Lab 3

Pia Carstens

Nike Pulow

Group 9

Tasks 1, 2, 3, 4

Lab 3 - Task 1

Lamport Timestamps:

- causally related events are ordered correctly
- no detection of concurrency
- simple implementation
- consists of one value O(1)

Vector Clocks:

- can identify causal relation and concurrency
- more complex implementation
- n entries for n processes O(n)

Lab 3 - Task 1

- weak consistency cannot distinguish causally related from concurrent events:

```
y -> x => C(x) !< C(y)
```

- strong consistency detects causality and concurrency:

```
VC(x) < VC(y) => x -> y
```

(x happened before y)

```
VC(x) || VC(y) => x || y
```

(x and y are concurrent)

Lab 3 - Task 2

server.py ll. 232-252

```
def create_entry_request(self):
        [...]
            with self.lock:
                [...]
                with self.lock:
                    self.clock.increment(self.id)
                create_ts = self.clock.copy()
                entry = Entry(entry_id, entry_value, create_ts)
                self.board.add_entry(entry)
                for other in self.server_list:
                    message = (other, {‘type’: ‘propagate’, ‘entry_value’: entry_value,
                                       ‘entry_id’: entry_id, ‘timestamp’: create_ts.to_list(),
                                       ‘sent_from’: self.id})
                    self.queue_out.put(message)
        [...]
```

- increment clock, copy clock to create_ts and add entry
- send message to every server, including where message came from

Lab 3 - Task 3

server.py ll. 338-356

```
def handle_message(self, message):
        [...]
        if type == ‘propagate’:
                entry_value = message[‘entry_value’]
                entry_id = message[‘entry_id’]
                if len(message[‘timestamp’]) == len(self.clock.to_list()):
                    entry_timestamp = VectorClock.from_list(entries = message[‘timestamp’])
                    with self.lock:
                            if not self.id == message[‘sent_from’]:
                            self.clock.increment(self.id)
                        self.clock.update(entry_timestamp)
                        self.status[‘num_entries’] += 1
                        entry = Entry(entry_id, entry_value, entry_timestamp)
                        self.board.add_entry(entry)
        [...]
```

- check, if propagate message comes from self -> don’t increment clock twice
- update timestamp and add entry to board

Lab 3 - Task 3

server.py ll. 65-73

```
def get_ordered_entries(self):
        entries = [e for e in list(self.indexed_entries.values()) if
                   not e.is_deleted()]
        sorted_entries = sorted(entries, key=lambda x: (x.create_ts, x.id))
        return sorted_entries
[...]
```

- sort by create_ts first, then id
- might be useful to sort by modify_ts first to put newly updated at top of the list

Lab 3 - Task 4

server.py ll. 260-275

```
def update_entry_request(self, entry_id):
        [...]
            entry_value = request.forms.get(‘value’)
            with self.lock:
                [...]
                entry = self.board.indexed_entries.get(entry_id)
                if entry is None or entry.is_deleted():
                    return {‘error’: ‘entry does not exist or has been deleted.’}
                self.clock.increment(self.id)
                entry.value = entry_value
                entry.modify_ts = self.clock.copy()

                [...]
```

- get entry from **`entry_id`** from indexed_entries list
- increment own clock, update value, set **`modify_ts`**
- propagate message to all servers

Lab 3 - Task 4

server.py ll. 385-372

```
def handle_message(self, message):
        [...]
        elif type == ‘modify’:
        entry_id = message[‘entry_id’]
        modify_ts = VectorClock.from_list(entries = message[‘timestamp’])
        entry_value = message[‘entry_value’]
        entry = self.board.indexed_entries.get(entry_id)
        if entry is None or entry.is_deleted():
            return {‘error’: ‘entry does not exist or has been deleted.’}
        with self.lock:
            if not self.id == message[‘sent_from’]:
                self.clock.increment(self.id)
            self.clock.update(modify_ts)
            if entry.modify_ts is None or entry.modify_ts < modify_ts:
                entry.value = entry_value
                entry.modify_ts = modify_ts
                self.board.add_entry(entry)
      [...]
```

Lab 3 - Task 4

server.py ll. 292-305

```
def delete_entry_request(self, entry_id):
        try:
            [...]
            entry = self.board.indexed_entries.get(entry_id)
            if entry is None or entry.is_deleted():
                    return {'error': 'entry does not exist or has been deleted.'}
            with self.lock:
                print("Deleting entry with id {}".format(entry_id))

                self.clock.increment(self.id)
                entry.delete_ts = self.clock.copy()
                self.board.add_entry(entry)

                [...]
```

- get entry form board and check if it's already been deleted
- increment own clock, update timestamp and add changes to board

Lab 3 - Task 4

server.py ll. 260-275

```
def handle_message(self, message):
        [...]
        elif type == ‘delete’:
            entry_id = message[‘entry_id’]
            delete_ts = VectorClock.from_list(entries = message[‘timestamp’])
            entry = self.board.indexed_entries.get(entry_id)
            if entry is None or entry.is_deleted():
                return {‘error’: ‘entry does not exist or has been deleted.’}
            with self.lock:
                if not self.id == message[‘sent_from’]:
                    self.clock.increment(self.id)
                self.clock.update(delete_ts)
                if entry.delete_ts is None or entry.delete_ts < delete_ts:
                    entry.delete_ts = delete_ts
                    self.board.add_entry(entry)
        [...]
```