Лекция 8. Разбор задач КР и СР

① Решить задачу взаимно исключающего доступа с помощью compare_and_swap (old, new)

② Для этой задачи построить СП, на ней отобразить непрогрессивную некритическую секцию и прогрессивную К.С.

X - регистр x := 1

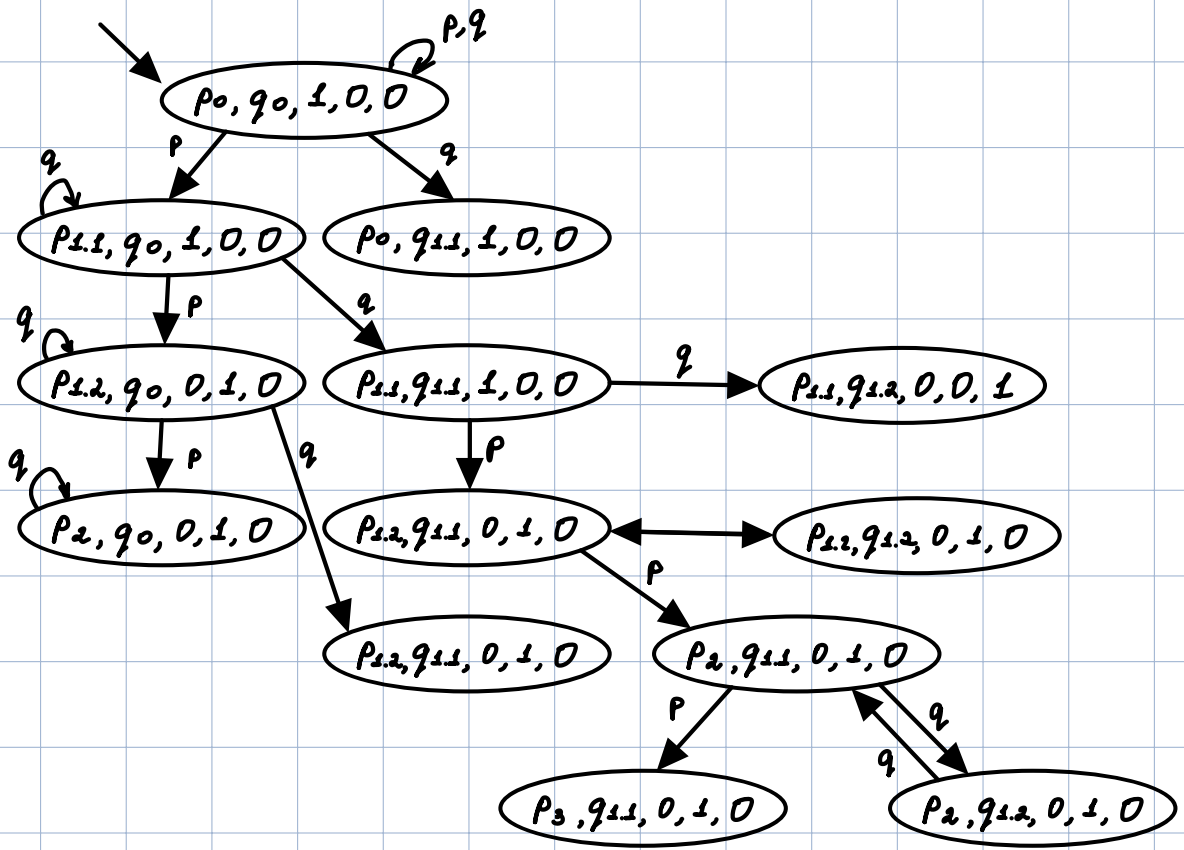
```
acquire_mutex(){
    bool r  - локальная переменная
    repeat {
P1.1.       r := X.compare_and_swap(1,0)
P1.2.   } until (r)
}
release_mutex (){
P       x := 1
}
```

```
compare_and_swap(old,new) {
    if (x = old) {
        x := new
        return (true)
    }
    return (false)
}
```

Реализация

```
loop forever {
P0      // не К.С.
P1      acquire_mutex()
P2      // К.С.
P3      release_mutex()
}
```

Система переходов $< p, q, x, r_p, r_q >$



Из самостоятельной работы

```
int turn := 1

P::                              Q::

loop forever{                    loop forever{
    //не к.с.                        //не к.с.
    while (turn ≠ 1) {               while (turn ≠ 2) {
        skip }                           skip }
    //к.с.                           //к.с.
    turn := 2                        turn := 1
}                                }
```

На Promela

```
int turn = 1
bool crit[2], want[2] = {0, 0}
active [2] proctype P() {
do ::
    do
    :: skip
    :: break
    od
    want[_pid] = 1
    want[_pid] = 0
      (turn == _pid)
    crit[_pid] = 1
    crit[_pid] = 0
    turn = 1 - _pid
od
}
```

нельзя не К.С.

LTL

$[](want[0] \rightarrow \diamond crit[0])$

$[](want[1] \rightarrow \diamond crit[1])$

$[]!(crit[0] \,\&\, crit[1])$

Алгоритм Петерсона для 2 процессов

TURN – MWMR атомарный регистр

TURN[0], TURN[1] – SWMR

переименовали

bool FLAG[0] := 0, FLAG[1] := 0, AFTER_YOU := i $i, j \in \{0, 1\}$

SWMR SWMR MWMR

```
acquire_mutex (i) {
    FLAG[i] = 1
    AFTER_YOU := i
    while (FLAG[j] = 1 & AFTER_YOU = i) {skip}
}
release_mutex (i) {
    FLAG[i] := 0
}
```

Решение n-производителей и m-потребителей с мониторами

неправильное

```
monitor M {
    int k := m + n
    NB_FULL := 0
    BUF [0...k-1]
    cond C_PROD, C_CONS, C_MUTEX
}
B.produce (i) {
    if (NB_FULL = k) {
        C_PROD.wait()
    }
    C.MUTEX.wait()
```

C_MUTEX — какое условие? его нет.

C.MUTEX.wait() — если процесс попал сюда, то он заблокируется навсегда

```
        BUF[in].write(i)
        in := (in+1) mod k
        C_MUTEX.signal()
        NB_FULL := NB_FULL + 1
        C_CONS.signal()
    }
    B.concume(i) {
        if (NB_FULL = 0) {
            C_CONS.wait()
        }
        C_MUTEX.wait()
        ...
    }
```