.push_back(&b4);

int vehicleRect1 = Rect(20, 20, 25, 15, 0, RGB(50, 100, 200));
int vehicleRect2 = Rect(60, 20, 25, 15, 0, RGB(200, 100, 50));
Show(vehicleRect1);
Show(vehicleRect2);

Vehicle vehicle1(&storage, boilers, 20, 20, vehicleRect1, 0);
Vehicle vehicle2(&storage, boilers, 20, 20, vehicleRect2, 1);
pthread_t vehicleThread1, vehicleThread2;
pthread_create(&vehicleThread1, NULL, Vehicle::runHelper, &vehicle1);
pthread_create(&vehicleThread2, NULL, Vehicle::runHelper, &vehicle2);

int infoText = Text(50, 350, "Press ESC to exit", RGB(150, 150, 150));
```

```
    Show(infoText);

    int key = 0;
    bool running = true;
    while (running) {
        key = InputChar();
        if (key == 27) {
            CloseGraph();
            return 0;
        }
    }

    pthread_join(vehicleThread1, NULL);
    pthread_join(vehicleThread2, NULL);

    CloseGraph();
    return 0;
}
```

4. А теперь сделайте сетевой вариант разработанной программы. Пусть теперь хранилище работает на одной машине, котлы на другой, а транспортные средства переносят топливо между этими двумя машинами. Вам понадобится разделить вашу программу на две, которые вы будете запускать на разных узлах сети. Чтобы не надоедать соседу с просьбами о запуске вашей программки, которая к тому же будет постоянно подвешивать
его компьютер, отладку можно вести на своем компьютере. Напомним, что имя узла содержится в переменной HOSTNAME.

boiler.cpp:

```
#include <vingraph.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <string>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <cstring>

class BoilerLocal {
public:
   BoilerLocal(int id) : id(id), fuel(100) {
      pthread_mutex_init(&mtx, NULL);
      rect = Rect(200 + id * 150, 200, 100, 80, 1, RGB(200, 150, 0));
```

```
    statusRect = Rect(202 + id * 150, 202, 96, 76, 0, RGB(255, 255, 100));

    char title[20];
    sprintf(title, "BOILER %d", id + 1);
    text = Text(225 + id * 150, 230, title, RGB(0, 0, 0));

    Show(rect);
    Show(statusRect);
    Show(text);
}

std::string getStatus() {
    pthread_mutex_lock(&mtx);
    std::string status;
    if (fuel == 0) status = "EMPTY";
    else if (fuel < 30) status = "LOW";
    else status = "OK";
    pthread_mutex_unlock(&mtx);
    return status;
}

void consume() {
    pthread_mutex_lock(&mtx);
    if (fuel > 0) fuel--;
    updateDisplay();
    pthread_mutex_unlock(&mtx);
}
```

```
    void refuel(int fuelType) {
        pthread_mutex_lock(&mtx);
        fuel = 100;
        updateDisplay();
        printf("[Boiler %d] Refueled with type %d\n", id + 1, fuelType);
        pthread_mutex_unlock(&mtx);
    }

    int getFuel() const { return fuel; }
    int getId() const { return id; }

private:
    void updateDisplay() {
        int fuelLevel = (fuel * 70) / 100;
        int fuelColor;

        if (fuel < 30) fuelColor = RGB(255, 50, 50);
        else if (fuel < 60) fuelColor = RGB(255, 200, 50);
        else fuelColor = RGB(50, 200, 50);

        Fill(statusRect, fuelColor);

        char fuelText[20];
        sprintf(fuelText, "%d%%", fuel);
        int fuelTextObj = Text(225 + id * 150, 250, fuelText, RGB(0, 0, 0));
        Show(fuelTextObj);
        usleep(50000);
        Delete(fuelTextObj);
    }

    int id;
    int fuel;
    int rect;
    int statusRect;
    int text;
    pthread_mutex_t mtx;
};

void* serverThread(void* arg) {
    std::vector<BoilerLocal*>* boilers =
        static_cast<std::vector<BoilerLocal*>*>(arg);

    int serverSock = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSock < 0) {
        perror("socket");
        return NULL;
    }

    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(8080);
    addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(serverSock, (sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("bind");
        close(serverSock);
        return NULL;
    }

    listen(serverSock, 5);

    while (1) {
        int clientSock = accept(serverSock, NULL, NULL);
        if (clientSock < 0) continue;

        char buf[1024];
        int len = recv(clientSock, buf, sizeof(buf) - 1, 0);
        if (len > 0) {
            buf[len] = '\0';
            int id, fuelType;

            if (sscanf(buf, "STATUS %d", &id) == 1) {
                if (id >= 0 && id < (int)boilers->size()) {
                    std::string status = (*boilers)[id]->getStatus();
                    send(clientSock, status.c_str(),
                         status.size(), 0);
                }
            } else if (sscanf(buf, "REFUEL %d %d", &id, &fuelType) == 2) {
                if (id >= 0 && id < (int)boilers->size()) {
                    (*boilers)[id]->refuel(fuelType);
                    const char* ok = "OK";
                    send(clientSock, ok, std::strlen(ok), 0);
                }
            }
        }
        close(clientSock);
    }

    close(serverSock);
    return NULL;
}

void* consumeThread(void* arg) {
    std::vector<BoilerLocal*>* boilers =
        static_cast<std::vector<BoilerLocal*>*>(arg);

    while (1) {
        size_t i;
        for (i = 0; i < boilers->size(); ++i) {
            (*boilers)[i]->consume();
        }
        usleep(500000);
    }
    return NULL;
}

int main() {
    ConnectGraph();

    int titleBg = Rect(250, 20, 200, 40, 1, RGB(100, 100, 200));
    int serverText = Text(280, 40, "BOILER SERVER", RGB(255, 255, 255));
    Show(titleBg);
Show(serverText);

BoilerLocal b1(0);
BoilerLocal b2(1);
BoilerLocal b3(2);
BoilerLocal b4(3);

std::vector<BoilerLocal*> boilers;
boilers.push_back(&b1);
boilers.push_back(&b2);
boilers.push_back(&b3);
boilers.push_back(&b4);

int infoText = Text(50, 350, "Press ESC to exit", RGB(150, 150, 150));
Show(infoText);

pthread_t srvThread, consThread;
pthread_create(&srvThread, NULL, serverThread, &boilers);
pthread_create(&consThread, NULL, consumeThread, &boilers);

int key = 0;
while (1) {
   key = InputChar();
   if (key == 27) break;
   usleep(10000);
}

pthread_cancel(srvThread);
```

```
    pthread_cancel(consThread);


    pthread_join(srvThread, NULL);
    pthread_join(consThread, NULL);


    CloseGraph();
    return 0;
}
```

Storage.cpp:

```
#include <vingraph.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <cstring>


class BoilerRemote {
public:
    BoilerRemote(int id, const char* host, int port) : id(id), host(host), port(port) {}


    std::string getStatus() {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0) return "ERROR";
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    inet_pton(AF_INET, host, &addr.sin_addr);

    if (connect(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
        close(sock);
        return "ERROR";
    }

    char buf[1024];
    sprintf(buf, "STATUS %d", id);
    send(sock, buf, std::strlen(buf), 0);

    int len = recv(sock, buf, 1024, 0);
    buf[len] = '\0';
    close(sock);

    return std::string(buf);
}

void refuel(int fuelType) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) return;

    sockaddr_in addr;
    addr.sin_family = AF_INET;
```

```
        addr.sin_port = htons(port);
        inet_pton(AF_INET, host, &addr.sin_addr);

        if (connect(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
            close(sock);
            return;
        }

        char buf[1024];
        sprintf(buf, "REFUEL %d %d", id, fuelType);
        send(sock, buf, std::strlen(buf), 0);
        close(sock);
    }

    bool isLowFuelOrEmpty() {
        std::string status = getStatus();
        return (status.compare("LOW") == 0) || (status.compare("EMPTY") == 0);
    }

    bool isEmpty() {
        return getStatus().compare("EMPTY") == 0;
    }

    int getX() const { return 200 + id * 150; }
    int getY() const { return 200; }

private:
    int id;
    const char* host;
    int port;
};

class Storage {
public:
    Storage(int x, int y) : x(x), y(y) {
        pthread_mutex_init(&mtx, NULL);
        rect = Rect(x, y, 100, 80, 1, RGB(0, 100, 200));
        fill = Rect(x+2, y+2, 96, 76, 0, RGB(0, 200, 100));
        text = Text(x + 15, y + 35, "FUEL", RGB(255, 255, 255));
        Show(rect);
        Show(fill);
        Show(text);
    }

    int getFuelType() {
        pthread_mutex_lock(&mtx);
        int f = rand() % 10 + 1;
        printf("[Storage] issued fuel type %d\n", f);
        pthread_mutex_unlock(&mtx);
        return f;
    }

    void refreshDisplay() {
        pthread_mutex_lock(&mtx);
        Show(rect);
        Show(fill);
        Show(text);
        pthread_mutex_unlock(&mtx);
    }

    int getX() const { return x; }
    int getY() const { return y; }

private:
    int x, y;
    int rect, fill, text;
    pthread_mutex_t mtx;
};

class Vehicle {
public:
    Vehicle(Storage* s, std::vector<BoilerRemote*>& bs, int startX, int startY, int id)
        : storage(s), boilers(bs), vehicleId(id) {
        pthread_mutex_init(&moveMtx, NULL);
        vehicleRect = Rect(startX, startY, 25, 15, 0, id == 0 ? RGB(50, 100, 200) :
RGB(200, 100, 50));
        Show(vehicleRect);
    }
    ~Vehicle() { pthread_mutex_destroy(&moveMtx); }

    static void* runHelper(void* arg) { return ((Vehicle*)arg)->runThread(); }

    void* runThread() {
        int i = vehicleId;
        while (1) {
            int fuel = storage->getFuelType();
            printf("[Vehicle %d] carrying fuel %d to boiler %d\n", vehicleId+1, fuel, i+1);

            if (boilers[i]->isLowFuelOrEmpty()) {
                moveTo(storage->getX(), storage->getY()-50, boilers[i]->getX(), boilers[i]-
>getY()-50);
                while (!boilers[i]->isEmpty()) { usleep(500000); }
                boilers[i]->refuel(fuel);
                moveTo(boilers[i]->getX(), boilers[i]->getY()-50, storage->getX(), storage-
>getY()-50);
            }
            i = (i+1) % boilers.size();
            usleep(2000000);
        }
    }

    void refreshDisplay() { Show(vehicleRect); }

private:
    void moveTo(int sx, int sy, int ex, int ey) {
        pthread_mutex_lock(&moveMtx);
        int steps = 30;
        int dx = (ex - sx) / steps;
        int dy = (ey - sy) / steps;
        MoveTo(sx, sy, vehicleRect);
        for (int s=0; s<steps; s++) {
            Move(vehicleRect, dx, dy);
            Fill(vehicleRect, vehicleId==0 ? RGB(50, 100, 200) : RGB(200, 100, 50));
            usleep(40000);
        }
        pthread_mutex_unlock(&moveMtx);
    }

    Storage* storage;
    std::vector<BoilerRemote*>& boilers;
    int vehicleId;
    int vehicleRect;
    pthread_mutex_t moveMtx;
};

int main() {
    ConnectGraph();

    int titleBg = Rect(250, 20, 200, 40, 1, RGB(100, 100, 200));
    int electricPlantText = Text(280, 40, "ELECTRIC PLANT", RGB(255, 255, 255));
    Show(titleBg);
    Show(electricPlantText);

    Storage storage(50, 200);
    BoilerRemote br1(0,"127.0.0.1",8080);
    BoilerRemote br2(1,"127.0.0.1",8080);
    BoilerRemote br3(2,"127.0.0.1",8080);
    BoilerRemote br4(3,"127.0.0.1",8080);

    std::vector<BoilerRemote*> boilers;
    boilers.push_back(&br1);
    boilers.push_back(&br2);
    boilers.push_back(&br3);
    boilers.push_back(&br4);

    Vehicle v1(&storage,boilers,20,20,0);
    Vehicle v2(&storage,boilers,60,20,1);

    int infoText = Text(50, 350, "Press ESC to exit", RGB(150, 150, 150));
    Show(infoText);

    pthread_t t1,t2;
    pthread_create(&t1,NULL,Vehicle::runHelper,&v1);
    pthread_create(&t2,NULL,Vehicle::runHelper,&v2);

    int key;
    while (1) {
        key = InputChar();
        if (key == 27) break;
        storage.refreshDisplay();
        v1.refreshDisplay();
        v2.refreshDisplay();
        usleep(10000);
    }

    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    CloseGraph();
    return 0; }
```