Homework (Code)

① main-race.c

"balance" variable is accessed in main thread and in worker function which is run via a thread: Pthread_create uses that. Leads to data race.

Helgrind tool nicely tells the code lines with unprotected access.

② - Removing one of the two unprotected accesses, resolves the race condition

- Just protecting (lock around) one of the two places doesn't resolve the issue because other thread doesn't get blocked due to no lock around update.

- Now locks around both the places resolves the issue completely.

Note: "balance++" if we had atomic addition then no race condition without lock too.

③ Deadlock

Two threads stuck on acquiring resource that the other thread is holding and vice-versa

m1, m2 in this program due to order difference the issue arises, eg:- thread 0 with m1, thread 1 with m2. thread0 wants m2 and thread 1 wants m1.

Fixing the order of lock resolves the issue.

⑤ Global lock wrapped around the worker thread fixes the issue by blocking the other thread, internal m1, m2 order doesn't matter.

Helgrind reports the same error, because it can't figure out that it occurence can't happen any longer.

⑥ Code is inefficient because its busy waiting wasting CPU cycles.

⑦ Helgrind reports error because "done" variable is not synchronized, protected via lock for update.

(8) This code is both correct and performs better.

Perform - Signalling ensures parent can sleep and not waste CPU cycles
Wakes up when child signals that it is done.

Correct - "done" updates are protected via mutex.