```
atomic<uint64_t> id_gen(1)
atomic<uint64_t> clock(0);
atomic<uint64_t> lock_table[NUM_LOCKS];

atomic<uint64_t> *get_lock(void *addr) {
  return &lock_table[(((uintptr_t)addr)>>GRAIN) % NUM_LOCKS];
}
struct Descriptor {
  jmp_buf *checkpoint;
  uint64_t my_lock;
  uint64_t start_time;
  unordered_map<uintptr_t*, uintptr_t> writes;
  vector<atomic<uint64_t>*> reads;
  vector<pair<atomic<uint64_t>*, uint64_t>> locks;

  Descriptor() : my_lock(((id_gen++)<<1)|1) { }
};
void beginTx(jmp_buf *b) {
  checkpoint = b;
  start_time = clock;
}
void write(uintptr_t *addr, uintptr_t val) {
  writes.insert_or_assign(addr, val);
}
int read(uintptr_t *addr) {
  auto it = writes.find(addr);
  if (it != writes.end())
    return *it;

  atomic<uint64_t>* l = get_lock(addr);
  uint64_t pre = *l;
  uintptr_t val = std::atomic_ref<uintptr_t>(*addr).load(std::memory_order_acquire);
  uint64_t post = *l;
  if ((pre&1) || (pre != post) || (pre > start_time))
    abortTx();
  reads.push_back(l);
  return val;
}
```

FIGURE 20.11 Software transactions with ownership records (1/2).

Copyright © 2021 Elsevier Inc. All rights reserved.