Abstract

How to implement concurrent, reliable system software with transactions. Theory and practice.
Concurrency and error handling is usually complicated to implement and test. In this presentation, we'll see how both can be handled by the transaction concept.
We'll examine the I/O code of two example programs implemented in C. We'll first look at basic problems and afterwards how transactions can help to solve these.
On the practical side, we'll talk about the software *picotm*, a system-level transaction manager for POSIX systems. Reimplementing the example programs on top of picotm will make them thread-safe and less error prone.
Picotm can handle arbitrary resources. In the presentation's final part, we'll look at the functionality that is currently provided, such as transactional memory, C string and memory functions, memory allocation, file-descriptor I/O, and others.

System-Level Transactions with *picotm*

Thomas Zimmermann

March 9, 2018

Handling Errors

Transitioning through consistent states; doing I/O in between.

```
1   int fd0, fd1; /* file descriptors */
2
3   char ibuf[100]; /* input buffer */
4   char obuf[100]; /* output buffer */
5
6   while (true) {
7
8       wait_for_input();
9
10      fill_input_buffer(ibuf);
11
12      compute_output_buffer(ibuf, obuf);
13
14      write(fd0, obuf, sizeof(obuf));
15      write(fd1, obuf, sizeof(obuf));
16  }
```

Handling Concurrency

Two threads writing concurrently to the same file.

```
1   int fd; /* file descriptor */
2
3   void thread_1_func() {
4       char obuf[100]; /* output buffer */
5
6       compute_output_buffer(obuf); /* obuf = "42" */
7
8       pwrite(fd, obuf, sizeof(obuf), 256);
9   }
10
11  void thread_2_func() {
12      char ibuf[100]; /* input buffer */
13
14      pread(fd, ibuf, sizeof(ibuf), 256); /* ibuf = ? */
15
16      process_input_buffer(ibuf);
17  }
```

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic Our second example shall not write partial strings.

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic Our second example shall not write partial strings.

Consistent Our first example shall output consistent data to both files.

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic Our second example shall not write partial strings.

Consistent Our first example shall output consistent data to both files.

Isolated Threads in our second example shall not interfere.

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic	Our second example shall not write partial strings.
Consistent	Our first example shall output consistent data to both files.
Isolated	Threads in our second example shall not interfere.
Durable	Bonus point: Once we made a write, it should not disappear.

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic	Our second example shall not write partial strings.
Consistent	Our first example shall output consistent data to both files.
Isolated	Threads in our second example shall not interfere.
Durable	Bonus point: Once we made a write, it should not disappear.

- So we actually wanted transactional semantics!

What We Actually Wanted

- To fix our examples, we have to ensure a number of constraints.

Atomic	Our second example shall not write partial strings.
Consistent	Our first example shall output consistent data to both files.
Isolated	Threads in our second example shall not interfere.
Durable	Bonus point: Once we made a write, it should not disappear.

- So we actually wanted transactional semantics!
- Many databases around, but hardly anything for arbitrary software.

Intermezzo: A Closer Look at pwrite()

ssize_t pwrite(int fd, const void* buf, size_t count, off_t offset)

Intermezzo: A Closer Look at pwrite()

ssize_t pwrite(int fd, const void* buf, size_t count, off_t offset)

- Writes data to a specific location in a file

Intermezzo: A Closer Look at pwrite()

ssize_t pwrite(int fd, const void* buf, size_t count, off_t offset)

- Writes data to a specific location in a file

Execute	Add data to transaction's write set.
Apply	Write-out data from write set to file during commit.
Undo	Remove data from write set during roll back.

Intermezzo: A Closer Look at pwrite()

ssize_t pwrite(int fd, const void* buf, size_t count, off_t offset)

- Writes data to a specific location in a file

Execute Add data to transaction's write set.
Apply Write-out data from write set to file during commit.
Undo Remove data from write set during roll back.

- Enter *picotm* ***drum rolls***

Put into Practice with *picotm*

- *picotm* is a transaction manager for C applications and firmware

Put into Practice with *picotm*

- *picotm* is a transaction manager for C applications and firmware

Basic C interface of *picotm*

```
1  picotm_begin
2      /* execution phase; put your code here */
3
4  picotm_commit  /* commit phase; provided by picotm */
5
6      /* recovery phase; put your error handling here */
7  picotm_end
```

Handling Errors, transactionally

Transitioning through consistent states; doing transactional I/O in between.

```
1   int fd0, fd1; /* file descriptors */
2
3   char ibuf[100]; /* input buffer */
4   char obuf[100]; /* output buffer */
5
6   while (true) {
7
8       wait_for_input();
9
10      picotm_begin
11
12          fill_input_buffer_tx(ibuf);
13
14          compute_output_buffer_tx(ibuf, obuf); /* does malloc_tx() and free_tx() */
15
16          write_tx(fd0, obuf, sizeof(obuf));
17          write_tx(fd1, obuf, sizeof(obuf));
18
19      picotm_commit
20
21          if (picotm_error_is_non_recoverable()) {
22              notice_admin_and_abort();
23          } else {
24              handle_error_and_retry();
25              picotm_restart();
26          }
27
28      picotm_end
29  }
```

Transaction Log

- *picotm* keeps a log of all operations that are
  - delayed until commit time, or
  - reverted during a rollback.

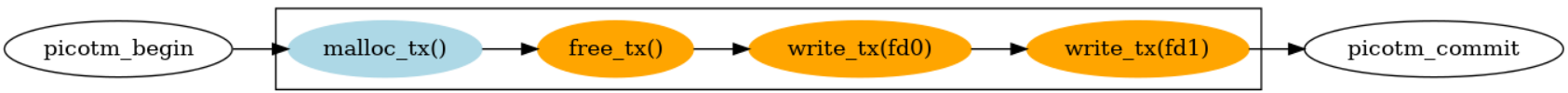


Figure: The complete transaction log for example 1. Delayed operations are displayed in Orange, revertable operations are in Light Blue.

Handling Concurrency, transactionally

Two threads writing transactionally to the same file.

```
1   int fd; /* file descriptor */
2
3   void thread_1_func()
4   {
5       picotm_begin
6           char obuf[100]; /* output buffer */
7           compute_output_buffer_tx(obuf); /* obuf = "42" */
8           pwrite_tx(fd, obuf, sizeof(obuf), 256);
9       picotm_commit
10          if (picotm_error_is_non_recoverable()) {
11              notice_admin_and_abort();
12          } else {
13              handle_error_and_retry();
14              picotm_restart();
15          }
16      picotm_end
17  }
18
19  void thread_2_func()
20  {
21      char ibuf[100]; /* input buffer */
22
23      picotm_begin
24          pread_tx(fd, ibuf, sizeof(ibuf), 256); /* ibuf = "42" */
25      picotm_commit
26          [...]
27      picotm_end
28
29      process_input_buffer(ibuf);
30  }
```

Modules of *picotm*

- All application functionality is provided by modules
- Modules can be combined as needed
- New modules can be added

Module interface for interacting with *picotm*.

```
1  /* Register a module */
2  struct picotm_module_ops {
3      picotm_module_lock_function lock;
4      picotm_module_unlock_function unlock;
5      picotm_module_is_valid_function is_valid;
6      picotm_module_apply_function apply;
7      picotm_module_undo_function undo;
8      picotm_module_apply_events_function apply_events;
9      picotm_module_undo_events_function undo_events;
10     picotm_module_update_cc_function update_cc;
11     picotm_module_clear_cc_function clear_cc;
12     picotm_module_finish_function finish;
13     picotm_module_uninit_function uninit;
14 };
15 unsigned long
16 picotm_register_module(const struct picotm_module_ops* ops);
17
18 /* Append event to transaction log */
19 void
20 picotm_append_event(unsigned long module, unsigned long op, uintptr_t cookie);
21
22 /* Inform picotm about an error */
23 void
24 picotm_recover_from_error(const struct picotm_error* error);
```

Transactional Memory

- load_tx()
- store_tx()
- privatize_tx()

```
1  int tx_x = 1;
2
3  picotm_begin
4      int x = load_int_tx(&tx_x);
5      x += 1;
6      store_int_tx(&tx_x, x);
7  picotm_commit
8      [...]
9  picotm_end
```

String and Memory helpers

- memcpy_tx(), memcmp_tx(), etc.
- strcpy_tx(), strcmp_tx(), etc.

```
1 |     char tx_buf[20];
2 |
3 |     picotm_begin
4 |         memset_tx(tx_buf, 0, sizeof(tx_buf));
5 |     picotm_commit
6 |         [...]
7 |     picotm_end
```

Memory Allocation

- malloc_tx()
- free_tx()

```
1 |   picotm_begin
2 |       char* buf = malloc_tx(20);
3 |       /* do something with 'buf' */
4 |       free_tx(buf);
5 |   picotm_commit
6 |       [...]
7 |   picotm_end
```

Safe Type Casting and Arithmetic

- No more overflows, underflows or div-by-zero errors

```
1   int tx_x = 1;
2
3   picotm_begin
4       int x = load_int_tx(&tx_x);
5       short x16 = cast_int_to_short_tx(x);
6
7       x16 = mul_short_tx(x16, 2);
8       x16 = add_short_tx(x16, 5);
9       x16 = div_short_tx(x16, 3);
10
11      x = cast_short_to_int_tx(x16); /* always correct acto C Standard */
12
13      store_int_tx(&tx_x, x);
14  picotm_commit
15      [...]
16  picotm_end
```

Data Structures

- Transactional lists, queues, multisets, stacks
- Interfaces similar to C++ STL

```
1  struct txlist_state tx_list_state; /* non-transactional list state */
2
3  picotm_begin
4      struct txlist_entry* entry = malloc_tx(sizeof(*entry));
5      txlist_entry_init_tm(entry);
6
7      struct txlist* list = txlist_of_state_tx(&tx_list_state);
8
9      txlist_push_back_tx(list, entry);
10 picotm_commit
11     [...]
12 picotm_end
```

File I/O

- open_tx(), close_tx()
- read_tx(), write_tx()
- pread_tx(), pwrite_tx()

```
1  int fd; /* file descriptor */
2  char buf[20];
3
4  picotm_begin
5      read_tx(fd, buf, sizeof(buf));
6  picotm_commit
7      [...]
8  picotm_end
```

Others

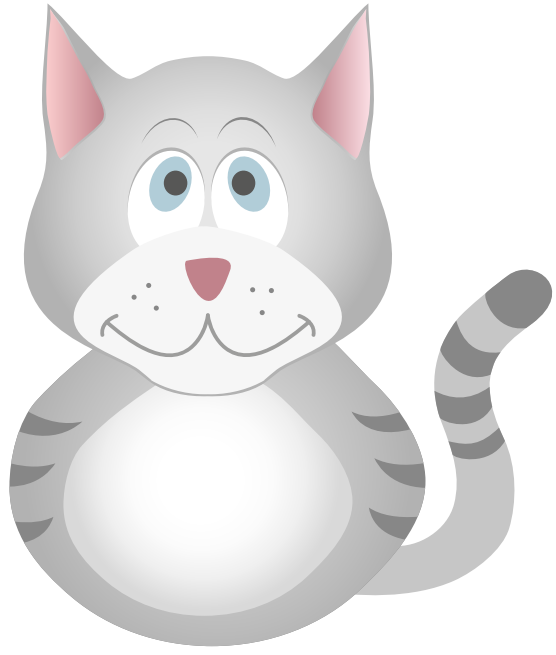
- errno
- C Standard Math Library
  - Math functions
  - Floating-Point environment
  - Floating-Point exceptions
- Some VFS support

Ideas and TODO List

- Support for tiny systems
- Unix signal handling
  - SIGSEG, SIGBUS, SIGILL
- 2-phase commits
  - Data formats
  - Network protocols

Summary

- Transactional code is safer and less error prone than traditional one.
- Implement error handling and concurency control *exactly* once.
- *picotm* is available as Open Source at
  - picotm.org
- More information, tutorials, background on my blog at
  - transactionblog.org
  - twitter.com/transactionblog
- Or reach out to me via
  - tdz@users.sourceforge.net



picotm.org