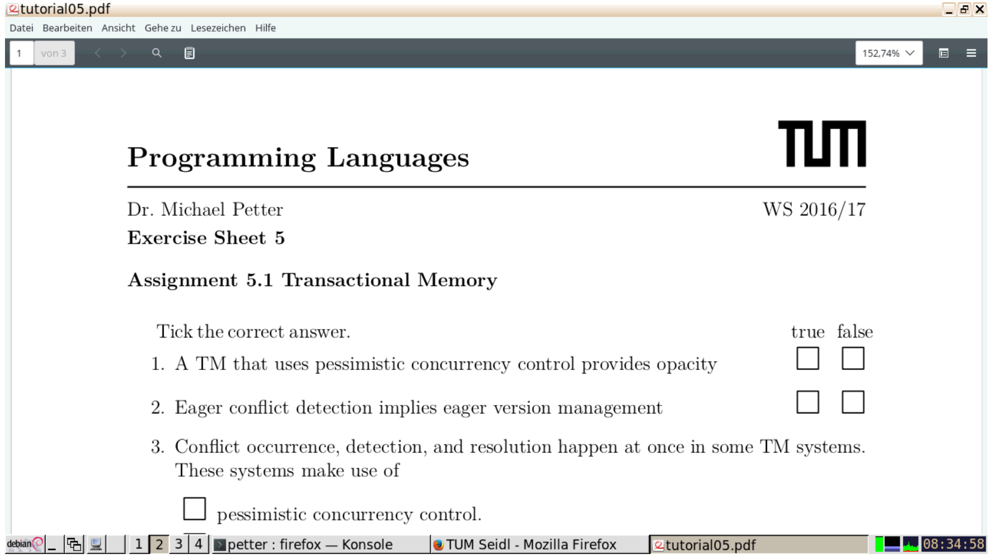
Script **generated by TTT**

Title: Petter: Programmiersprachen_Uebung (25.11.2016)

Date: Fri Nov 25 08:34:59 CET 2016

Duration: 70:51 min

Pages: 14



Programming Languages

Dr. Michael Petter WS 2016/17

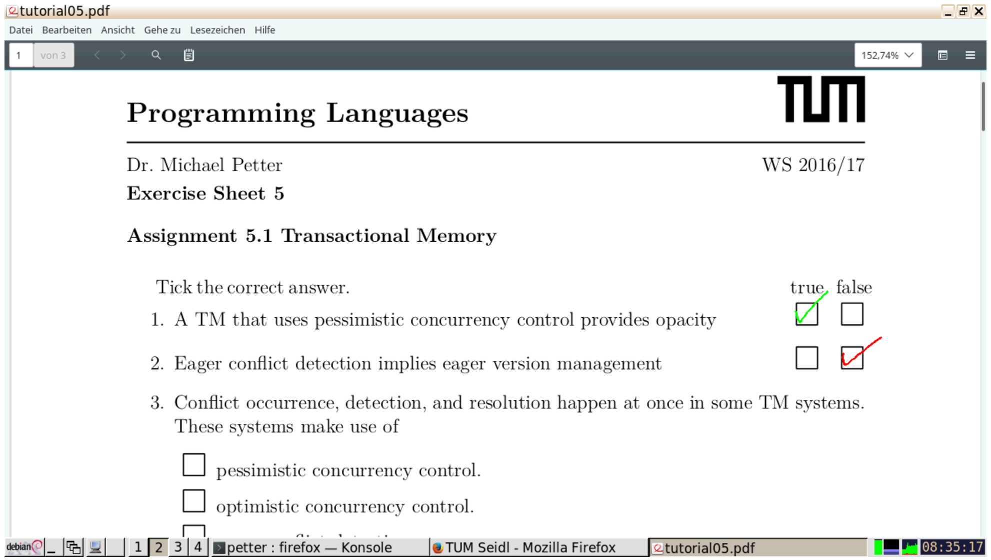
Exercise Sheet 5

Assignment 5.1 Transactional Memory

Tick the correct answer. true false

1. A TM that uses pessimistic concurrency control provides opacity ☐ ☐
2. Eager conflict detection implies eager version management ☐ ☐
3. Conflict occurrence, detection, and resolution happen at once in some TM systems. These systems make use of

☐ pessimistic concurrency control.



Programming Languages

Dr. Michael Petter WS 2016/17

Exercise Sheet 5

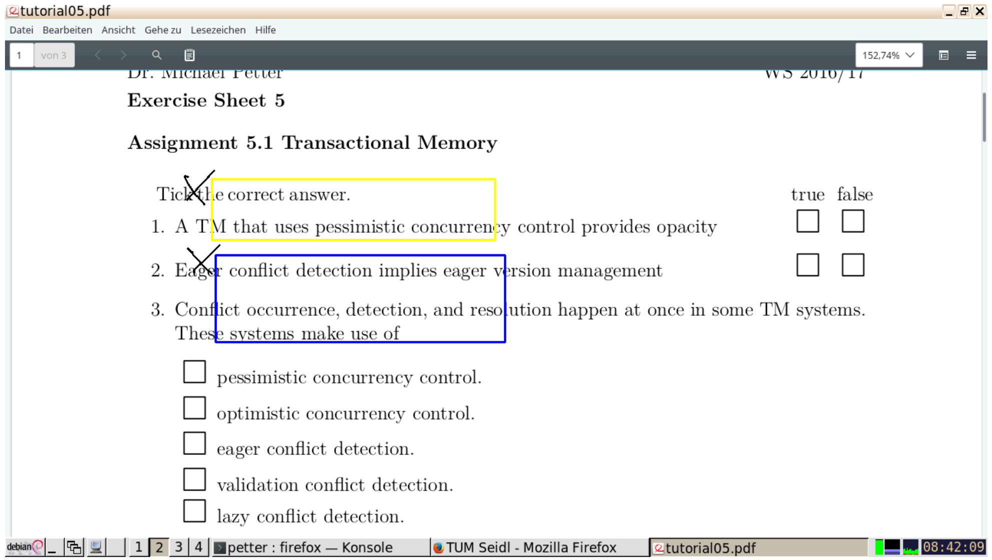
Assignment 5.1 Transactional Memory

Tick the correct answer. true false

1. A TM that uses pessimistic concurrency control provides opacity ☑ ☐
2. Eager conflict detection implies eager version management ☐ ☑
3. Conflict occurrence, detection, and resolution happen at once in some TM systems. These systems make use of

☐ pessimistic concurrency control.

☐ optimistic concurrency control.



Exercise Sheet 5

Assignment 5.1 Transactional Memory

Tick the correct answer. true false

1. A TM that uses pessimistic concurrency control provides opacity ☐ ☐
2. Eager conflict detection implies eager version management ☐ ☐
3. Conflict occurrence, detection, and resolution happen at once in some TM systems. These systems make use of

☐ pessimistic concurrency control.

☐ optimistic concurrency control.

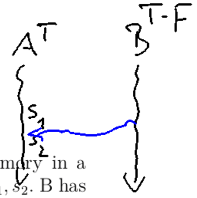
☐ eager conflict detection.

☐ validation conflict detection.

☐ lazy conflict detection.

tutorial05.pdf

- [ ] pessimistic concurrency control.
- [ ] optimistic concurrency control.
- [ ] eager conflict detection.
- [ ] validation conflict detection.
- [ ] lazy conflict detection.



4. A conflict occurs between two threads A and B where B accesses memory in a non-transactional way. The transaction of A consists of two statements s_1, s_2. B has observed the state after A has executed s_1 but before A has executed s_2.

- [ ] A and B will abort.
- [ ] The TM system uses lazy version management.
- [ ] The TM does not provide single-lock atomicity.
- [ ] The TM does not provide transactional sequential consistency.

5. A zombie transaction crashes the whole program due to reading unexpected and

08:44:47

tutorial05.pdf

- [ ] The TM does not provide transactional sequential consistency.

5. A zombie transaction crashes the whole program due to reading unexpected and inconsistent values. The TM implementation

- [ ] provides opacity.
- [ ] uses pessimistic concurrency control.
- [x] uses eager version management.

6. A TM implementation detects a conflict in the middle of a transaction although the actual write that created the conflict lies in the past. This kind of conflict detection is called

- [ ] eager conflict detection.
- [ ] tentative conflict detection.
- [ ] validation conflict detection.
- [ ] lazy conflict detection.

08:50:12

tutorial05.pdf

- [ ] uses eager version management.

6. A TM implementation detects a conflict in the middle of a transaction although the actual write that created the conflict lies in the past. This kind of conflict detection is called

- [ ] eager conflict detection.
- [ ] tentative conflict detection.
- [x] validation conflict detection.
- [ ] lazy conflict detection.

1

08:52:19

tutorial05.pdf

is called

- [ ] eager conflict detection.
- [ ] tentative conflict detection.
- [ ] validation conflict detection.
- [ ] lazy conflict detection.

1

7. A TM implementation uses optimistic concurrency control. Which of the following statements must be false?

- [ ] Because a transaction is a zombie, the program crashes after a write through a pointer that is NULL.

08:54:07

7. A TM implementation uses optimistic concurrency control. Which of the following statements must be false?

- [ ] Because a transaction is a zombie, the program crashes after a write through a pointer that is `NULL`.
- [ ] The implementation checks for conflicts only when committing.
- [ ] A transaction can be suspended until a conflicting transaction has finished.

Assignment 5.2 **Restricted Transactional Memory**

Consider the following code fragment on a machine with RTM and Caches:

```
int data = 0;
int s=0;
```

thread P_1:

```
int data = 0;
int s=0;
```

thread P_0:

```
while (s!=-1)
  if((s=_xbegin())==-1){
    data++;
    _xend();
  }
```

thread P_1:

```
if (_xbegin()==-1){
  data++;
  _xend();
}else {
  data++;
}
```

1. Fill in the gap with either “**will**”, “**will not**” or “**may or may not**”:

   After P_0 and P_1 both terminate, `data` __________ evaluate to 1.

---

thread P_0:

```
while (s!=-1)
  if((s=_xbegin())==-1){
    data++;
    _xend();
  }
```

```
if (_xbegin()==-1){
  data++;
  _xend();
}else {
  data++;
}
```

1. Fill in the gap with either “**will**”, “**will not**” or “**may or may not**”:

   After P_0 and P_1 both terminate, `data` *may or may not* evaluate to 1.

2. Fill in the gap with either “**will**”, “**will not**” or “**may or may not**”:

   After P_0 and P_1 both terminate, `data` *will not* evaluate to 3.

3. Consider the following interleaving of paths through the program:

```
  if((s=_xbegin())==-1){
                                _xend();
    data++;
                              }else {
    _xend();
                                data++;
  }
```

1. Fill in the gap with either “**will**”, “**will not**” or “**may or may not**”:

   After P_0 and P_1 both terminate, `data` __________ evaluate to 1.

2. Fill in the gap with either “**will**”, “**will not**” or “**may or may not**”:

   After P_0 and P_1 both terminate, `data` __________ evaluate to 3.

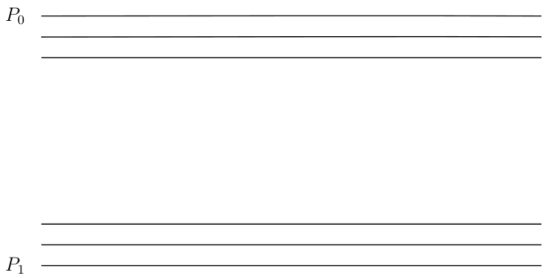
3. Consider the following interleaving of paths through the program:

   P_0 `s!=-1 (s=_xbegin())==-1 data++; s!=-1 (s=_xbegin())==-1 data++ _xend()`
   P_1 `_xbegin()==-1 data++;`

   Draw a happened-before diagram of this interleaving. The initial cache states for `s` and `data` are $S0, S0$.

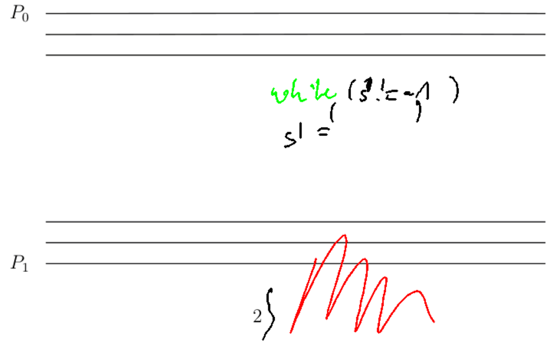
P_0 `s!=-1 (s=_xbegin())==-1 data++; s!=-1 (s=_xbegin())==-1 data++ _xend()`
P_1 `_xbegin()==-1 data++;`
→ time

Draw a happened-before diagram of this interleaving. The initial cache states for **s** and **data** are $S0, S0$.



2

Draw a happened-before diagram of this interleaving. The initial cache states for **s** and **data** are $S0, S0$.




```
int data = 0;
int s=0;
```

thread P_0:

```
while (s!=-1)
  if((s=_xbegin())==-1){
    data++;
    _xend();
  }
```

thread P_1:

```
if (_xbegin()==-1){
  data++;
  _xend();
}else {
  data++;
}
```

1. Fill in the gap with either "**will**", "**will not**" or "**may or may not**":

   After P_0 and P_1 both terminate, data ______________ evaluate to 1.