Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
—
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА №3

СИНХРОНИЗАЦИЯ ПРОЦЕССОВ

по дисциплине «Операционные системы»

Выполнил
студенты гр. 5131001/30002 Мишенев Н. С.

<подпись>

Руководитель
программист Огнёв Р. А.

<подпись>

Санкт-Петербург
2024г.

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

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

ХОД РАБОТЫ

1. **Описание разработанного алгоритма и обоснование его эффективности.**

В ходе выполнения лабораторной работы был разработан алгоритм, который позволяет решать классическую задачу узкого моста. Алгоритм действует следующим образом:

После появления первой машины, она запускается на мост, и, если вторая пришедшая машина едет в том же направлении она тоже запускается на мост, иначе ждет и после этого запускается на мост. Во время прохождения моста этими двумя машинами, к мосту подъезжают оставшиеся машины.

Как только на мосту оказывается 0 машин, функция ***exit_bridge(),*** вызывает созданную функцию ***notify(),*** которая запускает на мост необходимые машины, с учетом их приоритета (обычная машина или машина скорой помощи).

При этом, если последняя машина ехала, например, слева направо, то машины будут запускаться справа налево, таким образом решается проблема голодания процессов.

2. **Исходный код реализованных функций narrow_bridge_init(), arrive_bridge() и exit_bridge() с подробными комментариями.**

> narrow_bridge_init(...)

Для решения задачи была создана система семафоров и несколько констант, позволяющих отслеживать состояние моста и машин в любой момент времени.

```
11  /* Creating all the needed semaphores. These semaphores will
12      block exact groups of cars */
13  struct semaphore l_norm_sema;
14  struct semaphore r_norm_sema;
15  struct semaphore l_emer_sema;
16  struct semaphore r_emer_sema;
17
18  /* Creating all the needed constants. They will contain
19      specific car amounts and current bridge direction  */
20  int cars_on_bridge;
21  int l_norm_amt;
22  int r_norm_amt;
23  int l_emer_amt;
24  int r_emer_amt;
25  enum car_direction bridge_dir;
```

Рис. 1. Созданные константы и семафоры.

Далее все они были инициализированы своими значениями в функции ***narrow_bridge_init()*.**

```
69  // Called before test. Can initialize some synchronization objects.
70  void narrow_bridge_init(void)
71  {
72      sema_init(&l_norm_sema, 0);
73      sema_init(&r_norm_sema, 0);
74      sema_init(&l_emer_sema, 0);
75      sema_init(&r_emer_sema, 0);
76
77      cars_on_bridge = 0;
78      l_norm_amt = 0;
79      r_norm_amt = 0;
80      l_emer_amt = 0;
81      r_emer_amt = 0;
82  }
```

Рис. 2. Инициализация констант и семафоров.

> arrive_bridge(...)

При прибытии, первая и вторая машины рассматриваются отдельно, так как к этому моменту еще не приехали остальные, поэтому первая пришедшая машина устанавливает сторону движения на мосту, а вторая, если ее направление совпадает с первой, сразу же отправляется за ней.

Далее, производится подсчёт пришедших машин и их последующая блокировка.

```
84  void arrive_bridge(enum car_priority prio, enum car_direction dir)
85  {
86      if (cars_on_bridge == 0) {
87          cars_on_bridge++;
88          bridge_dir = dir;
89      } else if (cars_on_bridge == 1 && bridge_dir == dir) {
90          cars_on_bridge++;
91      } else {
92          if (prio == 0 && dir == 0) {
93              l_norm_amt++;
94              sema_down(&l_norm_sema);
95          } else if (prio == 0 && dir == 1) {
96              r_norm_amt++;
97              sema_down(&r_norm_sema);
98          } else if (prio == 1 && dir == 0) {
99              l_emer_amt++;
100             sema_down(&l_emer_sema);
101         } else if (prio == 1 && dir == 1) {
102             r_emer_amt++;
103             sema_down(&r_emer_sema);
104         }
105
106         bridge_dir = dir;
107     }
108 }
```

Рис. 3. Изменения в функции arrive_bridge(...)

> exit_bridge(...)

В эту функцию было добавлено пробуждение машин. Когда машина является последней на мосту, и после её съезда мост полностью свободен, она сигнализирует следующим машинам, с учетом приоритета, на стороне прибытия, о том, что им можно проезжать, если таковые ожидают своей очереди.

```
110 void exit_bridge(enum car_priority prio, enum car_direction dir)
111 {
112     cars_on_bridge--;
113     if (cars_on_bridge == 0) {
114         if (bridge_dir == 0)    {
115             if (r_emer_amt != 0) {
116                 notify("re");
117                 notify("rn");
118             } else if (l_emer_amt != 0) {
119                 notify("le");
120                 notify("ln");
121             } else if (r_norm_amt != 0) {
122                 notify("rn");
123             } else if (l_norm_amt != 0) {
124                 notify("ln");
125             }
126         } else {
127             if (l_emer_amt != 0) {
128                 notify("le");
129                 notify("ln");
130             } else if (r_emer_amt != 0) {
131                 notify("re");
132                 notify("rn");
133             } else if (l_norm_amt != 0) {
134                 notify("ln");
135             } else if (r_norm_amt != 0) {
136                 notify("rn");
```

Рис. 4. Изменения в функции exit_bridge(...)

Это производится с помощью функции ***notify(...)*** аргументом которой является строка, обозначающая сторону и приоритет машины.

```
27 /* Function will try to notify emergency cars on the specified side,
28     that can go through the bridge. Also, amount of cars will be
29     decremented and amount of cars on the bridge will be incremented.
30     Parameters:
31         [l - left, r - right][e - emergency, n - normal] */
32 void notify(char arg[])
33 {
34     char side = arg[0];
35     char car = arg[1];
36 
37     if (side == 'r') {
38         if (car == 'e') {
39             while (r_emer_amt != 0 && cars_on_bridge < 2) {
40                 r_emer_amt--;
41                 cars_on_bridge++;
42                 sema_up(&r_emer_sema);
43             }
44         } else if (car == 'n') {
45             while (r_norm_amt != 0 && cars_on_bridge < 2) {
46                 r_norm_amt--;
47                 cars_on_bridge++;
48 
49                 sema_up(&r_norm_sema);
50             }
51         }
```

Рис. 5. Функция notify(...)

Функция декрементирует количество соответствующих ожидающих машин, инкрементирует количество машин на мосту и поднимает семафор, который ответственен за этот тип машин. При этом, учитывается общее количество машин на мосту.

3. Диаграмма взаимодействия процессов на «узком мосте».

В соответствии с разработанным алгоритмом, была построена диаграмма взаимодействии “машин на мосту”, которая приведена ниже:

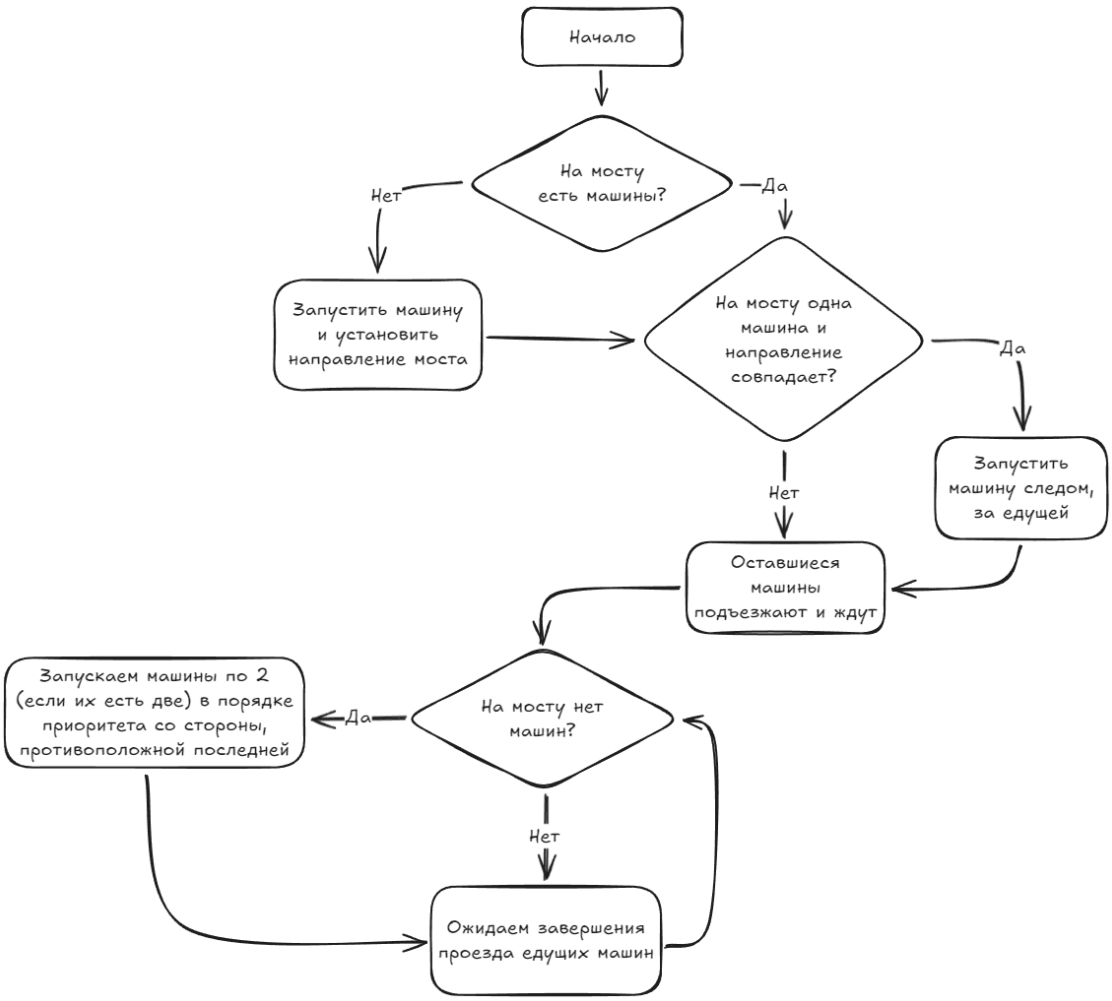

Рис. 6. диаграмма взаимодействии процессов

4. Полученный в результате выполнения программы вывод.

После компиляции ядра с внедренным решением был проведен ряд тестов, результаты некоторых будут упомянуты в отчете.

Были рассмотрены ситуации, когда машин очень мало, демонстрирующие правильность работы учёта приоритета, ситуации, когда машины распределены равномерно и когда неравномерно. Результаты выполнения тестов приведены ниже.

```
Executing 'narrow-bridge 0 0 0 0':
(narrow-bridge) begin
(narrow-bridge) end
Execution of 'narrow-bridge 0 0 0 0' complete.
```

Рис. 7. Результаты теста “narrow-bridge 0 0 0 0”

```
Executing 'narrow-bridge 0 0 0 1':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: emer, direct: l <- r, ticks=  37
(narrow-bridge) end
Execution of 'narrow-bridge 0 0 0 1' complete.
```

Рис. 8. Результаты теста “narrow-bridge 0 0 0 1”

```
Executing 'narrow-bridge 0 0 4 0':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: emer, direct: l -> r, ticks=  31
(narrow-bridge) Vehicle:    2, prio: emer, direct: l -> r, ticks=  31
(narrow-bridge) Vehicle:    3, prio: emer, direct: l -> r, ticks=  42
(narrow-bridge) Vehicle:    4, prio: emer, direct: l -> r, ticks=  42
(narrow-bridge) end
Execution of 'narrow-bridge 0 0 4 0' complete.
```

Рис. 9. Результаты теста “narrow-bridge 0 0 4 0”

```
Executing 'narrow-bridge 0 4 0 0':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: norm, direct: l <- r, ticks=  32
(narrow-bridge) Vehicle:    2, prio: norm, direct: l <- r, ticks=  32
(narrow-bridge) Vehicle:    3, prio: norm, direct: l <- r, ticks=  42
(narrow-bridge) Vehicle:    4, prio: norm, direct: l <- r, ticks=  42
(narrow-bridge) end
Execution of 'narrow-bridge 0 4 0 0' complete.
```

Рис. 10. Результаты теста “narrow-bridge 0 4 0 0”

```
Executing 'narrow-bridge 3 3 3 3':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: norm, direct: l -> r, ticks=  28
(narrow-bridge) Vehicle:    2, prio: norm, direct: l -> r, ticks=  28
(narrow-bridge) Vehicle:   10, prio: emer, direct: l <- r, ticks=  38
(narrow-bridge) Vehicle:   11, prio: emer, direct: l <- r, ticks=  38
(narrow-bridge) Vehicle:    7, prio: emer, direct: l -> r, ticks=  48
(narrow-bridge) Vehicle:    8, prio: emer, direct: l -> r, ticks=  48
(narrow-bridge) Vehicle:   12, prio: emer, direct: l <- r, ticks=  58
(narrow-bridge) Vehicle:    4, prio: norm, direct: l <- r, ticks=  58
(narrow-bridge) Vehicle:    9, prio: emer, direct: l -> r, ticks=  68
(narrow-bridge) Vehicle:    3, prio: norm, direct: l -> r, ticks=  68
(narrow-bridge) Vehicle:    5, prio: norm, direct: l <- r, ticks=  78
(narrow-bridge) Vehicle:    6, prio: norm, direct: l <- r, ticks=  78
(narrow-bridge) end
Execution of 'narrow-bridge 3 3 3 3' complete.
```

Рис. 11. Результаты теста “narrow-bridge 3 3 3 3”

```
Executing 'narrow-bridge 4 3 4 3':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: norm, direct: l -> r, ticks=  35
(narrow-bridge) Vehicle:    2, prio: norm, direct: l -> r, ticks=  35
(narrow-bridge) Vehicle:   12, prio: emer, direct: l <- r, ticks=  45
(narrow-bridge) Vehicle:   13, prio: emer, direct: l <- r, ticks=  45
(narrow-bridge) Vehicle:    8, prio: emer, direct: l -> r, ticks=  55
(narrow-bridge) Vehicle:    9, prio: emer, direct: l -> r, ticks=  55
(narrow-bridge) Vehicle:   14, prio: emer, direct: l <- r, ticks=  65
(narrow-bridge) Vehicle:    5, prio: norm, direct: l <- r, ticks=  65
(narrow-bridge) Vehicle:   10, prio: emer, direct: l -> r, ticks=  75
(narrow-bridge) Vehicle:   11, prio: emer, direct: l -> r, ticks=  75
(narrow-bridge) Vehicle:    6, prio: norm, direct: l <- r, ticks=  85
(narrow-bridge) Vehicle:    7, prio: norm, direct: l <- r, ticks=  85
(narrow-bridge) Vehicle:    3, prio: norm, direct: l -> r, ticks=  95
(narrow-bridge) Vehicle:    4, prio: norm, direct: l -> r, ticks=  95
(narrow-bridge) end
Execution of 'narrow-bridge 4 3 4 3' complete.
```

Рис. 12. Результаты теста “narrow-bridge 4 3 4 3”

```
Executing 'narrow-bridge 22 22 10 10':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: norm, direct: l -> r, ticks=  30
(narrow-bridge) Vehicle:    2, prio: norm, direct: l -> r, ticks=  30
(narrow-bridge) Vehicle:   55, prio: emer, direct: l <- r, ticks=  40
(narrow-bridge) Vehicle:   56, prio: emer, direct: l <- r, ticks=  40
(narrow-bridge) Vehicle:   45, prio: emer, direct: l -> r, ticks=  50
(narrow-bridge) Vehicle:   46, prio: emer, direct: l -> r, ticks=  50
(narrow-bridge) Vehicle:   57, prio: emer, direct: l <- r, ticks=  60
(narrow-bridge) Vehicle:   58, prio: emer, direct: l <- r, ticks=  60
(narrow-bridge) Vehicle:   47, prio: emer, direct: l -> r, ticks=  70
(narrow-bridge) Vehicle:   48, prio: emer, direct: l -> r, ticks=  70
(narrow-bridge) Vehicle:   59, prio: emer, direct: l <- r, ticks=  80
(narrow-bridge) Vehicle:   60, prio: emer, direct: l <- r, ticks=  80
(narrow-bridge) Vehicle:   49, prio: emer, direct: l -> r, ticks=  90
(narrow-bridge) Vehicle:   50, prio: emer, direct: l -> r, ticks=  90
(narrow-bridge) Vehicle:   61, prio: emer, direct: l <- r, ticks= 100
(narrow-bridge) Vehicle:   62, prio: emer, direct: l <- r, ticks= 100
(narrow-bridge) Vehicle:   51, prio: emer, direct: l -> r, ticks= 110
(narrow-bridge) Vehicle:   52, prio: emer, direct: l -> r, ticks= 110
(narrow-bridge) Vehicle:   63, prio: emer, direct: l <- r, ticks= 120
(narrow-bridge) Vehicle:   64, prio: emer, direct: l <- r, ticks= 120
(narrow-bridge) Vehicle:   53, prio: emer, direct: l -> r, ticks= 130
(narrow-bridge) Vehicle:   54, prio: emer, direct: l -> r, ticks= 130
(narrow-bridge) Vehicle:   23, prio: norm, direct: l <- r, ticks= 140
(narrow-bridge) Vehicle:   24, prio: norm, direct: l <- r, ticks= 140
(narrow-bridge) Vehicle:    3, prio: norm, direct: l -> r, ticks= 150
(narrow-bridge) Vehicle:    4, prio: norm, direct: l -> r, ticks= 150
(narrow-bridge) Vehicle:   25, prio: norm, direct: l <- r, ticks= 160
```

Рис. 13. Результаты теста “narrow-bridge 22 22 10 10”

```
Executing 'narrow-bridge 7 23 17 1':
(narrow-bridge) begin
(narrow-bridge) Vehicle:    1, prio: norm, direct: l -> r, ticks=  33
(narrow-bridge) Vehicle:    2, prio: norm, direct: l -> r, ticks=  33
(narrow-bridge) Vehicle:   48, prio: emer, direct: l <- r, ticks=  44
(narrow-bridge) Vehicle:    8, prio: norm, direct: l <- r, ticks=  44
(narrow-bridge) Vehicle:   31, prio: emer, direct: l -> r, ticks=  54
(narrow-bridge) Vehicle:   32, prio: emer, direct: l -> r, ticks=  54
(narrow-bridge) Vehicle:   33, prio: emer, direct: l -> r, ticks=  64
(narrow-bridge) Vehicle:   34, prio: emer, direct: l -> r, ticks=  64
(narrow-bridge) Vehicle:   35, prio: emer, direct: l -> r, ticks=  74
(narrow-bridge) Vehicle:   36, prio: emer, direct: l -> r, ticks=  74
(narrow-bridge) Vehicle:   37, prio: emer, direct: l -> r, ticks=  84
(narrow-bridge) Vehicle:   38, prio: emer, direct: l -> r, ticks=  84
(narrow-bridge) Vehicle:   39, prio: emer, direct: l -> r, ticks=  94
(narrow-bridge) Vehicle:   40, prio: emer, direct: l -> r, ticks=  94
(narrow-bridge) Vehicle:   41, prio: emer, direct: l -> r, ticks= 104
(narrow-bridge) Vehicle:   42, prio: emer, direct: l -> r, ticks= 104
(narrow-bridge) Vehicle:   43, prio: emer, direct: l -> r, ticks= 114
(narrow-bridge) Vehicle:   44, prio: emer, direct: l -> r, ticks= 114
(narrow-bridge) Vehicle:   45, prio: emer, direct: l -> r, ticks= 124
(narrow-bridge) Vehicle:   46, prio: emer, direct: l -> r, ticks= 124
(narrow-bridge) Vehicle:   47, prio: emer, direct: l -> r, ticks= 134
(narrow-bridge) Vehicle:    3, prio: norm, direct: l -> r, ticks= 134
(narrow-bridge) Vehicle:    9, prio: norm, direct: l <- r, ticks= 144
(narrow-bridge) Vehicle:   10, prio: norm, direct: l <- r, ticks= 144
(narrow-bridge) Vehicle:    4, prio: norm, direct: l -> r, ticks= 154
(narrow-bridge) Vehicle:    5, prio: norm, direct: l -> r, ticks= 154
```

Рис. 14. Результаты теста “narrow-bridge 7 23 17 1”

ВЫВОД

В ходе работы были изучены примитивы синхронизации и методы работы с ними, было разработано решение классической задачи узкого моста и проведено тестирование решения в рамках операционной системы Pintos.