1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Высшая школа кибербезопасности»**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «СИНХРОНИЗАЦИЯ ПРОЦЕССОВ»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/30003 Шевчук Н.Е.

<*подпись*>

Преподаватель Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2024

**Оглавление**

[1 Цель работы 3](#_Toc183812586)

[2 Ход работы 3](#_Toc183812587)

[2.1 Описание алгоритма 3](#_Toc183812588)

[2.1.1 Используемые механизмы синхронизации 3](#_Toc183812589)

[2.1.2 Функция arrive\_bridge() 4](#_Toc183812590)

[2.1.3 Функция exit\_bridge() 5](#_Toc183812591)

[2.2 Результаты тестирования 7](#_Toc183812592)

[2.3 Дополнительное задание “Голодные птица” 8](#_Toc183812593)

[2.3.1 Описание алгоритма 8](#_Toc183812594)

[2.3.2 Тестирование дополнительного задания 10](#_Toc183812595)

[3 Вывод 12](#_Toc183812596)

# Цель работы

Изучить примитивы синхронизации и методы работы с ними, решить классическую задачу узкого моста и протестировать решение в рамках операционной системы Pintos.

# Ход работы

## Описание алгоритма

### Используемые механизмы синхронизации

Для синхронизации потоков было создано 4 семафора, отвечающие за блокировку/разблокировку потоков-машин: скорых справа и слева, обычных машин справа и слева. Семафоры позволяют блокировать поток c помощью sema\_down() в функции arrive\_bridge(), если текущие условия не позволяют машине сразу въехать на мост. В функции exit\_bridge() при определённых условиях, позволяющих машине въехать на мост поток разблокируется с помощью sema\_up(). Также созданы глобальные переменные отвечающие за текущее направление движения на мосту, количество машин на мосту в текущий момент, количество скорых ожидающих справа и слева, количество обычных машин отвечающих справа и слева.

static struct semaphore normal\_left;

static struct semaphore normal\_right;

static struct semaphore emergency\_left;

static struct semaphore emergency\_right;

static int now\_on\_bridge;

static enum car\_direction dir\_cur;

static int wait\_normal\_left;

static int wait\_normal\_right;

static int wait\_emergency\_left;

static int wait\_emergency\_right;

void narrow\_bridge\_init(void) {

sema\_init(&normal\_left, 0);

sema\_init(&normal\_right, 0);

sema\_init(&emergency\_left, 0);

sema\_init(&emergency\_right, 0);

now\_on\_bridge = 0;

dir\_cur = -1;

wait\_normal\_left = 0;

wait\_normal\_right = 0;

wait\_emergency\_left = 0;

wait\_emergency\_right = 0;

}

### Функция arrive\_bridge()

При въезде на мост проверяются следующие условия:

1. Количество машин на мосту больше или равно двум
2. Текущее направление не совпадает с направлением машины

Если хотя бы одно из условий выполняется, то машина не может въехать на мост. В таком случае увеличивается счетчик соответствующего типа машин и опускается соответствующий семафор, блокируя поток. Если ни одно из условий не выполняется, то машина может въехать на мост. Тогда поток-машина устанавливает своё направление и увеличивает текущее количество машин на мосту.

Код функции:

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

if (now\_on\_bridge >= 2 || (dir\_cur != -1 && dir\_cur != dir)) {

if (prio == car\_emergency) {

if (dir == dir\_left) {

wait\_emergency\_left++;

sema\_down(&emergency\_left);

}

else {

wait\_emergency\_right++;

sema\_down(&emergency\_right);

}

}

else {

if (dir == dir\_left) {

wait\_normal\_left++;

sema\_down(&normal\_left);

} else {

wait\_normal\_right++;

sema\_down(&normal\_right);

}

}

}

else {

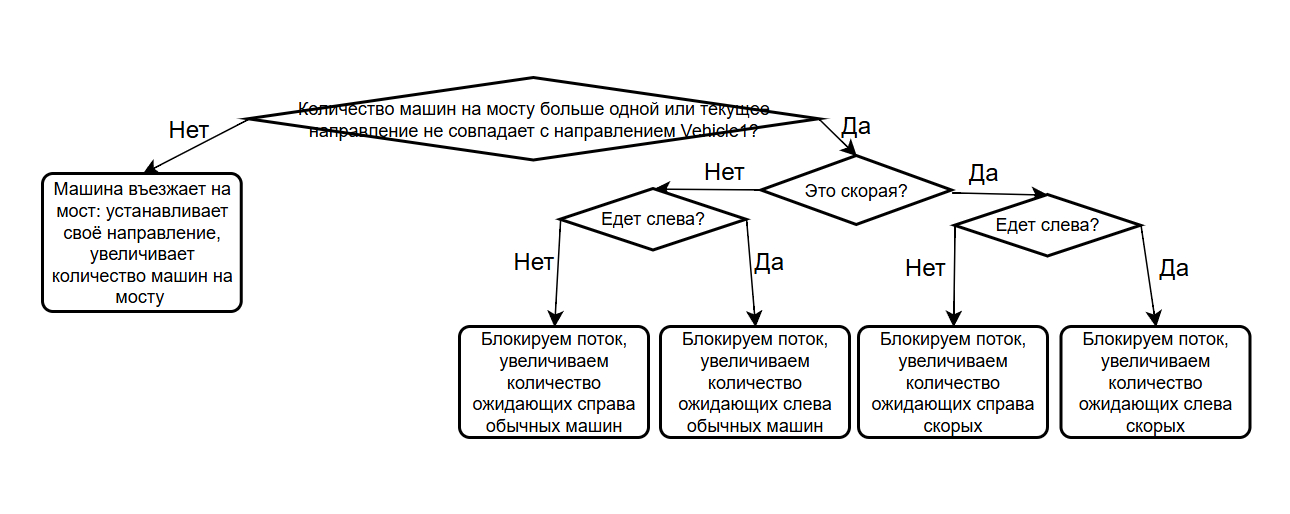
dir\_cur = dir;

now\_on\_bridge++;

}

}

Блок-схема функции arrive\_bridge():



### Функция exit\_bridge()

В этой функции разблокируются потоки, ранее заблокированные в arrive\_bridge. Соответственно, именно здесь реализованы:

* учет приоритета скорых (путём разблокировки их первыми);
* возможность проезда по мосту в одну сторону одновременно двум машинам (путём разблокировки двух потоков за раз);

Сначала проверяется больше ли нуля текущее количество машин на мосту. Если это так, то оно декрементируется. Дальнейшие разблокировки происходят только при условии, что количество машин на мосту равно нулю. Это позволяет потокам корректно ожидать своего пробуждения при вызове exit\_bridge(). Далее идёт перебор вариантов, ожидающих въезда на мост машин (см. блок-схему). При попадании в какой-то из вариантов поток разблокируется, декрементируется счетчик соответствующих ожидающих машин и инкрементируется счетчик количества машин на мосту. Затем проверяется возможность заезда на мост одновременно второй машины. Если возможность есть, то происходят аналогичные действия по разблокировке потока.

Код функции:

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

if (now\_on\_bridge > 0)

now\_on\_bridge--;

if (now\_on\_bridge == 0) {

if ((wait\_emergency\_left > 0 && dir == dir\_right) || (wait\_emergency\_right == 0 && wait\_emergency\_left > 0)) {

sema\_up(&emergency\_left);

wait\_emergency\_left--;

now\_on\_bridge++;

if (wait\_emergency\_left > 0) {

sema\_up(&emergency\_left);

wait\_emergency\_left--;

now\_on\_bridge++;

}

else if (wait\_normal\_left > 0) {

sema\_up(&normal\_left);

wait\_normal\_left--;

now\_on\_bridge++;

}

}

else if ((wait\_emergency\_right > 0 && dir == dir\_left) || (wait\_emergency\_left == 0 && wait\_emergency\_right > 0)) {

sema\_up(&emergency\_right);

wait\_emergency\_right--;

now\_on\_bridge++;

if (wait\_emergency\_right > 0) {

sema\_up(&emergency\_right);

wait\_emergency\_right--;

now\_on\_bridge++;

}

else if (wait\_normal\_right > 0) {

sema\_up(&normal\_right);

wait\_normal\_right--;

now\_on\_bridge++;

}

}

else if ((wait\_normal\_left > 0 && dir == dir\_right) || (wait\_normal\_left > 0 && wait\_normal\_right == 0)) {

sema\_up(&normal\_left);

wait\_normal\_left--;

now\_on\_bridge++;

if (wait\_normal\_left > 0) {

sema\_up(&normal\_left);

wait\_normal\_left--;

now\_on\_bridge++;

}

}

else if ((wait\_normal\_right > 0 && dir == dir\_left) || (wait\_normal\_left == 0 && wait\_normal\_right > 0)) {

sema\_up(&normal\_right);

wait\_normal\_right--;

now\_on\_bridge++;

if (wait\_normal\_right > 0) {

sema\_up(&normal\_right);

wait\_normal\_right--;

now\_on\_bridge++;

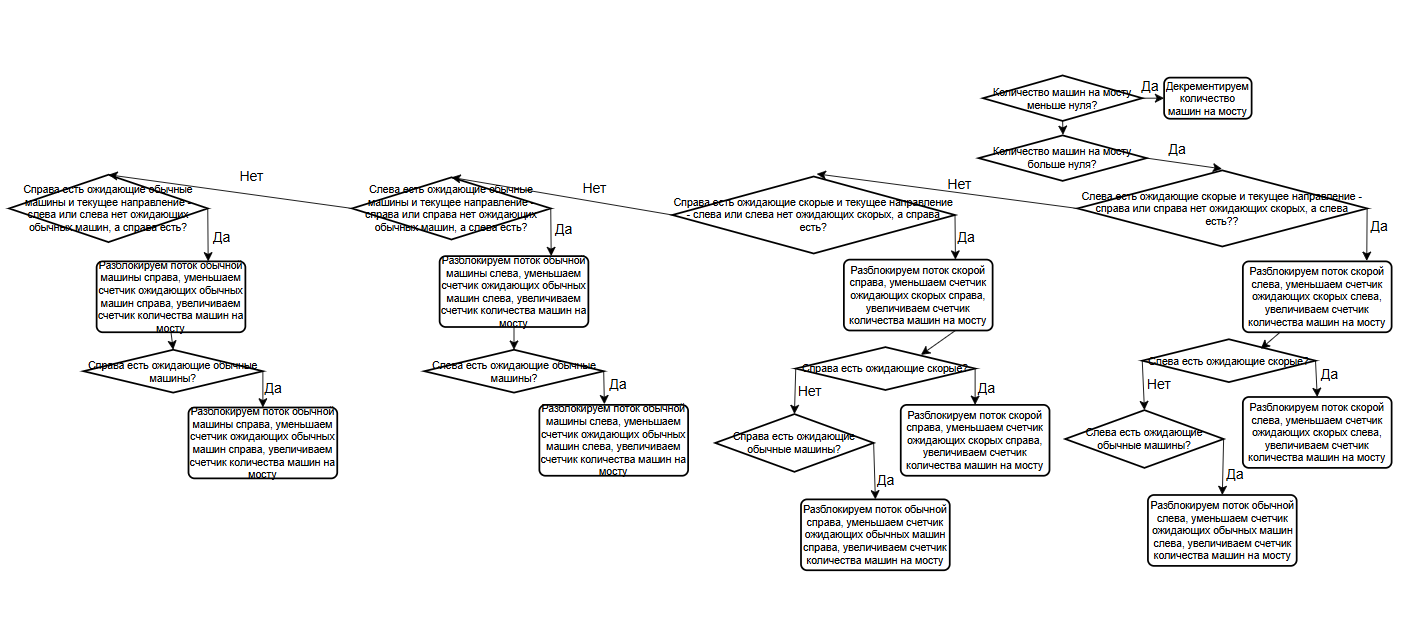
}

}

}

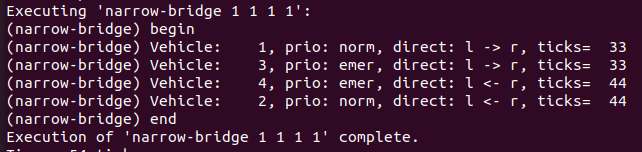
}

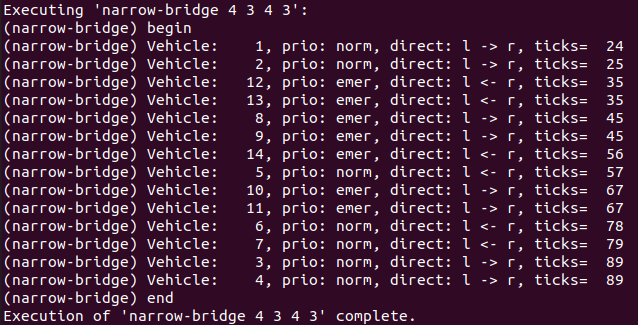
Блок-схема функции exit\_bridge():

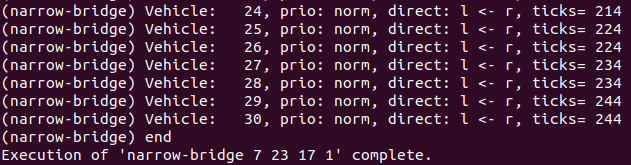
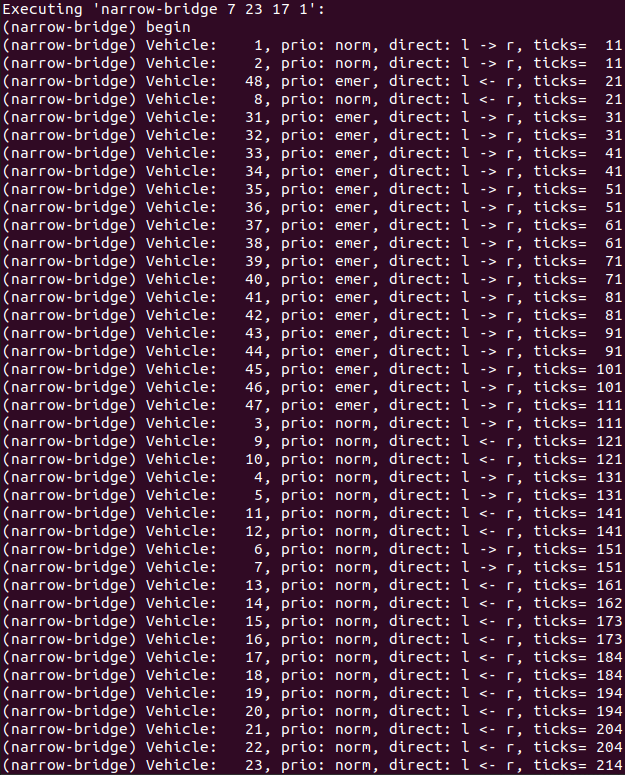


## Результаты тестирования

На рисунках ниже представлены результаты работы реализованного алгоритма при различных тестаx:







По результатам тестирования, алгоритм реализован корректно.

## Дополнительное задание “Голодные птица”

### Описание алгоритма

Для решения задачи были созданы семафоры

* bowl\_empty – отвечающий за пробуждение матери, если миска опустела
* bowl\_full – отвечающий за то, чтобы птенцы не начинали есть из миски до того, как мать ее наполнит.

Также добавлены глобальные переменные food\_left – количество оставшихся в миске порций, и bowl\_size – вместимость миски.

static struct semaphore bowl\_empty;

static struct semaphore bowl\_full;

static int food\_left;

static int bowl\_size;

static void init(unsigned int dish\_size) {

sema\_init(&bowl\_empty, 0);

sema\_init(&bowl\_full, 0);

food\_left = 0;

bowl\_size = dish\_size;

}

В функции bird() до бесконечного цикла миска заполняется в первый раз и поднимает семафор bowl\_full. Затем в бесконечном цикле поток-мать блокируется семафором bowl\_empty до тех пор, пока миска не станет пустой. Тогда он разблокируется, мать снова заполняет миску максимальным количеством порций и поднимает семафор bowl\_full, позволяя птенцам начать есть.

static void bird(void\* arg UNUSED) {

food\_left = bowl\_size;

msg("Mother bird: filling the bowl first time with %d portions", bowl\_size);

sema\_up(&bowl\_full);

while (true) {

msg("Mother bird: sleeping");

sema\_down(&bowl\_empty);

msg("Mother bird: filling the bowl with %d portions", bowl\_size);

food\_left = bowl\_size;

sema\_up(&bowl\_full);

}

}

В функции chick() в бесконечном цикле сначала поток-птенец блокируется семафором bowl\_full, ожидая когда миска заполнится. Затем птенец съедает порцию пищи и, если это была последняя порция, будит мать, поднимая семафор bowl\_empty. Если миска еще не пуста, то семафор bowl\_full поднимается, чтобы дать возможность следующему птенцу съесть свою порцию. После этого поток-птенец засыпает на некоторое время.

static void chick(void\* arg) {

int chick\_id = (int)arg;

while (true) {

sema\_down(&bowl\_full);

if (food\_left > 0) {

food\_left--;

msg("Chick %d: eating. Food left: %d", chick\_id, food\_left);

if (food\_left == 0) {

msg("Chick %d: waking up the mother", chick\_id);

sema\_up(&bowl\_empty);

}

}

if (food\_left > 0)

sema\_up(&bowl\_full);

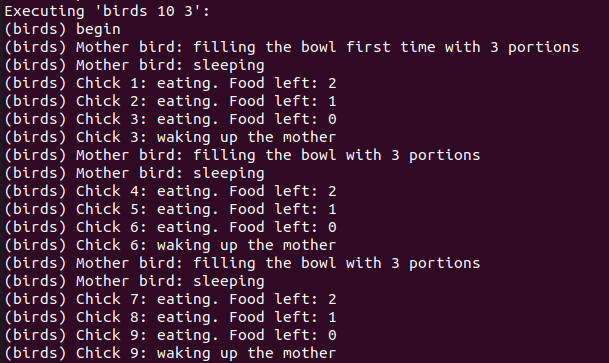
timer\_sleep(10);

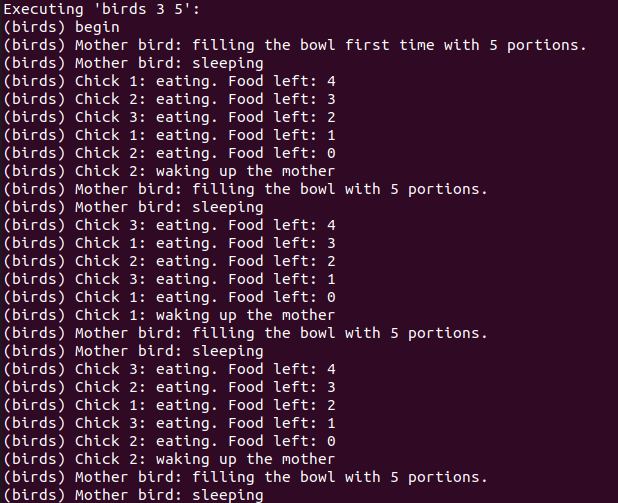
}

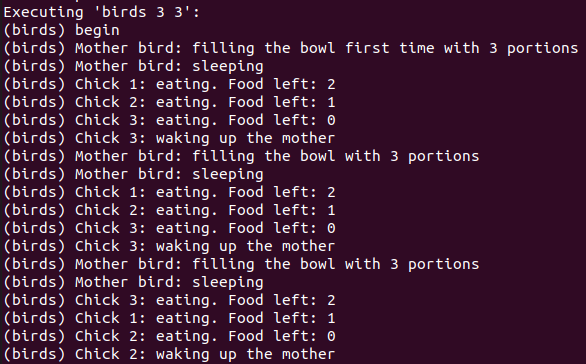
}

### Тестирование дополнительного задания

На рисунках ниже представлены результаты работы разработанного алгоритма при запуске различных тестов.







# Вывод

В ходе лабораторной работы были изучены примитивы синхронизации и методы работы с ними, решена классическая задача узкого моста и протестировано её решение в рамках операционной системы Pintos.