Subshells: Bringing Multithreading to Jupyter Kernels

Ian Thomas, QuantStack

JupyterCon 2025

The problem we are trying to solve

- Cannot interact with a kernel whilst it is busy executing code
- Would like to:
    - Inspect kernel state during long calculation
    - Visualise intermediate results
    - Execute arbitrary code in parallel

What is a kernel subshell?

- A separate thread of execution which can run code independently of the main shell and shares the same memory space
- Multithreading for Jupyter kernels

History of subshells proposal 1

- Changes to the Jupyter protocol occur via Jupyter Enhancement Proposals
- PR proposing subshells with discussion
- Accepted Jupyter Enhancement Proposal 91

History of subshells proposal 2

Two potential implementations:

- **Kernel subshells**
  - Subshell uniquely identified by kernel ID and new subshell ID
  - Kernel has single socket to receive messages for main shell and subshells
  - For `n` subshells we need `n+1` new threads, no new external sockets
- **Dependent kernels**
  - Subshell uniquely identified by a new kernel ID
  - Each subshell appears as a unique kernel
  - For `n` subshells we need `n` new threads and `n` new external sockets

History of subshells proposal 3

Timeline:

- December 2022: JEP proposed by David Brochart
- January 2024: JEP taken over by Ian Thomas
- February 2024: Proof of concepts for both approaches
- March 2024: Agreed kernel subshells not dependent kernels approach
- June 2024: Initial `ipykernel` implementation PR 1249, merged October 2024
- September 2024: JEP 91 approved
- May 2025: Second `ipykernel` implementation PR 1396, merged June
- October 2025: `ipykernel` 7.0.0 released. Currently on 7.1.0

What is a kernel (recap)

- Process such as `python` running on a computer somewhere
- Channels: `shell`, `control`, `stdin`, `iopub`
- Each channel can send and receive messages via sockets
- Other end of sockets connected to e.g. Jupyter server/lab
- Multiple threads for channels and sockets

Example communicating with a kernel

mermaid diagram showing control and shell threads, with kernel info request and execute request messages.

How subshells work 1

- Subshell is a separate thread of execution in a kernel process
- Identified by string `subshell_id`
  - Must be unique within kernel, UUID ensures globally unique
- Created and deleted via control messages such as `create_subshell_request` which returns new `subshell_id`
- Shell request messages support optional `subshell_id`
- Kernel advertises its support for subshells via `kernel_info_request` `supported_features`

How subshells work 2

Major difference is new shell channel thread

- Previously the shell execution thread also handled shell messages
  - Queue messages, execute sequentially
- Now shell channel thread handles shell channel messages
  - Read incoming message header, extracts `subshell_id`, sends message to correct main shell or subshell
  - Implementation detail: uses ZMQ inproc pair sockets, essentially shared memory but with a ZMQ socket API
  - Reply messages also sent via shell channel thread

Example communicating with a kernel with subshells

mermaid diagram similar to that in https://github.com/ipython/ipykernel/pull/1249

What you need to try out subshells

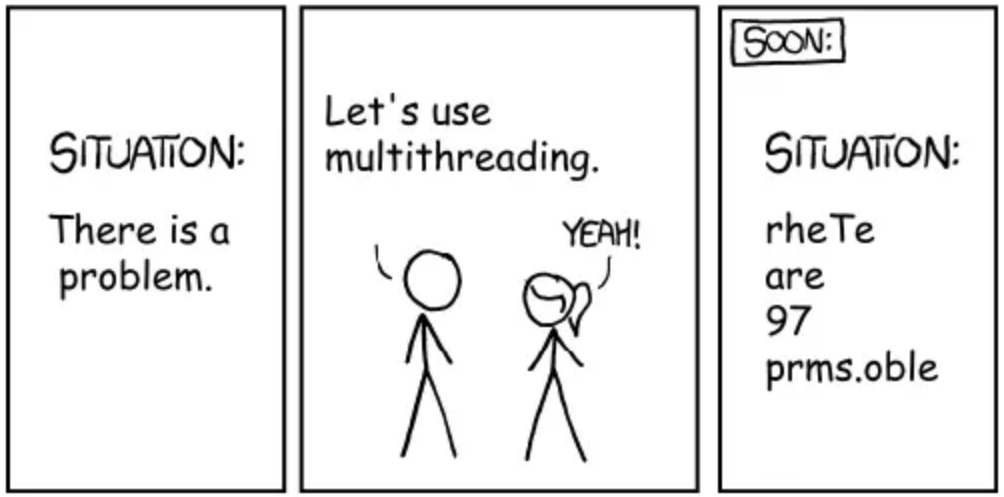
- `ipykernel >= 7.0.0`
  - `ipykernel` is the reference implementation.
  - Later releases fix problems, mostly caused by new shell channel thread rather than subshells themselves
- `jupyterlab >= 4.4.0`
- For `ipywidgets` support need `jupyterlab >= 4.4.0`
- For `%matplotlib ipympl` support need `ipympl >= 0.9.8`

Live demos

...

⚠️ Warning! ⚠️

Only use subshells if you understand multithreading!



Future problems that still need to be solved

- Execution count - separate for each subshell?
- History - separate or combined?
- Debugging from subshell threads - not yet supported
- Busy/idle status works on a per-subshell basis.
    - Reconsider whole approach to busy/idle?

How to implement subshells in a language kernel 1

For kernel developers

- Separate shell channel thread
  - Socket pair for communications with main shell
  - Shell channel thread sends messages as soon as possible
  - Main shell thread should execute messages sequentially
- Main shell thread should by default be the main thread, so it can handle signals and GUI event loops

How to implement subshells in a language kernel 2

- `kernel_info_request` `supported_features` must include "kernel subshells"
- Three new control channel messages to `create`, `list` and `delete` subshells
- Actual subshell
  - Create, stop, and delete the thread
  - Socket pair to communicate with shell channel thread
  - Execute one shell message at a time
- Care needed with data that is cached per kernel which may not be correct now
  - Example is parent message header which now need to be one per subshell
- `%subshell` magic optional but recommended
- Will need thread lock/mutex

Where we are at now

- Can use `ipykernel 7` in `jupyterlab` now for user code, as in demo
- Please report any problems
- `ipywidgets` in `jupyterlab` supports comms over subshells, can be changed in settings
- `ipywidgets`-derived extensions should support it automatically too
- No support yet in any other kernels (as far as I am aware) or `jupyterlite`

Where we are going

- Expecting some extensions to be modified to use subshells when available, such as variable explorer
- Support in other `jupyterlab` kernels?
- Support in `xeus` kernels?
  - Not restricted to python
- `jupyterlite` kernels are single threaded, research project to look at `pthreads`

Resources

- Jupyter Enhancement Proposal 91
  - Documentation
  - PR with discussion
- `ipykernel`
  - Original implementation on top of `anyio`
  - Current implementation of top of `tornado` / `asyncio`
- `jupyterlab`
  - Support kernel subshells
  - Comms over subshells
- `ipympl`
  - Use thread lock to support comms via subshells

Thanks