

OPERATING-SYSTEM CONSTRUCTION

Exercise 7: Interrupt-transparent Queue, SPIN, Task #7

<https://tud.de/inf/os/studium/vorlesungen/betriebssystembau>

TILL SMEJKAL (slides by HORST SCHIRMEIER)

Agenda

- The Interrupt-Transparent Queue
- Correctness Proofs with SPIN
- Lab Task #7

Agenda

- **The Interrupt-Transparent Queue**
- Correctness Proofs with SPIN
- Lab Task #7

Tricky Pointers: Queue in Task #3

- Queue elements inherit from class Chain
 - Thereby, they inherit a pointer to the next element
- A Queue object contains
 - a pointer to the first element
 - **a pointer to a pointer called 'tail'?!**

```
class Chain {  
public:  
    Chain* next;  
};
```

```
class Queue {  
    Chain* head;  
    Chain** tail;  
public:  
    Queue () { head = 0; tail = &head; }  
    void enqueue (Chain* item);  
    Chain* dequeue ();  
    void remove (Chain*);  
};
```

Tricky Pointers: Queue in Task #3

- 'tail' is a pointer to the 'next' pointer in the last element
 - This simplifies enqueueing!

`q.enqueue(&e1)`

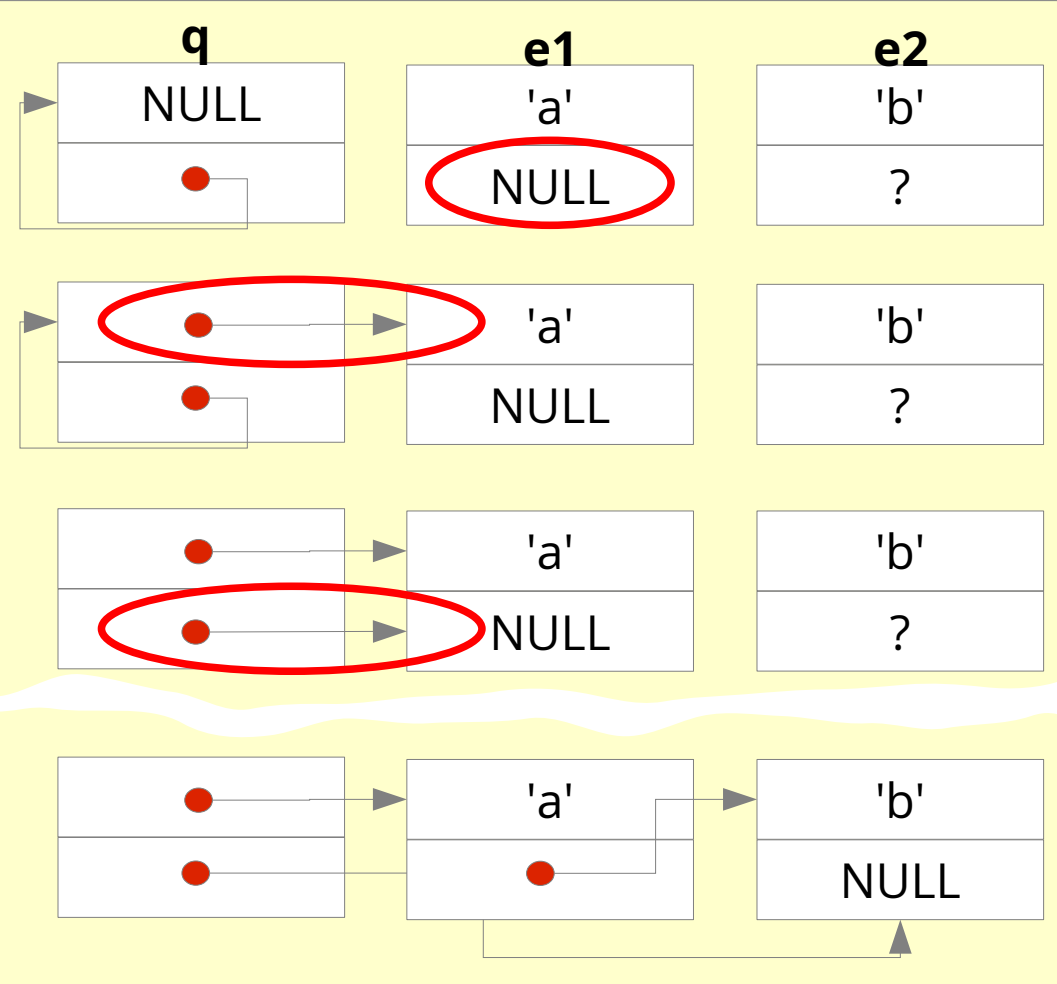
`item->next = NULL;`

`*tail = item;`

`tail = &item->next;`

`q.enqueue(&e2)`

`item->next = NULL;`
`*tail = item;`
`tail = &item->next;`



Interrupt-Transparent Queue

On the whiteboard or in ...

[1] F. Schön, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. ***On Interrupt-Transparent Synchronization in an Embedded Object-Oriented Operating System.*** In The Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000), pages 270-277, Newport Beach, California, March 15-17, 2000. IEEE Computer Society. ISBN 0-7695-0607-0.
doi: [10.1109/ISORC.2000.839540](https://doi.org/10.1109/ISORC.2000.839540)

Agenda

- The Interrupt-Transparent Queue
- **Correctness Proofs with SPIN**
- Lab Task #7

Model Checking (with *SPIN*)

- No deductive mathematical proof!
- Correctness proof by exhaustive analysis of the **state space**
 - State space must be **finite**
 - Fixed number of processes, value ranges, memory, ...
 - Complexity limits for “large” systems / algorithms
 - Verification of a simplified model (→ **Model Checking**)
- Basic principle
 - Model and negated correctness properties → Finite automata
 - Search: Input string that both automata accept (counterexample)

Model Checking with *SPIN*

- Modeling of the system/algorithm in **PROMELA**
 - Very simple/reduced language, similarities with C/Java
 - No pointers, references, functions, classes, generics, ...
 - Reduction to the essential → Minimal state machines
- Specifying correctness properties
 - **assert**(...) statements (local)
 - Linear temporal logic (global)

Example: Mutual Exclusion in C

```
// Peterson's solution to the mutual exclusion problem (1981)
volatile bool flag[2] = {false, false};
volatile int turn;

void P0() {
    flag[0] = true;
    turn = 1;
    while (flag[1] && turn == 1) { /* busy wait */ }
    // critical section
    ...
    // end of critical section
    flag[0] = false;
}

void P1() {
    flag[1] = true;
    turn = 0;
    while (flag[0] && turn == 0) { /* busy wait */ }
    // critical section
    ...
    // end of critical section
    flag[1] = false;
}
```

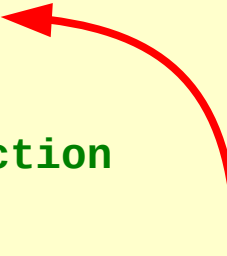
Example: Implementation in PROMELA

```
// Peterson's solution to the mutual exclusion problem (1981)
bool turn, flag[2];           // the shared variables, booleans
byte ncrit;                   // nr of procs in critical section

active [2] proctype user() { // two processes
    assert(_pid == 0 || _pid == 1);
    again:
    flag[_pid] = 1;
    turn = _pid;
    (flag[1 - _pid] == 0 || turn == 1 - _pid);

    ncrit++;
    assert(ncrit == 1);        // critical section
    ncrit--;

    flag[_pid] = 0;
    goto again
}
// analysis:
// $ spin -a peterson.pml
// $ gcc -o pan pan.c
// $ ./pan
```



Boolean expressions
(without **if**) **block**
until the condition
holds

Problem: Model $\leftrightarrow$ Implementation?

Agenda

- The Interrupt-Transparent Queue
- Correctness Proofs with SPIN
- **Lab Task #7**