# Troubleshooting

## Detecting pinning

The current prototype has rudimentary support for reporting when a thread is pinned, say when a virtual thread parks while owning a monitor or with a native frame on the stack. This reporting is enabled by running with the system property *jdk.tracePinnedThreads* specified to the java launcher.

Running with -Djdk.tracePinnedThreads (or -Djdk.tracePinnedThreads=full) will print a complete stack trace of a virtual thread when parking pins its carrier thread. The reason, be it native frames, synchronized frames, or frames with synchronized blocks will be highlighted in the output.

Running with -Djdk.tracePinnedThreads=short will print a "short" stack trace that includes just the frames with problematic code.

The reporting is filtered to avoid the printing the same stack trace repeatedly.

## Thread dumps

The traditional thread dump (Ctrl-\, jstack, jcmd Thread.print) does not include virtual threads. If the thread dump is taken when a virtual thread is mounted then its stack will be shown in the stack trace of its carrier thread. No information about unmounted virtual threads will be included.

The current prototype has an alternative thread dump that includes virtual threads started with a thread Executor (*Executors.newVirtualThreadExector* or *Executors.newThreadExector*). The thread dump is generated in plain text or JSON format. The thread dump can be generated programmatically with com.sun.management.ThreadMXBean. The jcmd tool can be used to generate the thread dump to a file:

```
jcmd <pid> JavaThread.dump /d/threads.txt
jcmd <pid> JavaThread.dump -format=json -overwrite /d/threads.json
```

The JSON output has two arrays. There is one array for the threads that includes the identifier, name, and stack trace of each thread. The second array has the thread executors and the identifiers of all threads started by the executor.