1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «**Синхронизация процессов**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/20001 Маронова К.Д.

<*подпись*>

1. Преподаватель
2. Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2023

**1. Цель работы**

Изучение примитивов синхронизации и методов работы с ними, решение классической задачи узкого моста и тестирование решения в рамках операционной системы Pintos.

**2. Постановка задачи**

Основная задача лабораторной работы состоит из нескольких пунктов:

1. Разработать систему семафоров для управления движением автомобилей по мосту без образования заторов и разрушения моста.

2. Реализовать код инициализации объектов синхронизации и глобальных переменных в функции narrow\_bridge\_init().

3. Разработать процедуру arrive\_bridge() для оценки возможности автомобиля пересечь мост безопасно. При наличии блокировки въезда, должен быть режим ожидания безопасного момента.

4. Реализовать процедуру exit\_bridge() для снятия ограничений на въезд.

**3. Описание решения**

1. Описание разработанного алгоритма синхронизации и обоснование его эффективности:

Вся работа производилась в файле narrow-bridge.c.

Для начала была создана структура bridge, которая включает в себя 4 семафора для регулирования движения каждого типа машин в зависимости от направления (struct semaphore left\_lane, struct semaphore right\_lane, struct semaphore emergency\_left, struct semaphore emergency\_right), а также общий счетчик машин, находящихся в данный момент на мосту (bridge\_cnt).

В функции narrow\_bridge\_init инициализируются все семафоры значением 0 (так как необходимо в начальный момент времени заблокировать все семафоры на въезд, чтобы проанализировать ситуацию), счетчик машин на мосту так же зануляется.

В функции arrive\_bridge учтены все условия, при которых та или иная машина не может проехать мост сразу же, как на него пришла. Если поступившая на вход машина подходит под какое-то из этих условий, то она «засыпает», а ее семафор блокируется, пока не появится причина, по которой ее можно будет разбудить. После захода машины в функцию обязательно делается инкремент переменной narrow\_bridge.bridge\_cnt.

Далее выполняется функция exit\_bridge, которая и реализует проезд и съезд машины с моста. Для машин скорой помощи с каждой стороны проверяется: есть ли еще спящие машины скорой помощи -> если есть, то «пробуждай» ее семафор. Если машины скорой помощи еще не закончились, то «буди» следом вторую машину скорой помощи.   
Однако, если для первой машины скорой помощи не оказалось машины скорой помощи в пару, то осуществляется проверка наличия с этой стороны моста обычной машины. Если такие есть, то машина скорой помощи «забирает» с собой обычную машину, чтобы исключить голодание процессов.   
Таким образом два sema\_up делаются практически одновременно, что и позволяет осуществить пересечение моста двумя автомобилями в один момент времени.

Гарантия того, что машины скорой помощи пересекут мост первыми, осуществляется посредством правильной расстановки условий. Первый if проверяет как раз-таки наличие машин скорой помощи, а все остальные else if выполняются только в том случае, если первый if выполнен не был.

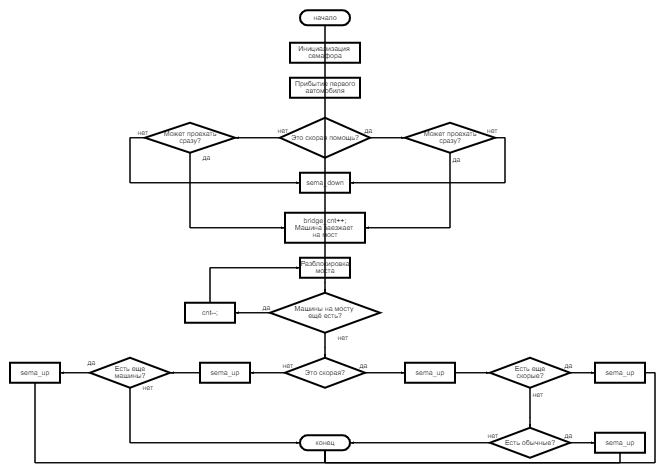
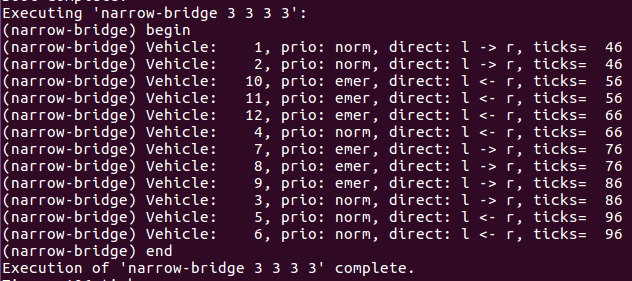
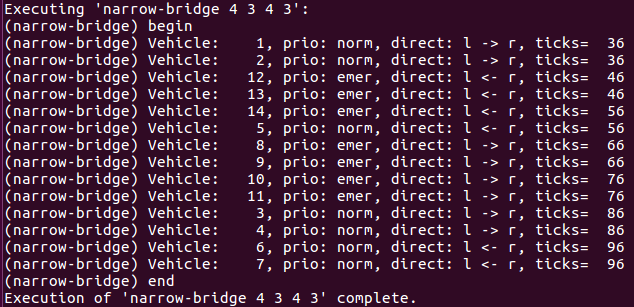


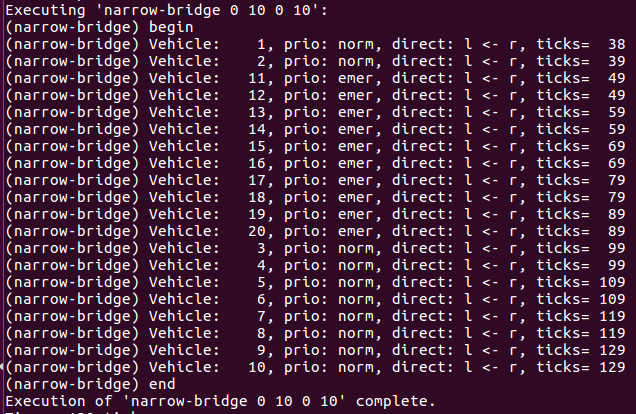
Рис. 1. «Диаграмма взаимодействия процессов».

**4. Тестирование и результаты работы программы**

Все необходимые тесты были успешно пройдены:







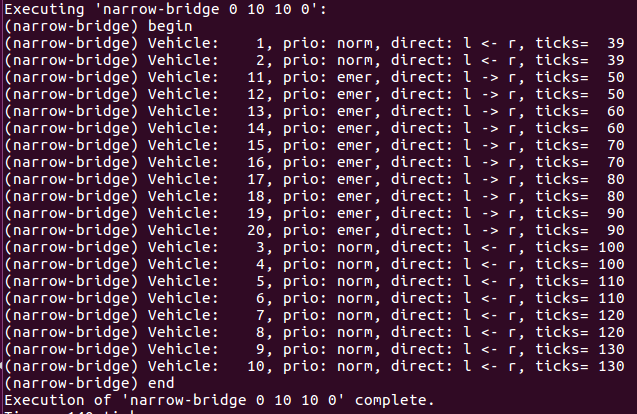


Рис. 2. «Результаты некоторых тестов».

**5. Выводы**

В ходе выполнения данной лабораторной работы была изучена классическая задача узкого моста, и реализован алгоритм, позволяющий получить ее решение в рамках Pintos. Для данной успешной реализации, позволяющей обеспечивать контроль над передвижением автомобилей по мосту, использовались различные средств, включая счетчики и такие примитивы синхронизации, как семафоры, созданные для машин каждого типа.

Основные проблемы, встретившиеся в ходе работы заключались в следующем: 1) Изначально было неясно, как реализовать проезд первых двух поступивших машинок перед скорыми. Это было решено корректировкой условий в arrive\_bridge.

Листинг программы:

/\* File for 'narrow\_bridge' task implementation.

SPbSTU, IBKS, 2017 \*/

#include <stdio.h>

#include "tests/threads/tests.h"

#include "threads/thread.h"

#include "threads/synch.h"

#include "devices/timer.h"

#include "narrow-bridge.h"

#include <debug.h>

struct bridge {

struct semaphore left\_lane;

struct semaphore right\_lane;

struct semaphore emergency\_left; //arrive

struct semaphore emergency\_right; //exit

int bridge\_cnt;//счетчик машин которые сейчас НА МОСТУ

};

int flag = 1;

struct bridge narrow\_bridge;

short cnt\_wait\_l\_em = 0; short cnt\_wait\_r\_em = 0; short cnt\_wait\_l\_car = 0; short cnt\_wait\_r\_car = 0; //счетчик машин, ожидающих заезд (спят)

int tmp\_dir = 3; //текущее направление (0/1 или 3- если не задано)

void narrow\_bridge\_init(void) {

sema\_init(&narrow\_bridge.left\_lane, 0);

sema\_init(&narrow\_bridge.right\_lane, 0);

sema\_init(&narrow\_bridge.emergency\_left, 0);// Семафор для блокировки автомобилей скорой помощи на выезде

sema\_init(&narrow\_bridge.emergency\_right, 0);

narrow\_bridge.bridge\_cnt = 0;

}

void arrive\_bridge(enum car\_priority prio, enum car\_direction dir) {

/\* это идентично для кажлого типа машин

Считаем количество машин на мосту (если все условия благоволят/ считаем

количество ожидающих заезд машин \*/

if (dir == dir\_right) {

if (prio == car\_normal) {

if ((narrow\_bridge.bridge\_cnt >= 2 || cnt\_wait\_r\_em > 0)) {

cnt\_wait\_r\_car++;

sema\_down(&narrow\_bridge.right\_lane);

}

}

else {

if (narrow\_bridge.bridge\_cnt >= 2) {

cnt\_wait\_r\_em++;

sema\_down(&narrow\_bridge.emergency\_right);

}

}

}

else {

if (prio == car\_normal) {

if ((narrow\_bridge.bridge\_cnt >= 2 || cnt\_wait\_l\_em > 0)) {

cnt\_wait\_l\_car++;

sema\_down(&narrow\_bridge.left\_lane);

}

}

else {

if (narrow\_bridge.bridge\_cnt >= 2) {

cnt\_wait\_l\_em++;

sema\_down(&narrow\_bridge.emergency\_left);

}

}

}

narrow\_bridge.bridge\_cnt++;

}

void exit\_bridge(enum car\_priority prio, enum car\_direction dir) {

if (narrow\_bridge.bridge\_cnt > 0) {

narrow\_bridge.bridge\_cnt--;

}

if (narrow\_bridge.bridge\_cnt == 0) {

if (cnt\_wait\_r\_em > 0) {

sema\_up(&narrow\_bridge.emergency\_right);

cnt\_wait\_r\_em--;

if (cnt\_wait\_r\_em != 0) {

sema\_up(&narrow\_bridge.emergency\_right);

cnt\_wait\_r\_em--;

}

else if (cnt\_wait\_r\_car != 0) {

sema\_up(&narrow\_bridge.right\_lane);

cnt\_wait\_r\_car--;

}

}

else if (cnt\_wait\_l\_em > 0) {

sema\_up(&narrow\_bridge.emergency\_left);

cnt\_wait\_l\_em--;

if (cnt\_wait\_l\_em != 0) {

sema\_up(&narrow\_bridge.emergency\_left);

cnt\_wait\_l\_em--;

}

else if (cnt\_wait\_l\_car != 0) {

sema\_up(&narrow\_bridge.left\_lane);

cnt\_wait\_l\_car--;

}

}

else if (cnt\_wait\_l\_car > 0) {

sema\_up(&narrow\_bridge.left\_lane);

cnt\_wait\_l\_car--;

if (cnt\_wait\_l\_car != 0) {

sema\_up(&narrow\_bridge.left\_lane);

cnt\_wait\_l\_car--;

}

}

else if (cnt\_wait\_r\_car > 0) {

sema\_up(&narrow\_bridge.right\_lane);

cnt\_wait\_r\_car--;

if (cnt\_wait\_r\_car != 0) {

sema\_up(&narrow\_bridge.right\_lane);

cnt\_wait\_r\_car--;

}

}

}

}