

Лекция 7. Алгоритмы блокировок

Операционные системы

24 марта 2016 г.

Пример: сравнение данных POSIX

Пример

```
typedef
  struct
  {
    pthread_mutex_t
      m_hMutex;
    // ...
  }
  Data;
```

Пример (окончание)

```
int compare(Data *pData1, Data *pData2)
{
  int nResult = -1;
  pthread_mutex_lock(&pData1->m_hMutex);
  // ...
  pthread_mutex_lock(&pData2->m_hMutex);
  // nResult = ...;
  pthread_mutex_unlock(&pData2->m_hMutex);
  pthread_mutex_unlock(&pData1->m_hMutex);
  return nResult;
}
```

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Определение

Определение (сеть Петри)

$$\{S, D, F, M_0, W\}$$

- $S \neq \emptyset$ — множество **позиций** («мест»);
- $D \neq \emptyset$ — множество **переходов**, $S \cap D = \emptyset$;
- $F \subset (S \times D) \cup (D \times S)$ — множество **дуг** (отношение инцидентности);
- $M_0: S \rightarrow \mathbb{Z}_+$ — **начальная разметка**;
- $W: F \rightarrow \mathbb{N}$ — множество **весов дуг**.

Функции инцидентности, порождённые F

- $I: D \rightarrow 2^S$ — **входная функция** (прямая функция инцидентности);
- $O: D \rightarrow 2^S$ — **выходная функция** (обратная функция инцидентности).

Пример

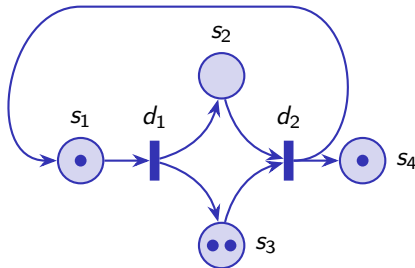


Рис. 1: сеть Петри

Пример

$$S = \{s_1, s_2, s_3, s_4\}$$

$$D = \{d_1, d_2\}$$

$$I(d_1) = \{s_1\}$$

$$O(d_1) = \{s_2, s_3\}$$

$$I(d_2) = \{s_2, s_3\}$$

$$O(d_2) = \{s_1, s_4\}$$

$$M_0 = (1, 0, 2, 1)$$

Срабатывание перехода

Функционирование сетей Петри

- Переход d является **активированным**, если

$$\forall s \in I(d) \quad M(s) \geq W(s, d).$$

- Активированный переход **может работать**.
- В результате срабатывания перехода меняется разметка:

$$\begin{aligned} \forall s \in I(d) \quad M'(s) &= M(s) - W(s, d) \\ \forall s \in O(d) \quad M'(s) &= M(s) + W(d, s) \end{aligned}$$

Срабатывание перехода

Функционирование сетей Петри

- Переход d является **активированным**, если

$$\forall s \in I(d) \quad M(s) \geq W(s, d).$$

- Активированный переход **может работать**.
- В результате срабатывания перехода меняется разметка:

$$\begin{aligned} \forall s \in I(d) \quad M'(s) &= M(s) - W(s, d) \\ \forall s \in O(d) \quad M'(s) &= M(s) + W(d, s) \end{aligned}$$

Срабатывание перехода

Функционирование сетей Петри

- Переход d является **активированным**, если

$$\forall s \in I(d) \quad M(s) \geq W(s, d).$$

- Активированный переход **может работать**.
- В результате срабатывания перехода меняется разметка:

$$\begin{aligned} \forall s \in I(d) \quad M'(s) &= M(s) - W(s, d) \\ \forall s \in O(d) \quad M'(s) &= M(s) + W(d, s) \end{aligned}$$

Срабатывание перехода (окончание)

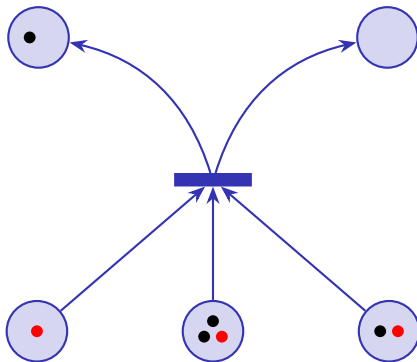


Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

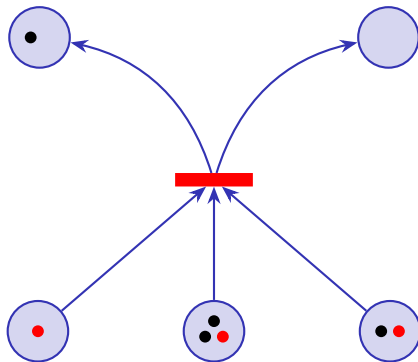


Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

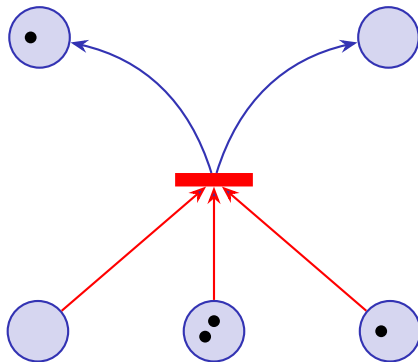


Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

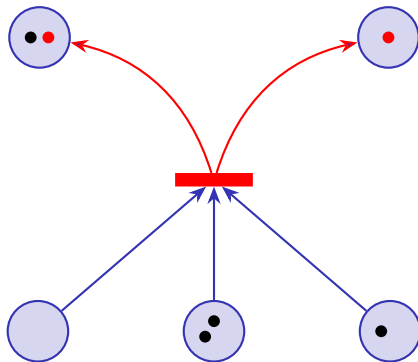


Рис. 2: последовательность срабатывания перехода

Особенности функционирования

Функционирование сетей Петри

- Возможные моменты срабатываний переходов — **дискретны**: ($\in \mathbb{Z}_+$).
- Переход срабатывает за одно **непрерываемое событие** («нулевая длительность», «транзакция»).
- Несколько разрешённых переходов могут находиться **в конфликте** друг с другом — только **один** из них может сработать («неодновременность наступления», недетерминированность).
- Несколько разрешённых переходов, не находящихся в конфликте друг с другом, могут срабатывать **независимо** (асинхронность).
- **Не требуется** обязательное срабатывание разрешённого перехода (момент срабатывания $\in \mathbb{Z}_+ \cup \{+\infty\}$, недетерминированность).

Примеры сетей Петри



Рис. 3: срабатывание переходов в определённой последовательности

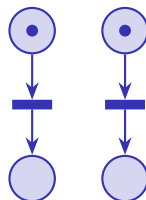


Рис. 4: независимое срабатывание переходов

Примеры сетей Петри (продолжение)

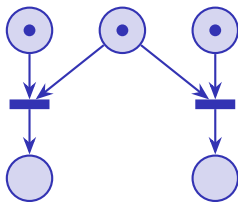


Рис. 5: блокировка перехода другим (конфликт переходов)

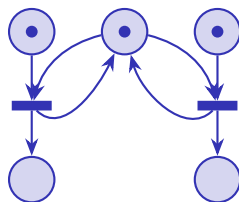


Рис. 6: неодновременное срабатывание переходов

Примеры сетей Петри (продолжение)

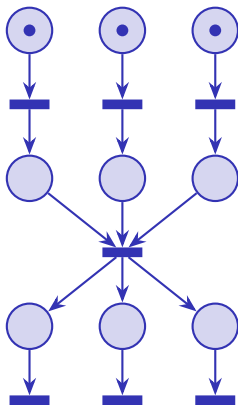


Рис. 7: барьер

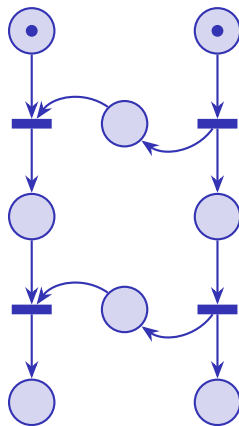


Рис. 8: конвейер

Последовательность срабатываний

Определение

Последовательность состояний и переходов $\sigma = (M_0, d_{i_1}, M_1, d_{i_2}, \dots, d_{i_n}, M_n)$, или просто: $\sigma = (d_{i_1}, d_{i_2}, \dots, d_{i_n})$, называется **последовательностью срабатываний**, если каждый переход d_{i_k} удовлетворяет условию разрешённости, и разметка M_{k+1} получается из M_k при помощи d_{i_k} .

Обозначение

$L(N, M_0)$ — множество **последовательностей срабатываний** (σ), достижимых в сети N при начальной разметке M_0 ;

Живучесть переходов и сети

Определения (живучесть перехода)

Переход называется:

L_0 -живым (мёртвым), если он **не принадлежит** ни одной **последовательности** $\sigma \in L(N, M_0)$ (не может сработать).

L_1 -живым, если он **принадлежит** некоторой **последовательности** $\sigma \in L(N, M_0)$ (может сработать).

...

Взаимные блокировки

Определение

Мёртвая блокировка: (*взаимная блокировка, deadlock*) — ситуация, в которой несколько процессов, заняв некоторые ресурсы, бесконечно ожидают освобождения других ресурсов, занятых ими же.

Живая блокировка: (*неустойчивая взаимная блокировка, livelock*) — ситуация, в которой несколько процессов, одновременно обнаружив взаимную блокировку, запускают механизм разблокирования, попадая в другую ситуацию блокировки.

Задача об обедающих философах

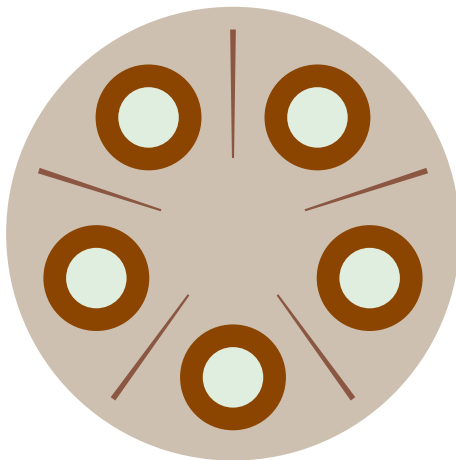


Рис. 9: обеденный стол в задаче о 5 философах

Требования ко взаимным исключениям

Требования

- В любой момент времени в одном критическом разделе может находиться не более одного процесса.
- Процесс, завершающий работу в некритическом разделе, не должен влиять на работу остальных.
- Не должны возникать взаимные блокировки и голодания.
- Когда в критическом разделе нет ни одного процесса, любой процесс, запросивший доступ к нему, должен немедленно его получить.
- Алгоритмы должны работать для любого количества процессов и их относительной скорости работы.
- Любой процесс должен оставаться в критическом разделе только в течение ограниченного времени.

Алгоритм критической секции 1

Код процесса 0

```
// ...  
while (nTurn != 0)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
nTurn = 1;  
// ...
```

Код процесса 1

```
// ...  
while (nTurn != 1)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
nTurn = 0;  
// ...
```

Алгоритм критической секции 1

Код процесса 0

```
// ...  
while (nTurn != 0)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
nTurn = 1;  
// ...
```

Код процесса 1

```
// ...  
while (nTurn != 1)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
nTurn = 0;  
// ...
```

Проблемы

- строгое чередование;
- блокировка при сбое другого процесса.

Алгоритм критической секции 2

Глобальные переменные

```
bool abFlags[2] = { false, false };
```

Код процесса 0

```
while (abFlags[1]) // (1)
    /* пережидание */ ;
//
abFlags[0] = true; // (2)
//
// критический раздел
//
abFlags[0] = false;
```

Код процесса 1

```
while (abFlags[0]) // (1)
    /* пережидание */ ;
//
abFlags[1] = true; // (2)
//
// критический раздел
//
abFlags[1] = false;
```

Алгоритм критической секции 2

Глобальные переменные

```
bool abFlags[2] = { false, false };
```

Код процесса 0

```
while (abFlags[1]) // (1)
    /* пережидание */ ;
//
abFlags[0] = true; // (2)
//
// критический раздел
//
abFlags[0] = false;
```

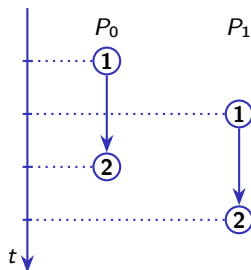


Рис. 10: порядок исполнения

Алгоритм критической секции 3

Код процесса 0

```
// ...  
abFlags[0] = true;    // (1)  
//  
while (abFlags[1])    // (2)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
abFlags[0] = false;  
// ...
```

Код процесса 1

```
// ...  
abFlags[1] = true;    // (1)  
//  
while (abFlags[0])    // (2)  
    /* пережидание */ ;  
//  
// критический раздел  
//  
abFlags[1] = false;  
// ...
```

Алгоритм критической секции 4

Код процесса 0

```
abFlags[0] = true;           // (1)
//
while (abFlags[1])           // (2)
{
    abFlags[0] = false;      // (3)
    // задержка
    abFlags[0] = true;       // (4)
}
//
// критический раздел
//
abFlags[0] = false;
```

Код процесса 1

```
abFlags[1] = true;           // (1)
//
while (abFlags[0])           // (2)
{
    abFlags[1] = false;      // (3)
    // задержка
    abFlags[1] = true;       // (4)
}
//
// критический раздел
//
abFlags[1] = false;
```

Алгоритм Деккера (Dijkstra, 1965)

Код процесса 0

```
abFlags[0] = true;           // (1)
while (abFlags[1])          // (2)
  if (nTurn == 1)           // (3)
  {
    abFlags[0] = false;     // (4)
    while (nTurn == 1)      // (5)
      /* пережидание */ ;
    abFlags[0] = true;      // (6)
  }
// критический раздел
nTurn = 1;                   // (7)
abFlags[0] = false;         // (8)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
while (abFlags[0])          // (2)
  if (nTurn == 0)           // (3)
  {
    abFlags[1] = false;     // (4)
    while (nTurn == 0)      // (5)
      /* пережидание */ ;
    abFlags[1] = true;      // (6)
  }
// критический раздел
nTurn = 0;                   // (7)
abFlags[1] = false;         // (8)
```


Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- `abFlags[0] == true` \Rightarrow

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- `abFlags[0] == true` \Rightarrow
 - P_1 вне критического раздела \Rightarrow

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе \Rightarrow

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован \Rightarrow

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел.

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел. Это невозможно \Leftarrow иначе $abFlags[1] == \mathbf{false}$.

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел. Это невозможно \Leftarrow иначе $abFlags[1] == \mathbf{false}$.
 - 2 P_1 ожидает входа в критический раздел.

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел. Это невозможно \Leftarrow иначе $abFlags[1] == \mathbf{false}$.
 - 2 P_1 ожидает входа в критический раздел. Это невозможно \Leftarrow иначе если $nTurn == 1 \Rightarrow P_1$ может в него войти.

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел. Это невозможно \Leftarrow иначе $abFlags[1] == \mathbf{false}$.
 - 2 P_1 ожидает входа в критический раздел. Это невозможно \Leftarrow иначе если $nTurn == 1 \Rightarrow P_1$ может в него войти.
 - 3 P_1 циклически монополизирует критический раздел.

Алгоритм Петерсона (Peterson, 1981)

Код процесса 0

```
abFlags[0] = true;           // (1)
nTurn = 1;                   // (2)
while (abFlags[1] &&        // (3)
       nTurn == 1)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[0] = false;         // (5)
```

Код процесса 1

```
abFlags[1] = true;           // (1)
nTurn = 0;                   // (2)
while (abFlags[0] &&        // (3)
       nTurn == 0)         // (4)
    /* пережидание */ ;
//
// критический раздел
//
abFlags[1] = false;         // (5)
```

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимного исключения)

- $abFlags[0] == \mathbf{true} \Rightarrow$
 - P_1 вне критического раздела $\Rightarrow P_1$ не сможет зайти в него.
 - P_1 в критическом разделе $\Rightarrow abFlags[1] == \mathbf{true} \Rightarrow P_0$ не сможет зайти.
- Взаимная блокировка исключена \Leftarrow противное: пусть P_0 заблокирован $\Rightarrow abFlags[1] == \mathbf{true}$ и $nTurn == 1$. P_0 может войти в критический раздел, если $abFlags[1] == \mathbf{false}$ или $nTurn == 0$. Случаи:
 - 1 P_1 не намерен входить в критический раздел. Это невозможно \Leftarrow иначе $abFlags[1] == \mathbf{false}$.
 - 2 P_1 ожидает входа в критический раздел. Это невозможно \Leftarrow иначе если $nTurn == 1 \Rightarrow P_1$ может в него войти.
 - 3 P_1 циклически монополизировать критический раздел. Это невозможно \Leftarrow перед \forall попыткой P_1 должен дать такую же возможность P_0 , устанавливая $nTurn = 0$ (не успевает $\Rightarrow abFlags[1] = \mathbf{false}$).

Алгоритм Лампорта (Lamport, 1974)

Блокировка

```
void lock(int i)
{
    abFlags[i] = true;
    anNumbs[i] = 1 + *std::max(anNumbs, anNumbs + NUM_THREADS);
    abFlags[i] = false;
    for (int j = 0; j < NUM_THREADS; ++ j)
    {
        while (abFlags[j])    // ожидание получения  $P_j$  номера
            /* пережидание */ ;
        while (anNumbs[j] != 0 &&
            std::make_pair(anNumbs[j], j) < std::make_pair(anNumbs[i], i))
            /* пережидание */ ;
    }
}
```

Алгоритм Лампорта (Lamport, 1974, окончание)

Разблокировка

```
void unlock(int i)
{
    anNumbs[i] = 0;
}
```


Проверка и установка

входные данные: i // по ссылке
начало

```
если  $i = 0$ , то
|    $i \leftarrow 1$ ;
|   вернуть „ложь“;
иначе
|   вернуть „истина“; //  $i \neq 0$ 
```

Рис. 11: алгоритм атомарной проверки и установки

Код процесса i

```
int bLock = 0; // глобальная
// ...
while (TestAndSet(&bLock))
    /* пережидание */ ;
//
// критический раздел
//
bLock = 0;
// ...
```

Проверка и установка

входные данные: i // по ссылке
начало

```
t ← i;  
i ← 1;  
вернуть (t ≠ 0);
```

Рис. 11: алгоритм атомарной проверки
и установки

Код процесса i

```
int bLock = 0;    // глобальная  
// ...  
while (TestAndSet(&bLock))  
    /* пережидание */ ;  
//  
// критический раздел  
//  
bLock = 0;  
// ...
```

Проверка и установка (окончание)

Пример (вход, X86)

```
enter_lock: tsl   reg, flag  
            cmp   reg, #0  
            jnz   enter_lock  
            ret
```

Пример (выход, X86)

```
leave_lock: mov   flag, #0  
            ret
```

Проверка и проверка и установка

Реализация 1

```
int bLock = 0;    // глобальная
// ...
do
{
    while (bLock)
        /* пережидание */ ;
}
while (TestAndSet(&bLock));
//
// критический раздел
//
bLock = 0;
```

Реализация 2

```
int bLock = 0;    // глобальная
// ...
while (bLock || TestAndSet(&bLock))
    /* пережидание */ ;
//
// критический раздел
//
bLock = 0;
```

Обмен

входные данные: r // регистр
 m ; // память

вспомогательные данные: t

начало

$$\left[\begin{array}{l} t \leftarrow m; \\ m \leftarrow r; \\ r \leftarrow t; \end{array} \right.$$

Рис. 12: алгоритм атомарного обмена

Соотношение

$$bLock + \sum_{i=1}^n bKeyI_i = n$$

Код процесса i

```
int bLock = 0;    // глобальная
// ...
int bKeyI = 1;   // локальная
while (bKeyI)
    Exchange(bKeyI, bLock);
//
// критический раздел
//
Exchange(bKeyI, bLock);
// ...
```

Обмен

входные данные: r // регистр
 m ; // память

вспомогательные данные: t

начало

```
[   $t \leftarrow m$ ;  
   $m \leftarrow r$ ;  
   $r \leftarrow t$ ;
```

Рис. 12: алгоритм атомарного обмена

Соотношение

$$bLock + \sum_{i=1}^n bKeyI_i = n$$

Код процесса i

```
int bLock = 0;    // глобальная  
// ...  
int bKeyI = 1;   // локальная  
while (bKeyI)  
    Exchange(bKeyI, bLock);  
//  
// критический раздел  
//  
Exchange(bKeyI, bLock);  
// ...
```

Связь между проверкой-установкой и обменом

входные данные: i // по ссылке
начало

```
[  $t \leftarrow i$ ;  
   $i \leftarrow 1$ ;  
  вернуть ( $t \neq 0$ );
```

Рис. 13: алгоритм атомарной проверки и установки

входные данные: i // по ссылке
начало

```
[  $t \leftarrow 1$ ;  
  обмен( $i, t$ ); // атомарная  
  вернуть ( $t \neq 0$ );
```

Рис. 14: алгоритм атомарной проверки и установки

Особенности аппаратной реализации критических секций

Преимущества

- Простота.
- Применимость к любому количеству процессов и процессоров с общей памятью.
- Поддержка нескольких критических разделов.

Недостатки

- Использование пережидания.
- Возможность голодания.
- Возможность взаимной блокировки (процессы с разными приоритетами).

Семафоры (сильные/слабые)

входные данные:

s // переменная-семафор

начало

$s.count \leftarrow s.count - 1;$

если $s.count < 0$, **то**

Поместить текущий процесс в $s.queue$;
Заблокировать текущий процесс;

Рис. 15: алгоритм попытки захвата семафора

входные данные:

s // переменная-семафор

начало

$s.count \leftarrow s.count + 1;$

если $s.queue$ не пуста, **то**

Удалить процесс P
из $s.queue$;
Поместить процесс P
в список активных;

Рис. 16: алгоритм освобождения семафора

Реализация семафоров в однопроцессорной среде

входные данные:

s // переменная-семафор

начало

- ┌ Запретить прерывания;
- ┌ Остальная реализация;
- ┌ Разрешить прерывания;

Рис. 17: алгоритм попытки захвата семафора

входные данные:

s // переменная-семафор

начало

- ┌ Запретить прерывания;
- ┌ Остальная реализация;
- ┌ Разрешить прерывания;

Рис. 18: алгоритм освобождения семафора

Реализация семафоров при помощи атомарной операции

входные данные:

s // переменная-семафор

начало

пока $\text{пу}(s.flag)$, выполнять
└ Пережидание;
Остальная реализация;
 $s.flag \leftarrow 0$;

Рис. 19: алгоритм попытки захвата семафора

входные данные:

s // переменная-семафор

начало

пока $\text{пу}(s.flag)$, выполнять
└ Пережидание;
Остальная реализация;
 $s.flag \leftarrow 0$;

Рис. 20: алгоритм освобождения семафора