AMD

SUPPORTING NATIVE PTHREADS IN SYSCALL EMULATION MODE

BRANDON POTTER

JUNE 14TH, 2015

WHAT ARE M5THREADS' PROBLEMS?

AMD

- Gem5 currently supports pthreads using the gem5-specific m5threads library.
- M5threads is not a complete pthread library.
  - Some runtime frameworks need extensive support.
- M5threads is not what a real system would run.
- Confuses new users.
  - Users need to learn that the library exists.
  - Then they need to compile and link against it.
  - Only supports archive format.
  - Hopefully, no one runs into any problems:
    - https://www.mail-archive.com/gem5-dev@m5sim.org/msg07204.html
    - http://www.mail-archive.com/gem5-users%40gem5.org/msg11054.html

OUTLINE

AMD

- What is the Native Posix Thread Library (NPTL)?
- Which system calls needed modifications?
  - `clone`
  - `futex`
  - `set_tid_address`
  - `exit / exit_group`
  - `execve`
- How extensively do these changes support the pthread API?
- Putting it all together diagram. (time permitting)

WHAT IS THE NATIVE POSIX THREAD LIBRARY?

AMD

- NPTL is the POSIX thread library that comes with GLIBC.
- POSIX compliant pthread library.
- Replaced earlier non-compliant libraries circa 2005 (early Linux 2.6 kernels).
- Tightly coupled with Linux kernel.

CLONE

AMD

- `Clone` is responsible for both thread and process creation.
  - Gem5 had a prior version of `clone` that supported thread creation only.
- LiveProcess duplicated inside `clone` and bound to new ThreadContext.
  - Attributes are specified in the flags argument. (man 2 clone)
- Added support for TLS, thread groups, and `futex` support:
  - Required adding the following flags: `CLONE_THREAD, CLONE_PARENT_SETTID, CLONE_CHILD_CLEARTID, CLONE_SETTLS.`
- ThreadContext is statically allocated at runtime on command line.
  - Ownership changes dynamically at runtime; not currently recycled after use.

FUTEX

AMD

- `Futex` is the synchronization mechanism for pthreads.
  - Operates at the boundary of userspace and the kernel.
  - Only need to call into a `futex` system call if the lock is contended.
  - User depends on kernel to put the contended thread to sleep and awakens it later when another thread as finished with the lock.

- Originally written by Daniel Sanchez while he was at Stanford.

- Basic operation uses two methods, `FUTEX_WAIT` (sleep) and `FUTEX_WAKE`.

- `Futex` needed to be extended to work with thread groups and we refactored the code into its own class (Thanks to Alexandru Dutu – AMD Research).

SET_TID_ADDRESS

AMD

- The Linux kernel has two fields for each process, `set_child_tid` and `clear_child_tid`, that start out as NULL.
- The `set_child_tid` field indicates the address where the child should write its PID at startup.
- The `clear_child_tid` field indicates the address where the child should clear its PID and call a `futex` wakeup on.
- This functionality helps notify the parent thread (thread group leader) when child threads finish their tasks.

EXIT / EXIT_GROUP

AMD

- `Exit` ends process execution and returns an `exit` status to its parent.

- `Exit_group` ends execution of an entire thread group.
  - Thread group is a set of threads that share a thread group leader.
  - The thread group leader is the process which starts cloning threads.

- Added support to `exit` and implemented `exit_group`.
  - When `exit_group` is called, all threads for that thread group leader need to exit; achieve this by checking this state when system calls are called.
  - If `CHILD_CLEARTID` was set, need to call `FUTEX_WAKE` on that address.
  - Calling `exit` no longer exits simulation until the last context has finished running.
  - Releases LiveProcess state, such as file descriptors, inside of `exit`.

EXECVE

AMD

- Not directly related to pthread support, but can now support multiple processes with moderate effort.

- Completely new, there was no notion of `execve` in the code previously.

- `Execve` must do the following:
  - Load object file.
  - Inherit file descriptors from parent when appropriate.
  - Set standard file descriptor to their defaults.
  - Supply argv and envp to process constructor.
  - Setup process identifier information (uid, pid, ppid, etc.)
  - Reset `SIGCHLD` value.
  - Setup the thread context.

PTHREADS API COVERAGE

AMD

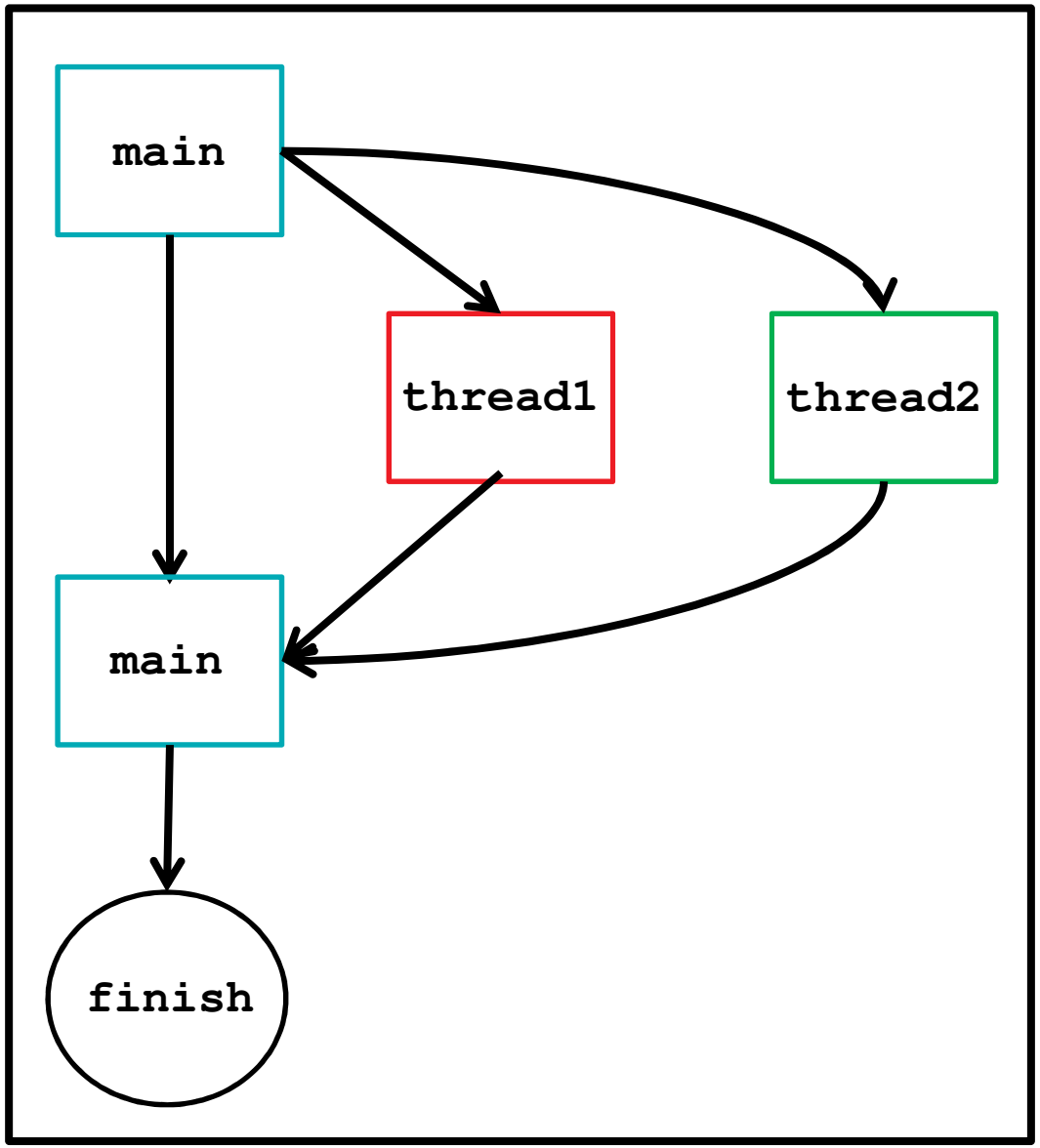
- `pthread_create`
- `pthread_exit`
- `pthread_join`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`
- `pthread_mutex_init`
- `pthread_mutex_destroy`
- `pthread_cond_signal`
- `pthread_cond_wait`
- `pthread_attr_init`
- `pthread_attr_destroy`
- `pthread_attr_setdetachstate`
- `pthread_attr_getstacksize`
- `pthread_attr_setstacksize`

CONCLUSION

AMD

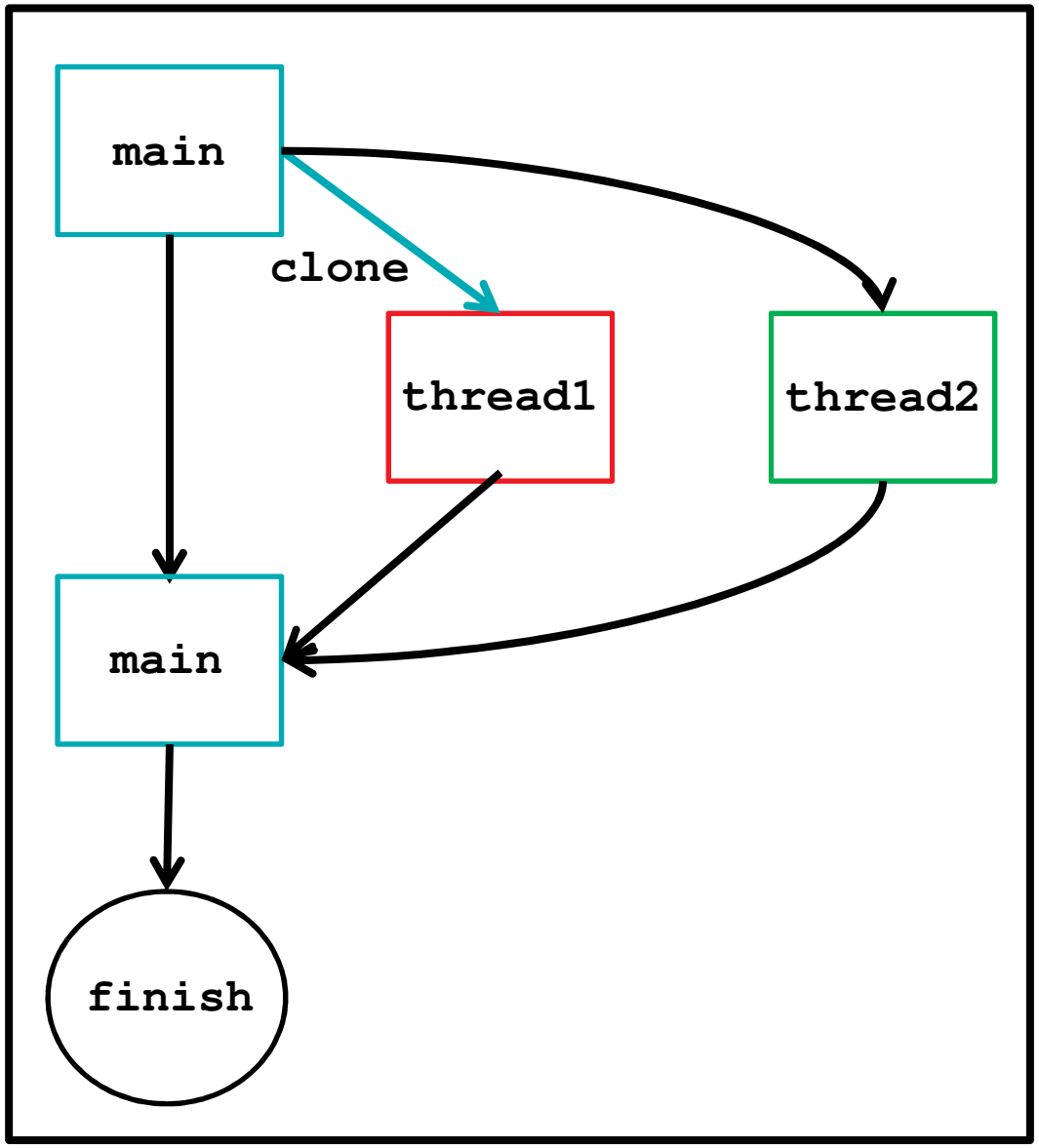
- Support for Linux NPTL library is preferable to workarounds like m5threads.
- Required changes to `clone`, `set_tid_address`, `futex`, `exit`, and `exit_group` to support the API presented in previous slide.
- Execution of concurrent processes in the future with `execve` and some additional support.
- Possible future work:
  - Address `futex` corner cases (several options are unsupported).
  - More support for `clone` options.
  - Full support for all ISAs.

PUTTING PIECES TOGETHER



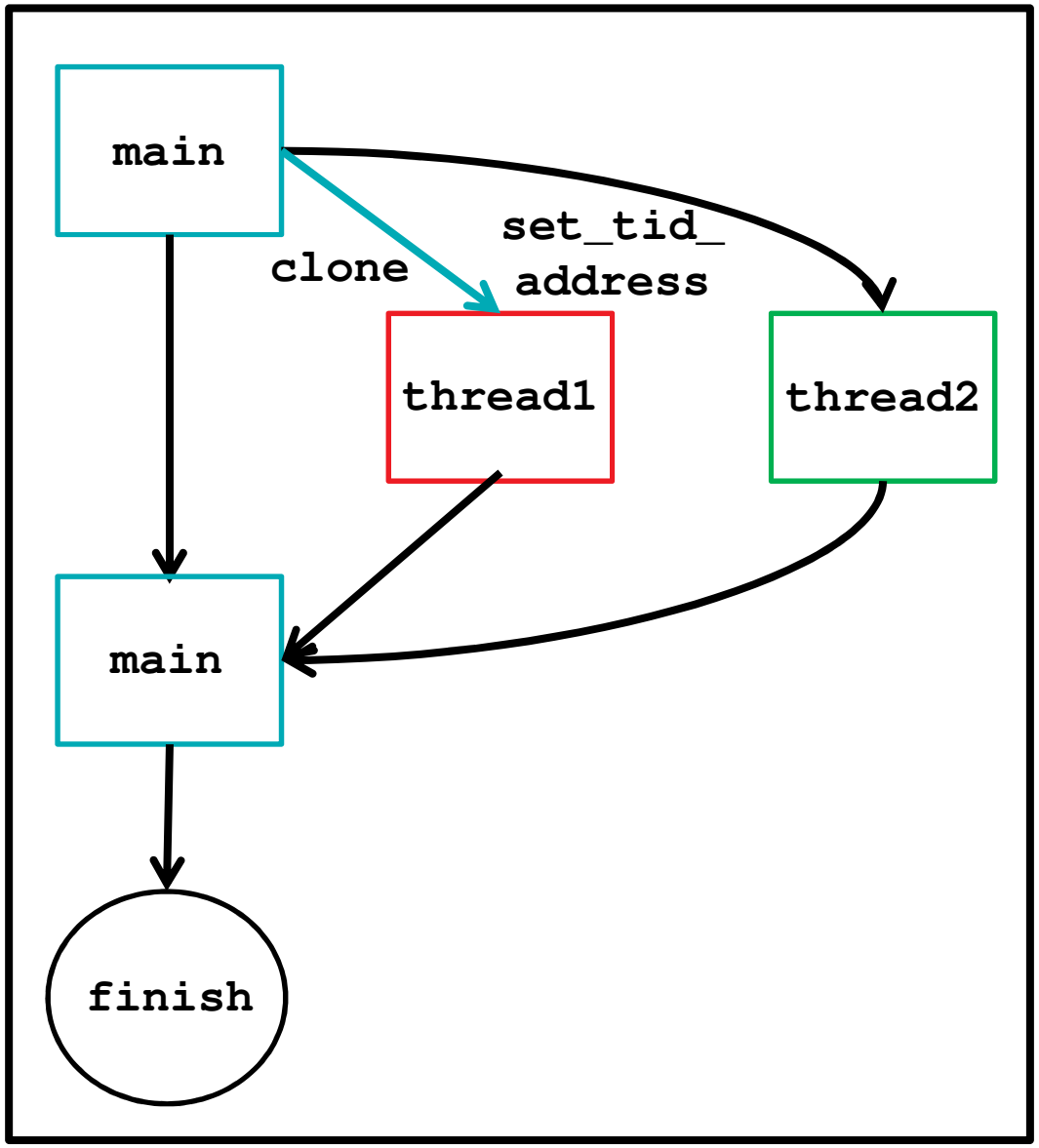

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



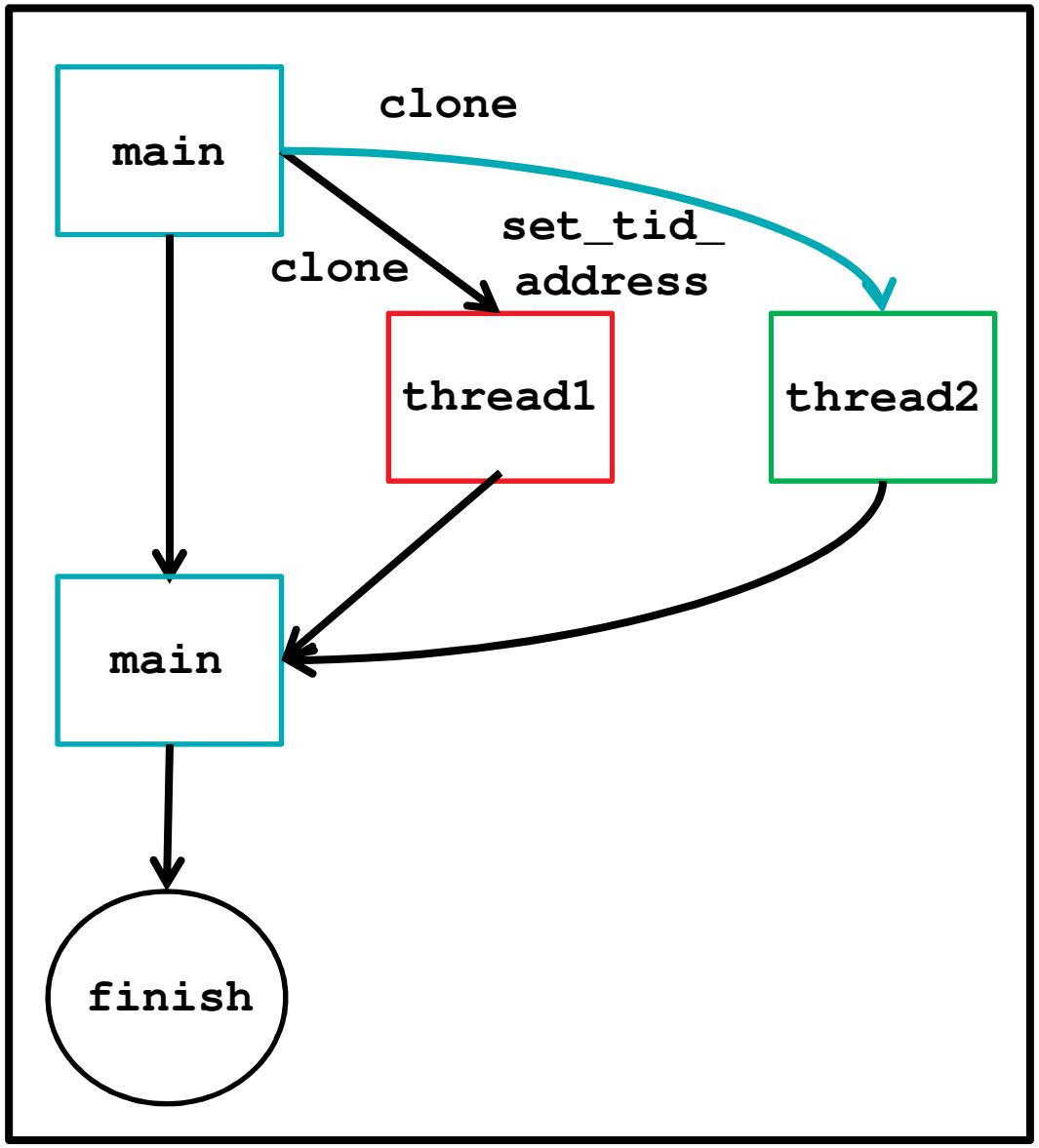

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



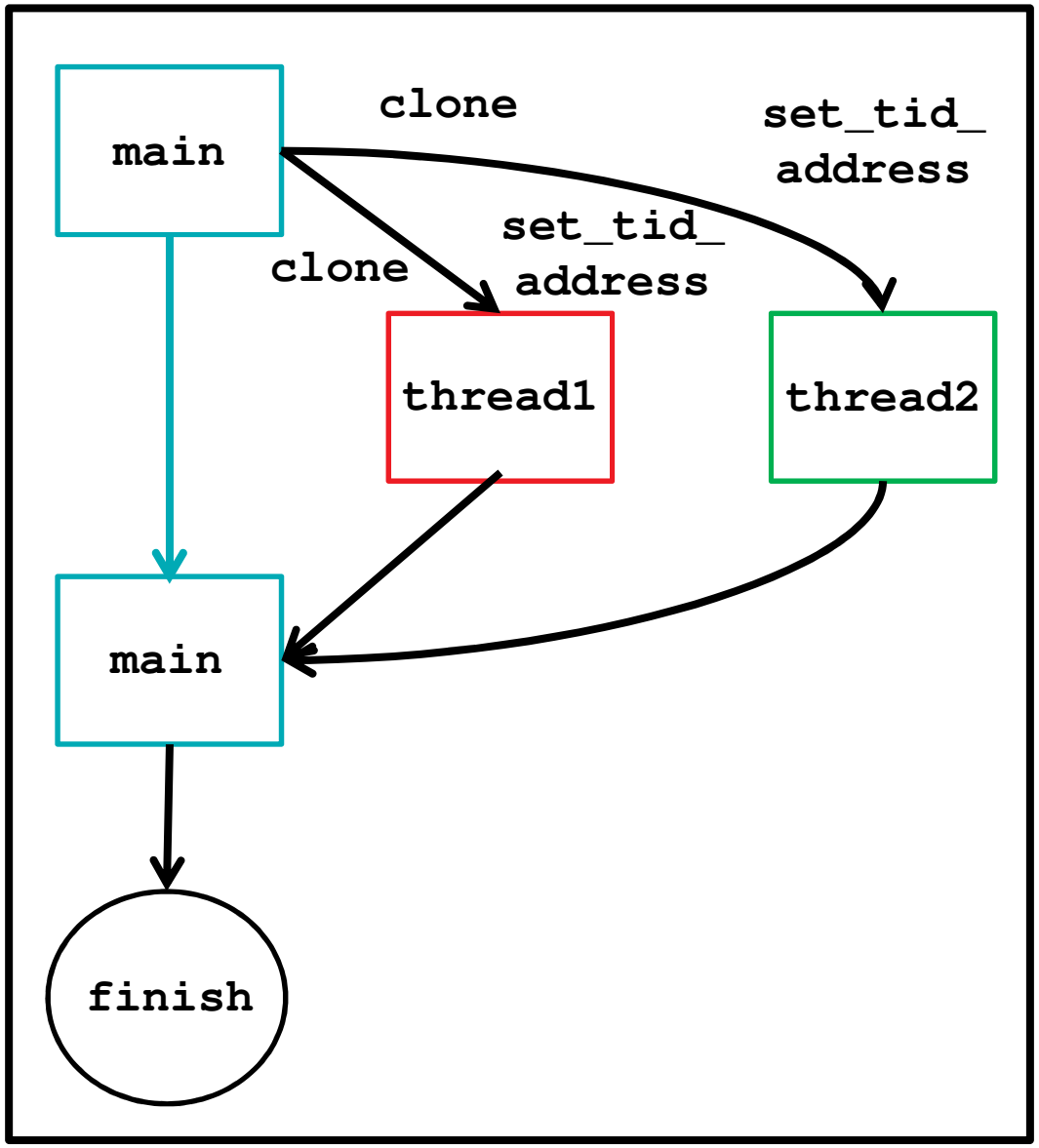

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



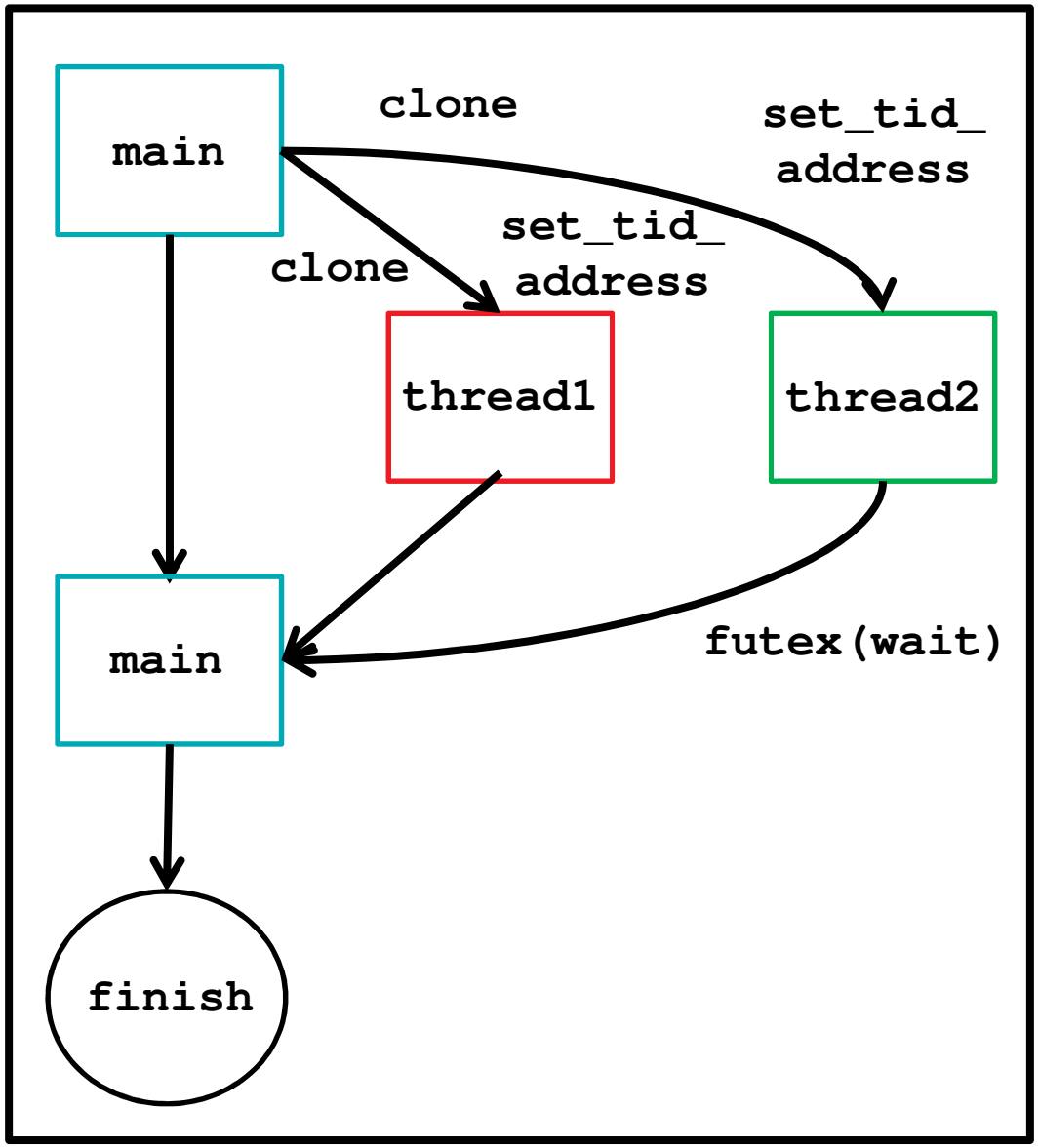

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
   pthread_mutex_lock(&lk);
   counter += 1;  counter += 1;
   pthread_mutex_unlock(&lk);
}
int main(void)
{
   pthread_create(lock_incr);
   pthread_create(lock_incr);
   pthread_join(…);
   pthread_join(…);
}
```

PUTTING PIECES TOGETHER



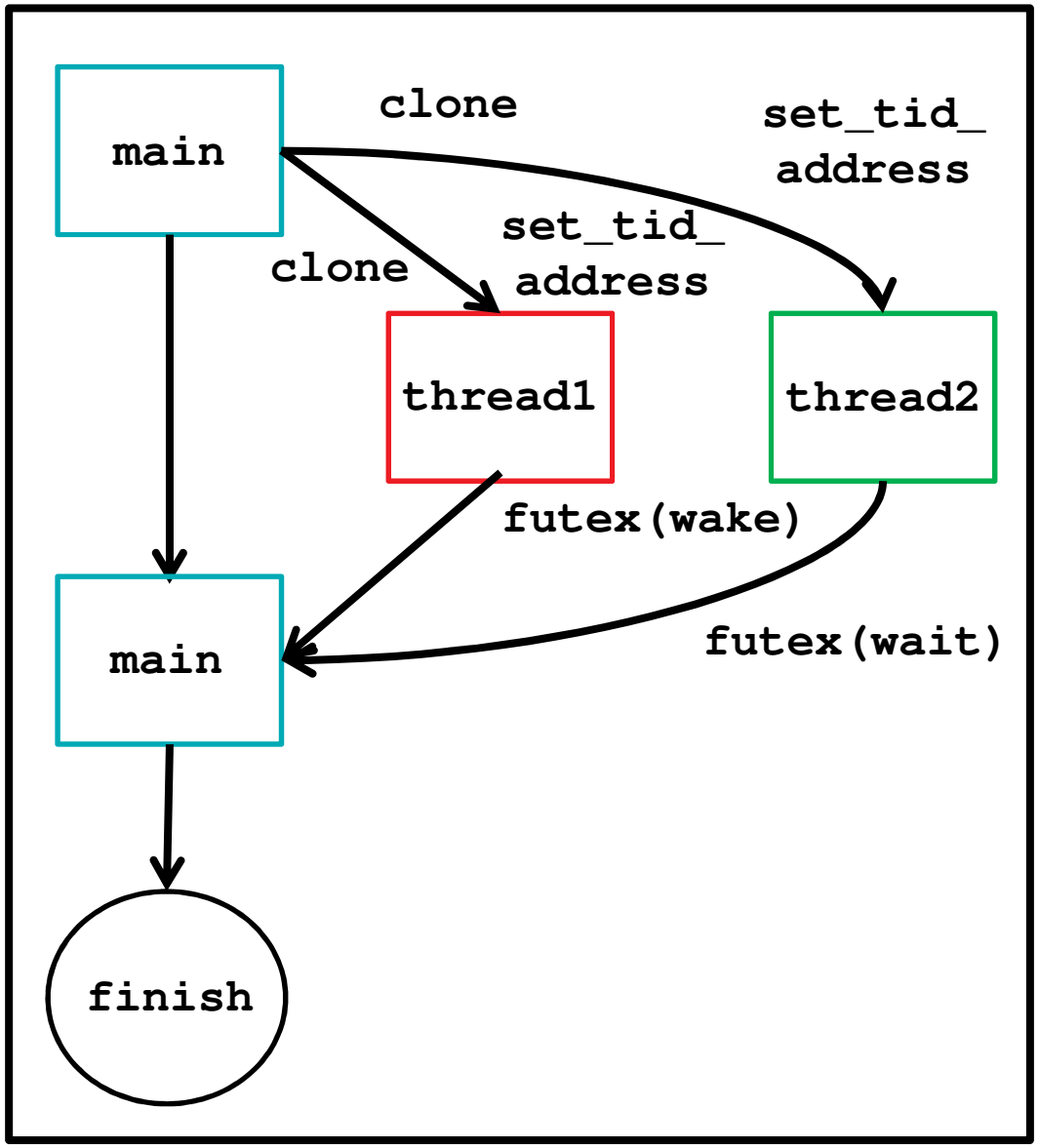

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1; counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



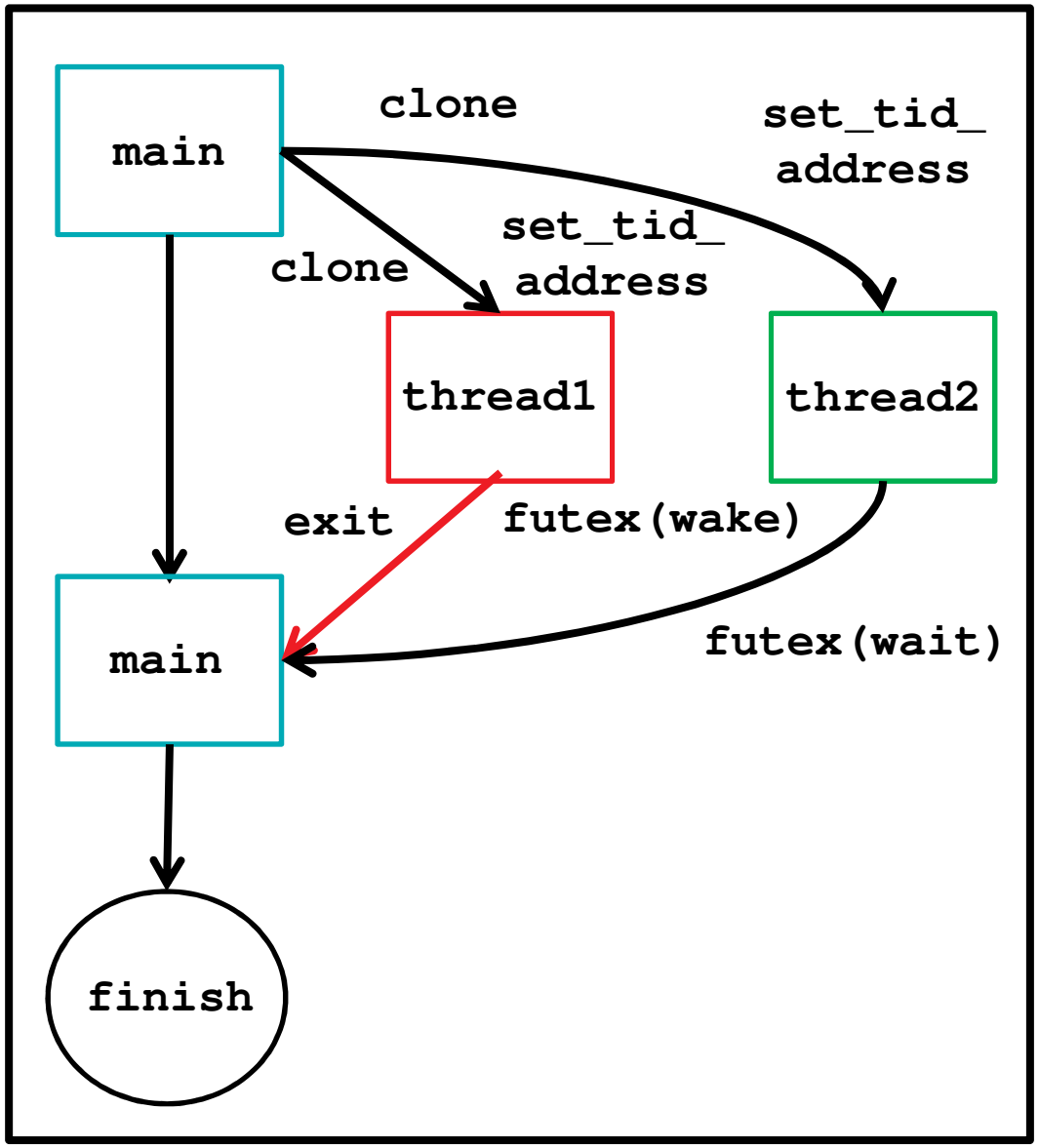

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
   pthread_mutex_lock(&lk);
   counter += 1;  counter += 1;
   pthread_mutex_unlock(&lk);
}
int main(void)
{
   pthread_create(lock_incr);
   pthread_create(lock_incr);
   pthread_join(…);
   pthread_join(…);
}
```

PUTTING PIECES TOGETHER



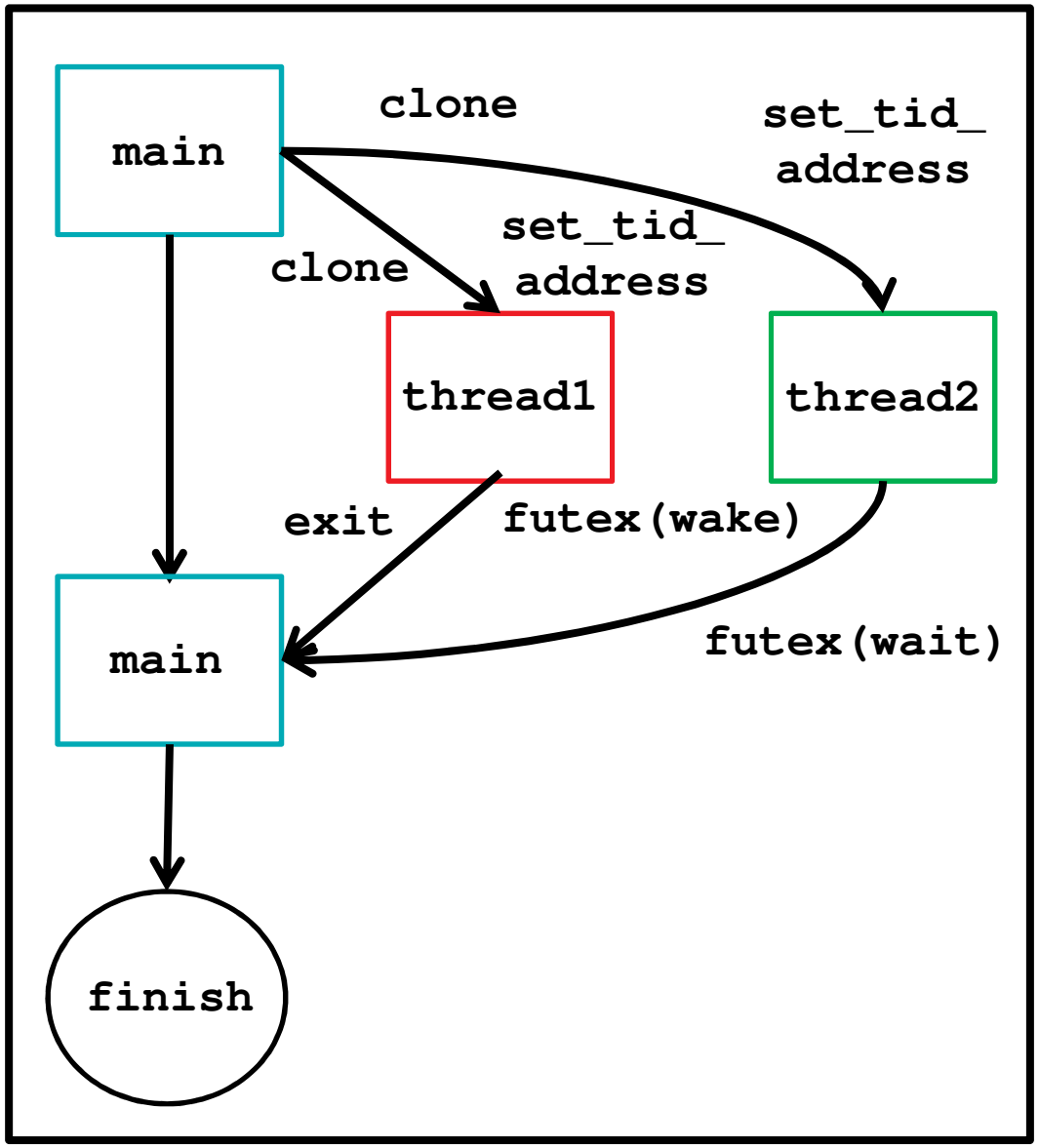

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
   pthread_mutex_lock(&lk);
   counter += 1;  counter += 1;
   pthread_mutex_unlock(&lk);
}
int main(void)
{
   pthread_create(lock_incr);
   pthread_create(lock_incr);
   pthread_join(…);
   pthread_join(…);
}
```

PUTTING PIECES TOGETHER



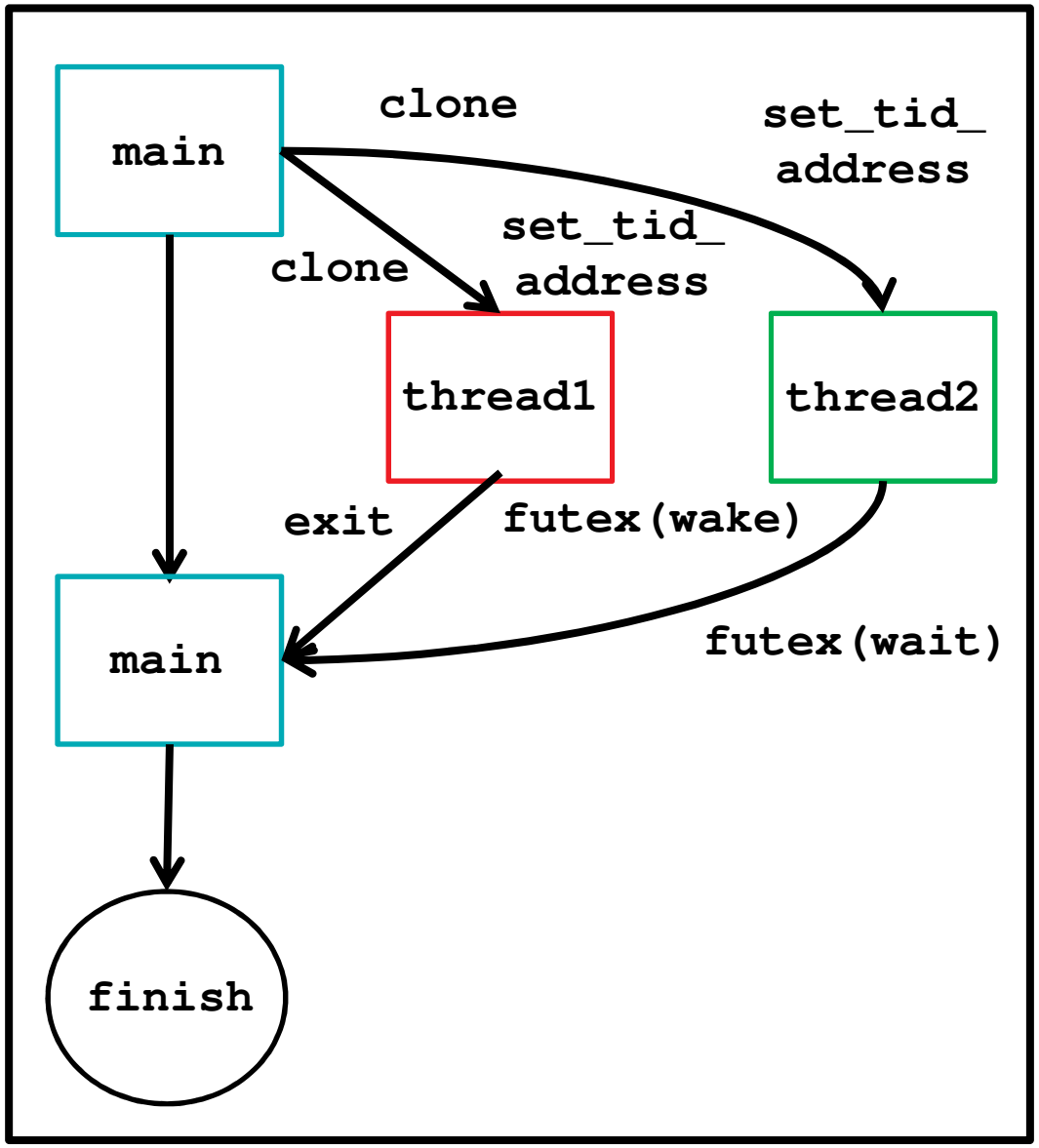

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1; counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



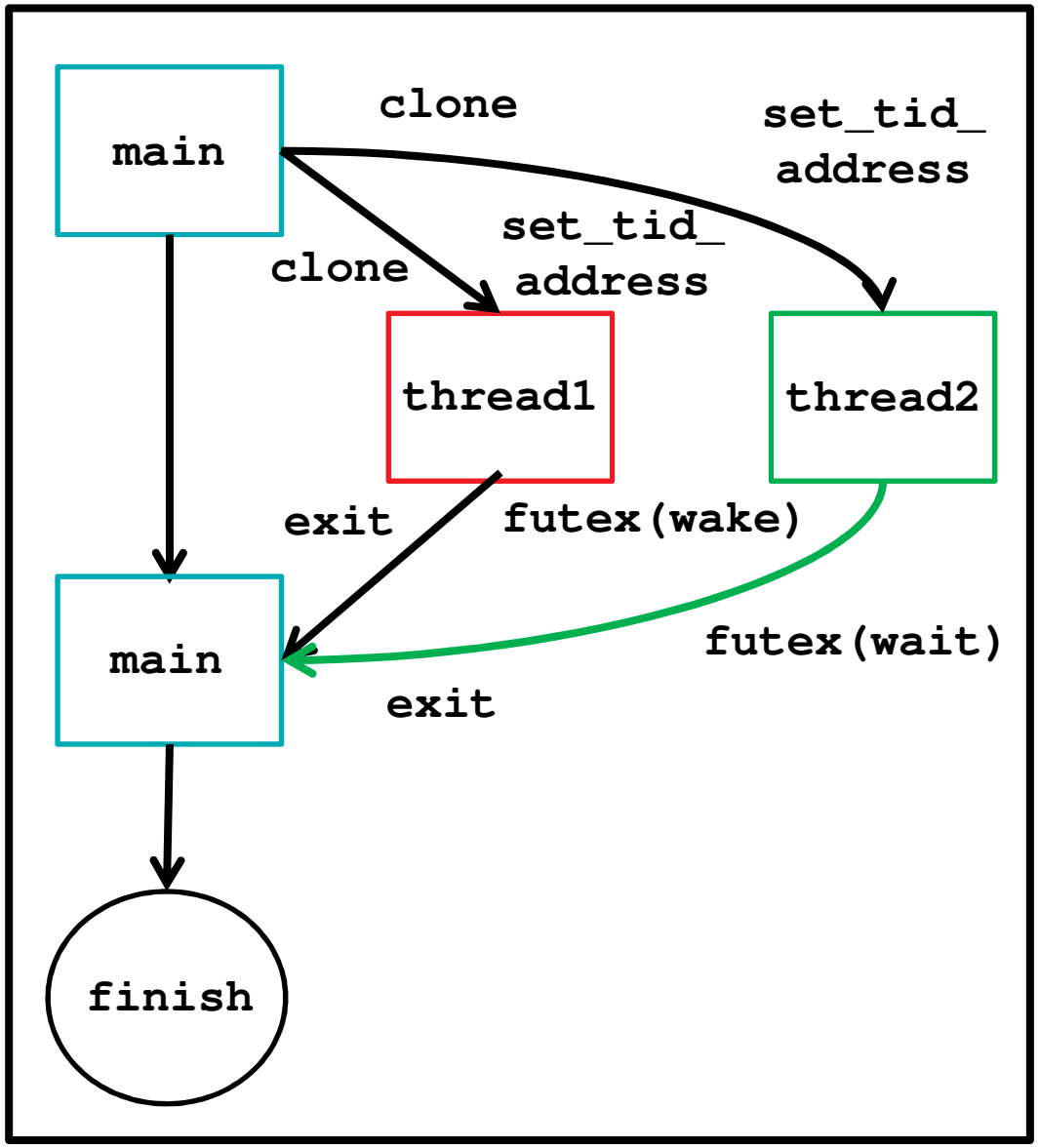

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1; counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER



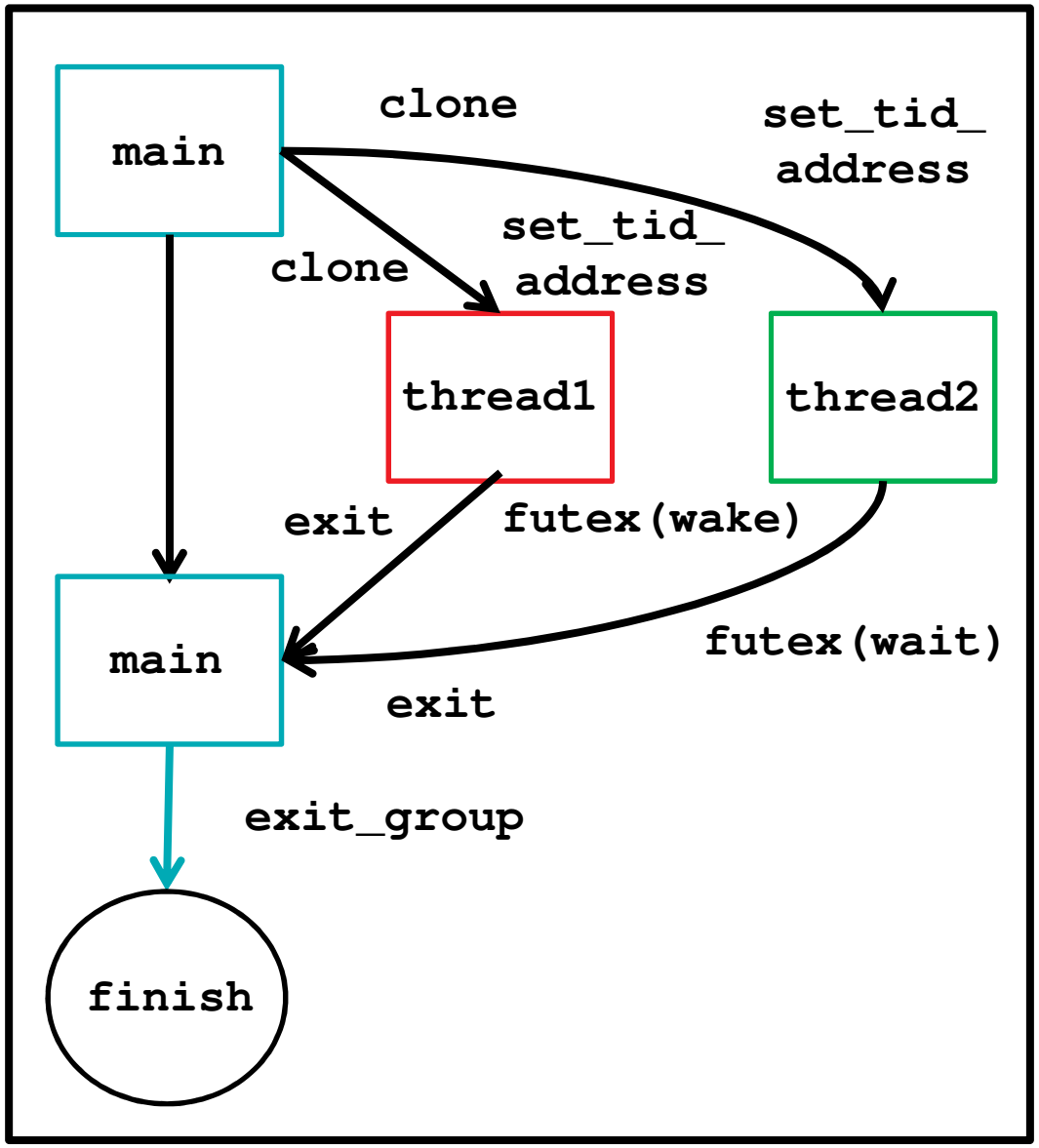

```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER




```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```

PUTTING PIECES TOGETHER




```
int counter = 0;
pthread_mutex_t lk;
void * lock_incr(void* tid)
{
    pthread_mutex_lock(&lk);
    counter += 1;  counter += 1;
    pthread_mutex_unlock(&lk);
}
int main(void)
{
    pthread_create(lock_incr);
    pthread_create(lock_incr);
    pthread_join(…);
    pthread_join(…);
}
```