Лекция 6. Алгоритмы блокировок

Операционные системы

11 ноября 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)

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

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

Функции инцидентности, порождённые F

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

Определение

Определение (сеть Петри)

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

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

Функции инцидентности, порождённые F

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

Определение

Определение (сеть Петри)

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

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

Функции инцидентности, порождённые F

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

Определение

Определение (сеть Петри)

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

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

Функции инцидентности, порождённые F

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

Определение

Определение (сеть Петри)

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

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

Функции инцидентности, порождённые F

$I \colon D \to 2^S$	—	входная функция (прямая функция инцидентности);
$O \colon D \to 2^S$	—	выходная функция (обратная функция инцидентности).

Определение

Определение (сеть Петри)

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

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

Функции инцидентности, порождённые F

$I \colon D \to 2^S$	—	входная функция (прямая функция инцидентности);
$O \colon D \to 2^S$	—	выходная функция (обратная функция инцидентности).

Пример

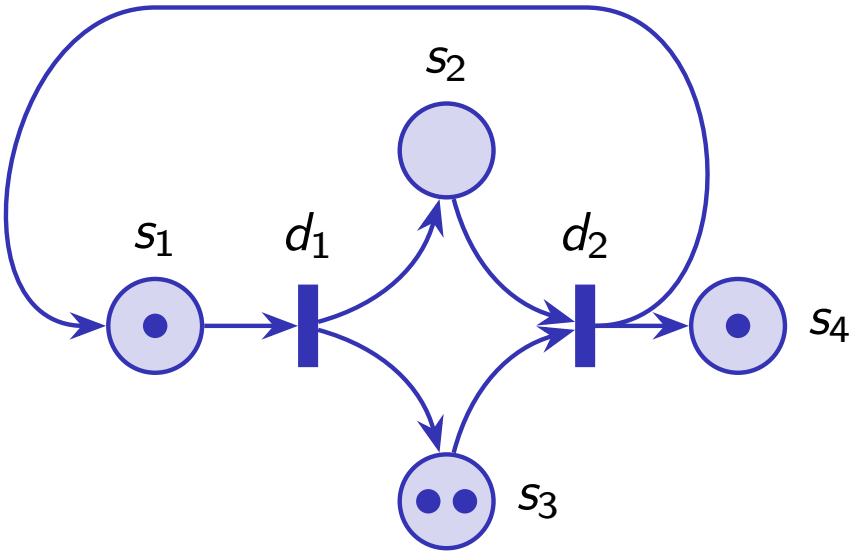

Рис. 1: сеть Петри

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

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

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

$$\forall s \in I(d) \quad M'(s) = M(s) - W(s,d)$$
$$\forall s \in O(d) \quad M'(s) = M(s) + W(d,s)$$

Срабатывание перехода

Функционирование сетей Петри

- Переход d является активированным, если

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

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

$$\forall s \in I(d) \quad M'(s) = M(s) - W(s,d)$$
$$\forall s \in O(d) \quad M'(s) = M(s) + W(d,s)$$

Срабатывание перехода

Функционирование сетей Петри

- Переход d является активированным, если

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

- Активированный переход может сработать.
- 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)

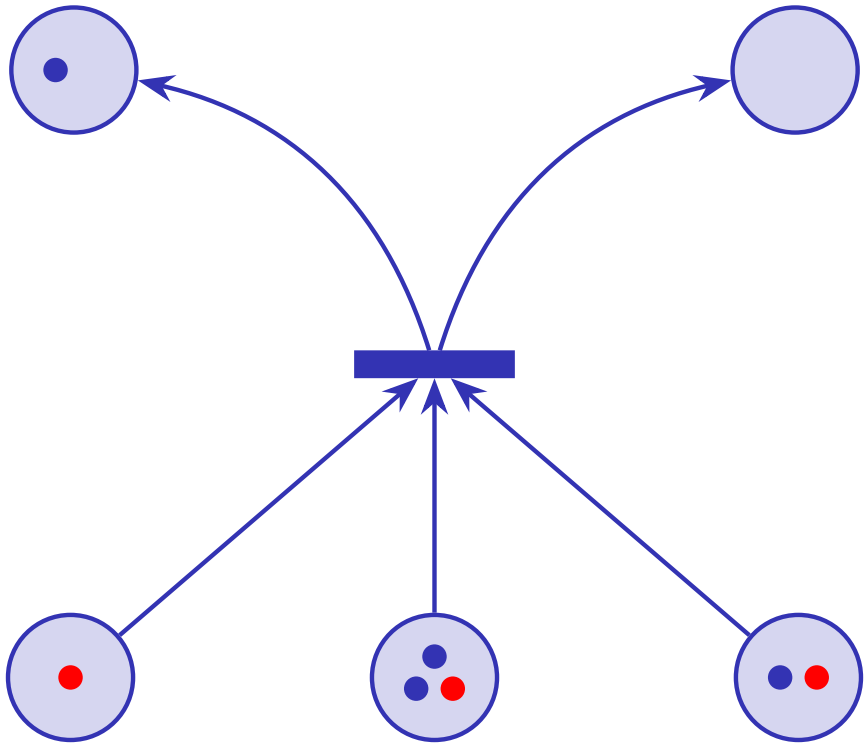

Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

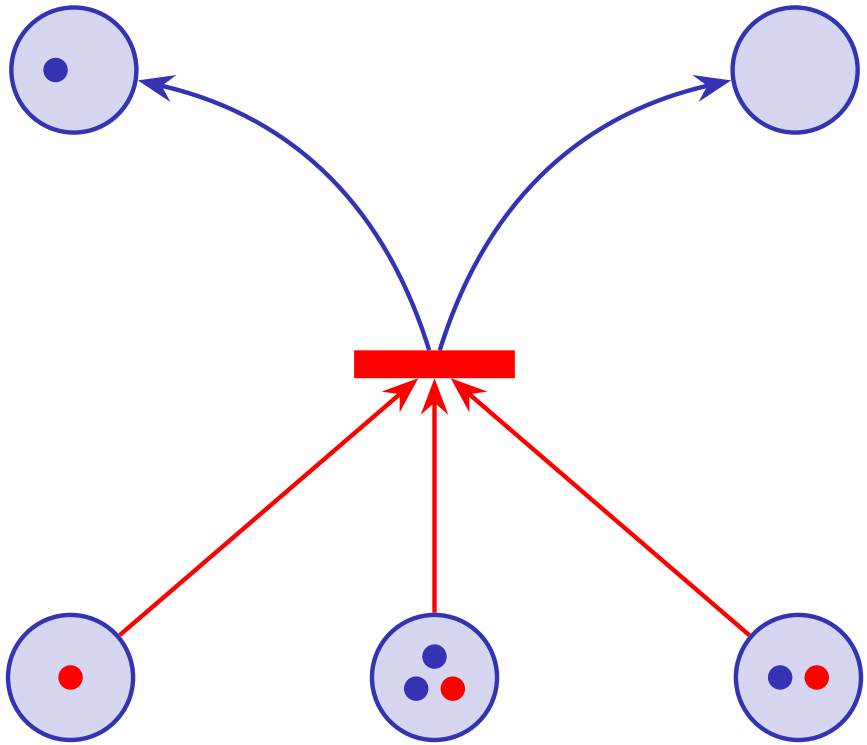

Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

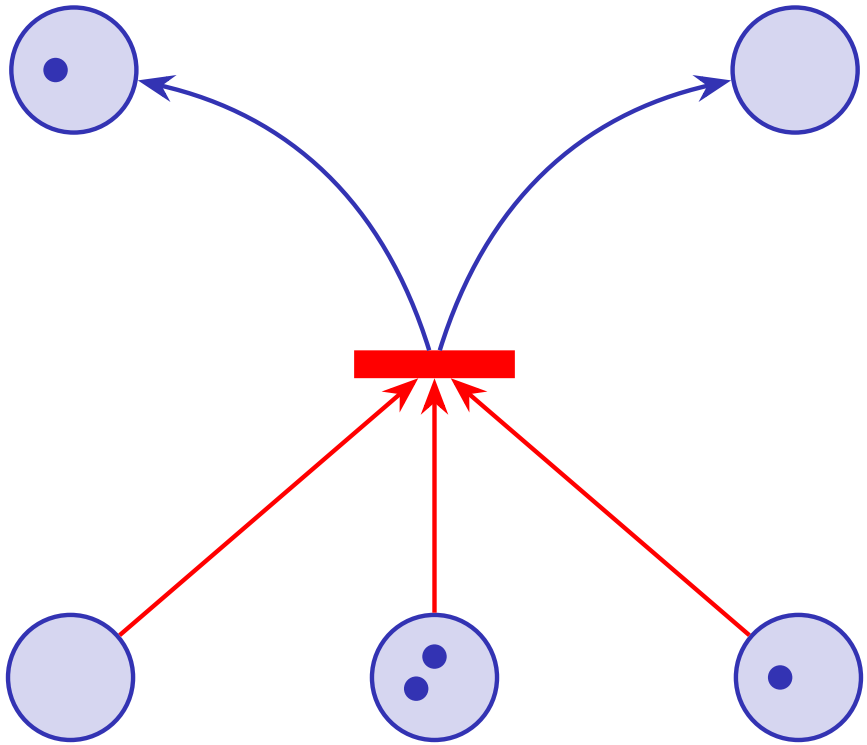

Рис. 2: последовательность срабатывания перехода

Срабатывание перехода (окончание)

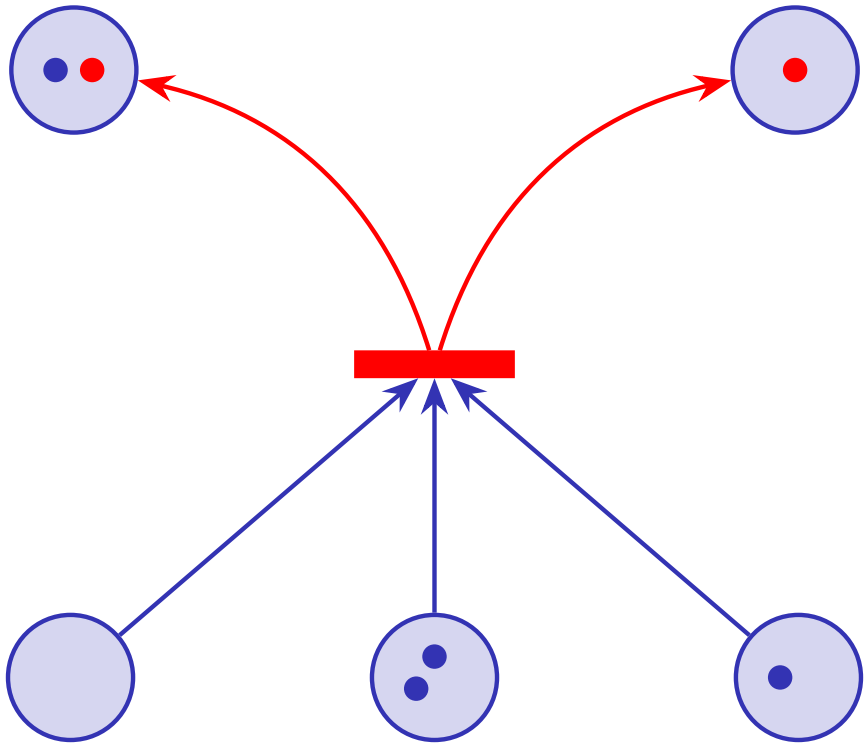

Рис. 2: последовательность срабатывания перехода

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

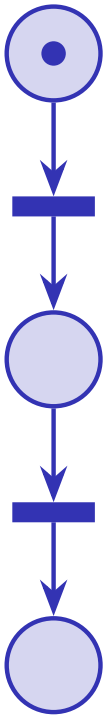

Рис. 3: срабатывание переходов в определённой последовательности

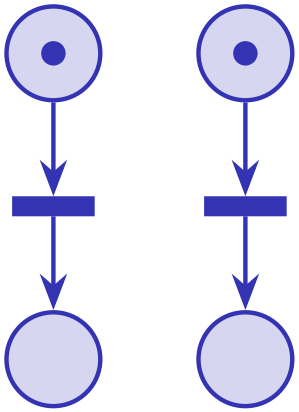

Рис. 4: независимое срабатывание переходов

Примеры сетей Петри (продолжение)

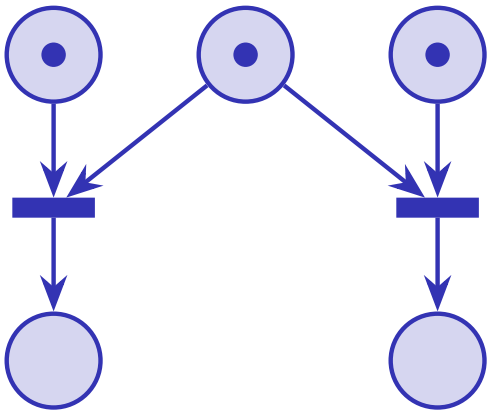

Рис. 5: блокировка перехода другим (конфликт переходов)

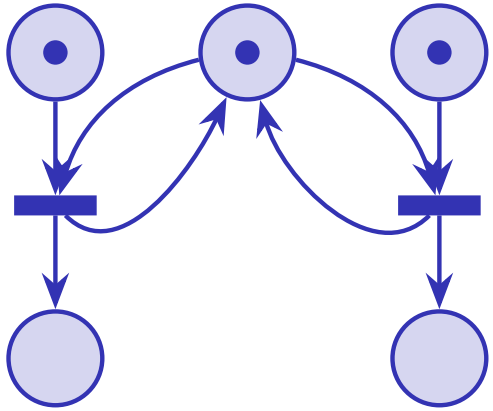

Рис. 6: неодновременное срабатывание переходов

Примеры сетей Петри (окончание)

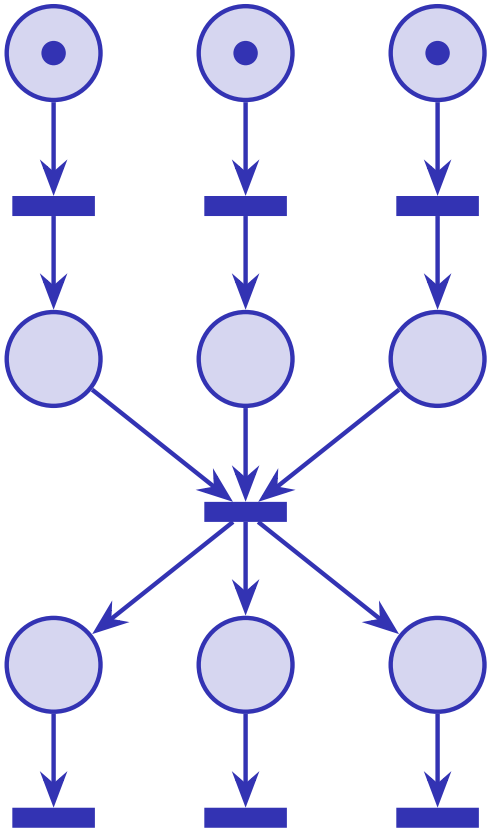

Рис. 7: барьер

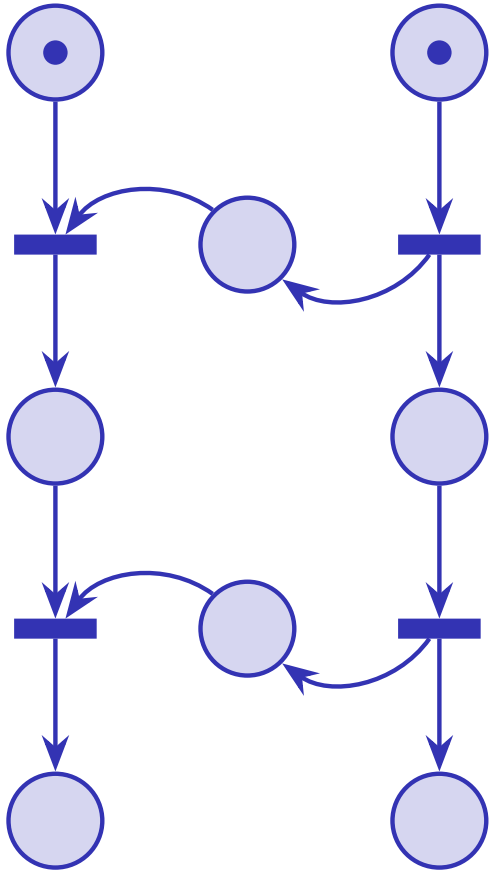

Рис. 8: конвейер

Последовательность срабатываний

Определение

Последовательность состояний и переходов $\sigma = (M_0, d_{i_1}, M_1, d_{i_2}, \ldots, d_{i_n}, M_n)$, или просто: $\sigma = (d_{i_1}, d_{i_2}, \ldots, d_{i_n})$, 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;

$R(N, M_0)$ — множество разметок (M), достижимых в сети N при начальной разметке M;

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).

...

L_4-живым (живым), если в $\forall M \in R(N, M_0)$, он является L_1-живым.

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

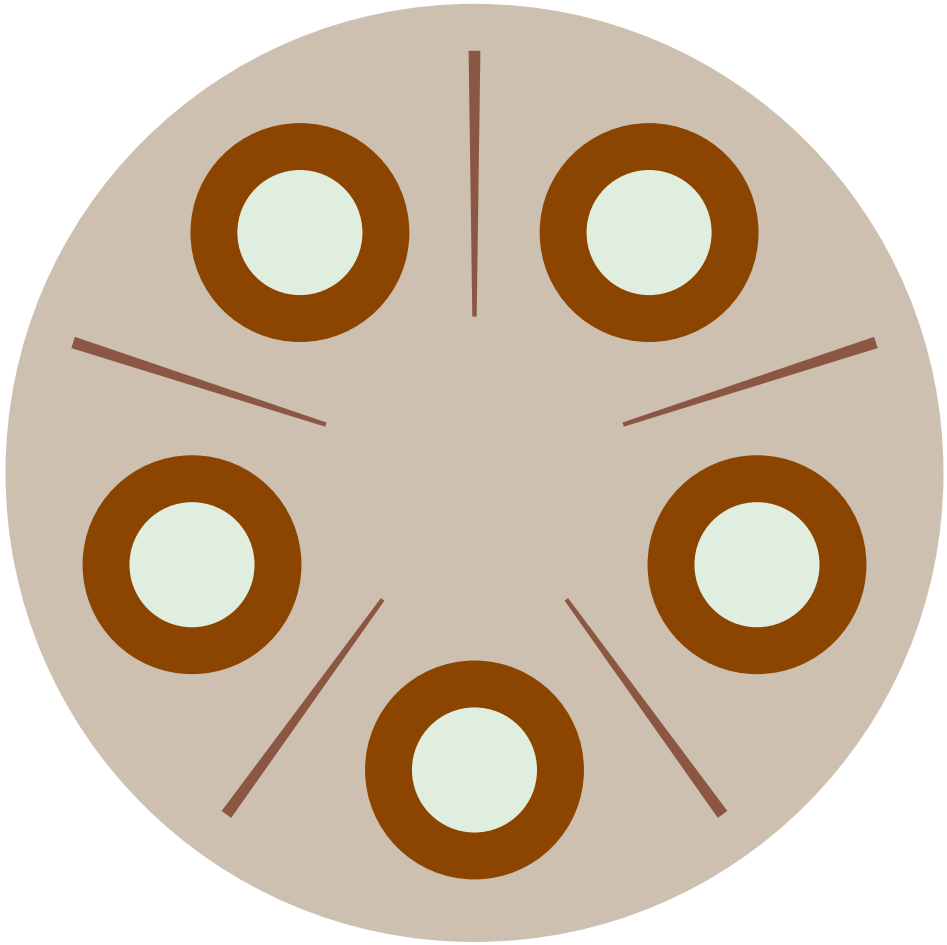

Рис. 9: обеденный стол в задаче о 5 философах

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;
```

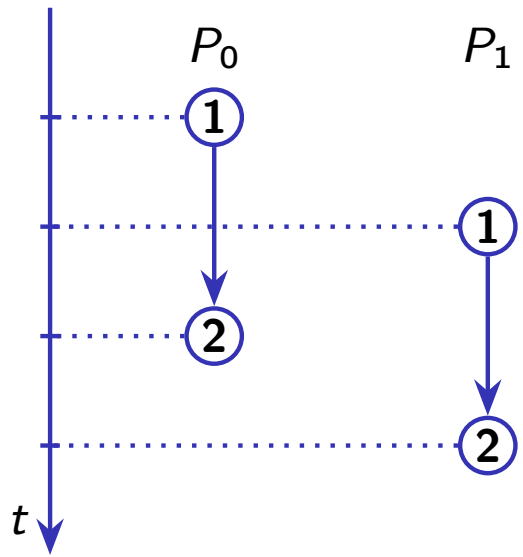

Рис. 10: порядок исполнения

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 взаимоисключения)

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

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимоисключения)

- `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 взаимоисключения)

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

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимоисключения)

- `abFlags[0] == 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 взаимоисключения)

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

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимоисключения)

- `abFlags[0]` == **true** $\Rightarrow$
  - P_1 *вне критического раздела* $\Rightarrow$ P_1 *не сможет зайти в него.*
  - P_1 *в критическом разделе* $\Rightarrow$ `abFlags[1]` == **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 взаимоисключения)

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

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

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимоисключения)

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

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

Работоспособность алгоритма Петерсона

Доказательство (обеспечение взаимоисключения)

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

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

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

- `abFlags[0] == true` $\Rightarrow$
  - P_1 *вне критического раздела* $\Rightarrow$ P_1 *не сможет зайти в него.*
  - P_1 *в критическом разделе* $\Rightarrow$ `abFlags[1] == true` $\Rightarrow$ P_0 *не сможет зайти.*
- *Взаимная блокировка исключена* $\Leftarrow$ *противное: пусть* P_0 *заблокирован* $\Rightarrow$ `abFlags[1] == true` *и* `nTurn == 1`. P_0 *может войти в критический раздел, если* `abFlags[1] == false` *или* `nTurn == 0`. *Случаи:*
  1. P_1 *не намерен входить в критический раздел. Это невозможно* $\Leftarrow$ *иначе* `abFlags[1] == 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. Это невозможно* $\Leftarrow$ *перед* $\forall$ *попыткой* P_1 *должен дать такую же возможность* P_0, *устанавливая* `nTurn = 0` *(не успевает* $\Rightarrow$ `abFlags[1] = 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 // $i \neq 0$
    вернуть „истина“;

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

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 (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

$t \leftarrow m$;
$m \leftarrow r$;
$r \leftarrow t$;

Рис. 12: алгоритм атомарного обмена

Соотношение

$$\mathrm{bLock} + \sum_{i=1}^{n} \mathrm{bKeyI}_i = n$$

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

$$\text{bLock} + \sum_{i=1}^{n} \text{bKeyI}_i = n$$

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 ПУ(s.flag), выполнять
        Пережидание;
    Остальная реализация;
    s.flag ← 0;
```

Рис. 19: алгоритм попытки захвата семафора

```
входные данные:
    s    // переменная-семафор
начало
    пока ПУ(s.flag), выполнять
        Пережидание;
    Остальная реализация;
    s.flag ← 0;
```

Рис. 20: алгоритм освобождения семафора

Задача о читателях и писателях

Постановка задачи

- Чтение данных возможно одновременно любым количеством читателей.
- Запись данных возможна одновременно только одним писателем.
- Во время записи ни один читатель не имеет доступа к данным.

Решение с приоритетным чтением (чтение)

вспомогательные данные:

// глобальные

$n_r = 0$; *// # читающих*

$s_{nr} = 1$; *// для защиты n_r*

$s_w = 1$; *// для записи*

начало

  ждать s_{nr}

    $n_r \leftarrow n_r + 1$;

    если $n_r = 1$, **то**

      Ждать(s_w);

  отпустить

Рис. 21: алгоритм попытки захвата чтения

начало

  ждать s_{nr}

    $n_r \leftarrow n_r - 1$;

    если $n_r = 0$, **то**

      Отпустить(s_w);

  отпустить

Рис. 22: алгоритм освобождения чтения

Решение с приоритетным чтением (запись)

начало
  Ждать(s_w);

Рис. 23: алгоритм попытки захвата записи

начало
  Отпустить(s_w);

Рис. 24: алгоритм освобождения записи

Задача о читателях и писателях (приоритетная запись)

Дополнение к условию

- Чтение данных невозможно, если хотя бы один писатель изъявил о намерении записи.

Дополнительные переменные

s_r: запрещает вход первого читателя при запросе на запись.

n_w: количество писателей, обеспечивает корректную установку s_r.

s_{nw}: гарантирует корректность изменения n_w.

s_{rr}: запрещает вход остальных читателей при запросе на запись.

Решение с приоритетным записи (глобальные данные)

вспомогательные данные: *// глобальные*

$n_r = 0$; *// количество читающих*
$s_{nr} = 1$; *// для защиты n_r*
$s_w = 1$; *// для записи*
$s_r = 1$; *// блокировка одного читателя при запросе на запись*
$n_w = 0$; *// количество пишущих*
$s_{nw} = 1$; *// для защиты n_w*
$s_{rr} = 1$; *// блокировка остальных читателей при запросе на запись*

Рис. 25: инициализация глобальных данных при приоритете записи

Решение с приоритетной записью (чтение)

начало
- **ждать** s_{rr}
  - **ждать** s_r
    - **ждать** s_{nr}
      - $n_r \leftarrow n_r + 1$;
      - **если** $n_r = 1$, **то**
        - Ждать(s_w);
    - **отпустить**
  - **отпустить**
- **отпустить**

Рис. 26: алгоритм попытки захвата чтения

начало
- **ждать** s_{nr}
  - $n_r \leftarrow n_r - 1$;
  - **если** $n_r = 0$, **то**
    - Отпустить(s_w);
- **отпустить**

Рис. 27: алгоритм освобождения чтения

Решение с приоритетной записью (запись)

начало
- **ждать** s_{nw}
  - $n_w \leftarrow n_w + 1$;
  - **если** $n_w = 1$, **то**
    - Ждать(s_r);
- **отпустить**
- Ждать(s_w)

Рис. 28: алгоритм попытки захвата записи

начало
- Отпустить(s_w);
- **ждать** s_{nw}
  - $n_w \leftarrow n_w - 1$;
  - **если** $n_w = 0$, **то**
    - Отпустить(s_r);
- **отпустить**

Рис. 29: алгоритм освобождения записи

Возможные сценарии исполнения алгоритма

Сценарии

Имеются только читатели

Возможные сценарии исполнения алгоритма

Сценарии

Имеются только читатели

1. Устанавливается s_w.
2. Нет очередей.

Возможные сценарии исполнения алгоритма

Сценарии

Имеются только читатели

1. Устанавливается s_w.
2. Нет очередей.

Имеются только писатели

Возможные сценарии исполнения алгоритма

Сценарии

Имеются только читатели

1. Устанавливается s_w.
2. Нет очередей.

Имеются только писатели

1. Устанавливаются s_w и s_r.
2. Есть очередь писателей на s_w.

Возможные сценарии исполнения (продолжение)

Сценарии (продолжение)

Есть читатели и писатели, первым выполняется чтение

Возможные сценарии исполнения (продолжение)

Сценарии (продолжение)

Есть читатели и писатели, первым выполняется чтение

1. Читателем устанавливается s_w.
2. Писателем устанавливается s_r.
3. Все писатели становятся в очередь на s_w.
4. Один читатель становится в очередь на s_r.
5. Остальные читатели становятся в очередь на s_{rr}.

Возможные сценарии исполнения алгоритма (окончание)

Сценарии (окончание)

Есть читатели и писатели, первым выполняется запись

Возможные сценарии исполнения алгоритма (окончание)

Сценарии (окончание)

Есть читатели и писатели, первым выполняется запись

1. Писателем устанавливается s_w.
2. Писателем устанавливается s_r.
3. Все писатели становятся в очередь на s_w.
4. Один читатель становится в очередь на s_r.
5. Остальные читатели становятся в очередь на s_{rr}.

Проблемы, вызванные блокировками

Проблемы производительности

- Дополнительные затраты при доступе к ресурсу.
- Конвой блокировки (lock convoy).

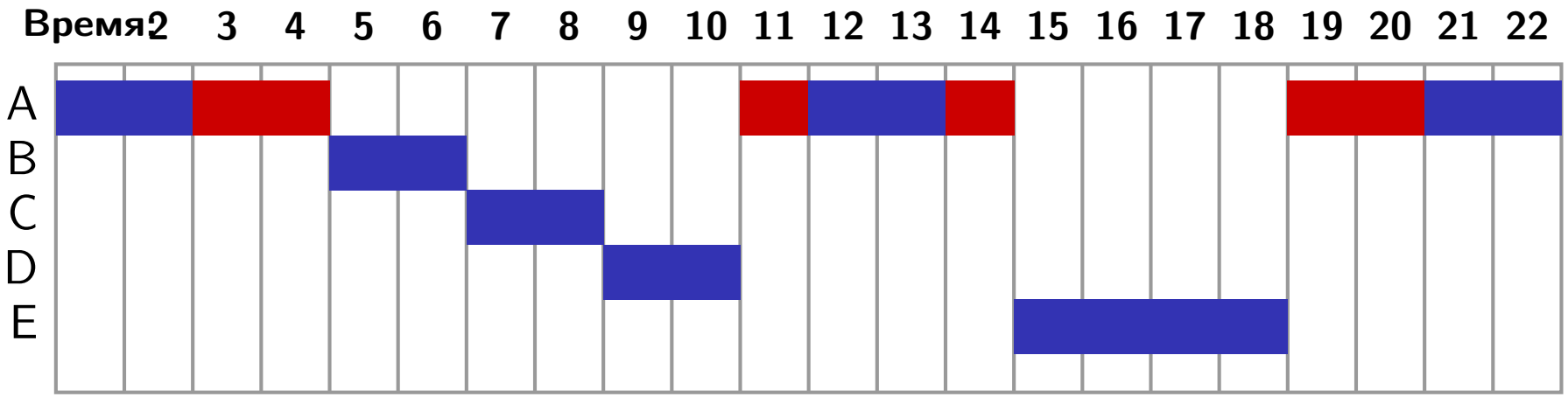

Рис. 30: конвой блокировки

Проблемы, вызванные блокировками (окончание)

Проблемы незавершения

- Соревнование за ресурс (resource contention).
- Мёртвая блокировка (deadlock).
- Живая блокировка (livelock).
- Инверсия приоритета (priority inversion).

Проблемы разработки

- Сложность отладки.
- Слабая комбинируемость блокировок (composability).

Комбинируемость блокировок

Пример

```
class Account
{
  int m_nTotal;
  pthread_mutex_lock m_Lock;
public:
  Account();
  int deposit(int nSum);
  int withdraw(int nSum);
  // ...
};
```

Пример (окончание)

```
int Account::deposit(int nSum)
{
  pthread_mutex_lock(&m_Lock);
  m_nTotal += nSum;
  pthread_mutex_unlock(&m_Lock);
}

int Account::withdraw(int nSum)
{
  deposit(-nSum);
}
```

Комбинируемость блокировок (окончание)

Пример

```
void transfer(Account &rA1, Account &rA2, int nSum)
{
  rA1.withdraw(nSum);
  rA2.deposit(nSum);
}
```

Условия мёртвой блокировки

Необходимые условия возникновения

- Взаимное исключение.
- Захват одной блокировки и ожидание другой.
- Нет принудительного вытеснения.
- Циклическое ожидание ($P_0 \to P_1, P_1 \to P_2, \ldots, P_{n-1} \to P_n, P_n \to P_0$).

Методы работы с мёртвыми блокировками

Методы обработки

- Игнорирование.
- Избежание (алгоритм банкира — Э. Дейкстра).
- Восстановление:
  - Завершение всех процессов;
  - Завершение по одному до разрыва цикла;
  - Отбор ресурсов, откат.

Безопасные состояния

Определение

- Последовательность процессов $\{P_{i_1}, P_{i_2}, \ldots, P_{i_n}\}$ является безопасной для текущего состояния захвата ресурсов, если $\forall j$ ресурсы, которые *может* запросить P_{i_j}, либо свободны, либо заняты $P_{i_k} \quad \forall k < j$.
- Состояние захвата называется безопасным, если для него $\exists$ безопасная последовательность процессов.