Possible race condition leading to the main loop dying? #374

Closed oakkitten opened this issue on Apr 11 · 16 comments · Fixed by #377

Labels bug

oakkitten commented on Apr 11

I just might be wrong because if this indeed a race condition it should be breaking more things. Anyway, I got this exception (line numbers might be wrong due to debug statements):

```
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/.../lib/python3.8/site-packages/aqt/mediasrv.py", line 89, in run
    self.server.run()
  File "/.../lib/python3.8/site-packages/waitress/server.py", line 323, in run
    self.asyncore.loop(
  File "/.../lib/python3.8/site-packages/waitress/wasyncore.py", line 285, in loop
    poll_fun(timeout, map)
  File "/.../lib/python3.8/site-packages/waitress/wasyncore.py", line 211, in poll
    r, w, e = select.select(r, w, e, timeout)
OSError: [Errno 9] Bad file descriptor
```

This error was extremely rare but since I was getting it while running tests I could just run a lot of them until one failed, which I did, and I think the problem is a follows.

1. First, thread `Thread-1` that the app I'm testing is launching, one that runs waitress server, assembles the descriptor lists for `select`:

waitress/src/waitress/wasyncore.py
Lines 154 to 166 in `603d2c1`

```
154     r = []
155     w = []
156     e = []
157     for fd, obj in list(map.items()):  # list() call FBO py3
158         is_r = obj.readable()
159         is_w = obj.writable()
```

```
        if is_r:
            r.append(fd)
        # accepting sockets should not be writable
        if is_w and not obj.accepting:
            w.append(fd)
        if is_r or is_w:
```

2. Then, thread `waitress-0` deletes one of the channels, in my case it was `<waitress.channel.HTTPChannel 127.0.0.1:54044 at 0x7f10ec052400>`, and immediately closes the socket:

   waitress/src/waitress/wasyncore.py
   Lines 460 to 470 in `603d2c1`

```
    def close(self):
        self.connected = False
        self.accepting = False
        self.connecting = False
        self.del_channel()
        if self.socket is not None:
            try:
                self.socket.close()
            except OSError as why:
                if why.args[0] not in (ENOTCONN, EBADF):
                    raise
```

   Stack of `waitiress-0` at the moment:

```
  File "/usr/lib/python3.8/threading.py", line 890, in _bootstrap
    self._bootstrap_inner()
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.8/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/.../lib/python3.8/site-packages/waitress/task.py", line 84, in handler_thread
    task.service()
  File "/.../lib/python3.8/site-packages/waitress/channel.py", line 426, in service
    task.service()
  File "/.../lib/python3.8/site-packages/waitress/task.py", line 168, in service
    self.execute()
  File "/.../lib/python3.8/site-packages/waitress/task.py", line 451, in execute
    self.channel.write_soon(app_iter)
  File "/.../lib/python3.8/site-packages/waitress/channel.py", line 377, in write_soon
    (flushed, exception) = self._flush_exception(self._flush_some)
  File "/.../lib/python3.8/site-packages/waitress/channel.py", line 132, in _flush_exception
    return (flush(), False)
  File "/.../lib/python3.8/site-packages/waitress/channel.py", line 270, in _flush_some
    num_sent = self.send(chunk)
  File "/.../lib/python3.8/site-packages/waitress/wasyncore.py", line 479, in send
    self.handle_close()
  File "/.../lib/python3.8/site-packages/waitress/channel.py", line 317, in handle_close
    wasyncore.dispatcher.close(self)
    File "/.../lib/python3.8/site-packages/waitress/wasyncore.py", line 519, in close
      traceback.print_stack()
```

3. Then, thread `Thread-1` is trying to see if the file descriptor of the socked closed above is writable, which leads to the the exception above:

   waitress/src/waitress/wasyncore.py
   Lines 171 to 177 in `603d2c1`

   ```
       try:
           r, w, e = select.select(r, w, e, timeout)
       except OSError as err:
           if err.args[0] != EINTR:
               raise
           else:
               return
   ```

Python 3.8.10, waitress 2.1.1, Ubuntu 20.04 LTS focal @ WSL2

👍 2

bertjwregeer added the bug label on Apr 21

bertjwregeer mentioned this issue on Apr 21

Bugfix: Don't close socket in the WSGI thread, delegate it back to the main thread! #377
Merged

bertjwregeer commented on Apr 21 Member

@oakkitten could you try this patch: #377 and see if it stops the main thread from dying? I have been unable to come up with a good way to test/validate this fixes the issue.

It's not the best solution, but short of rewriting `asyncore` it hopefully provides relief as we just try again.

oakkitten commented on Apr 22 Author

I'm not familiar with the codebase, but from what I've seen while trying to catch the issue—

Perhaps, instead of calling `del_channel`, which mutates `map`, and then closing the socket in a worker thread, it would be more simple to just flag the channel for closing? Like, you set a flag on a channel, such as `must_be_closed_in_the_main_loop`, or perhaps have a separate list of objects that must be closed (lists are said to be thread-safe), and then the main loop checks if anything should be closed before doing `select()`. This way the code would be a bit easier to reason about I think.

Also, you are doing

```
if fds != map.keys():
    fds = map.keys()
```

`dict.keys()` returns not a copy of keys, but a special `dict_keys` object that is a view of dictionary keys that changes along with the dictionary. So if read this right, `fds == map.keys()` will always be true.

Also, there's the issue with the modification of `map` itself. While it never blew up in my face during tests, it probably can, since when you modify a dictionary while iterating you get this:

```
>>> d = {x: x for x in range(5)}
>>> for x in d.keys():
...     print(x)
...     if x == 3:
...         del d[x]
...
0
1
2
3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size during iteration
```

I suppose even doing `list(map.keys())` is dangerous.

bertjwregeer commented on Apr 25 Member

Good point on the `dict.keys()` I forgot about that change in Py3.

We never iterate the map and delete from it though in the code change I made.

I am trying to avoid changing too much of asyncore as that was wholesale lifted from Py 3.5.

If you were to use `poll()` instead of `select()` I would imagine the problem would be masked because `poll()` will simply return that there is an error on the socket, whereas `select()` fails and raises `OSError`. So I was attempting to make `select()` act more like `poll()` in this situation.

This only really happens if the remote closes the connection before having read the full response, which is likely why it is not often seen in the wild, as most reverse proxies will have already read the full response.

The problem with attempting to add yet another list, is that there is no single object, the `map` is passed down from the `serve` function, and is passed around, but to add another list that would mean adding another variable that is passed around to the various functions as the `loop()` function does not currently have that.

Trying to set a flag inside the thread and trying to make sure that can be read throughout the `asyncore` functions so that it doesn't close the sockets was something I looked at, but it is difficult as there are various places where `asyncore` expects to be able to close the socket.

It's going to require more intrusive changes, which makes sense as `asyncore` was never really meant to be thread safe. This is a consequence of trying to send data faster without waking the main thread up.

bertjwregeer commented on Apr 25 Member

Using `list(map.keys())` should be fine. Since the only reason `select()` can fail is if it has a file descriptor that is no longer valid (and thus no longer in the map). Comparing the original map we used for `select()` against the new `map.keys()` should show that there is a difference, at that point we just start the `select()` call again with the new list of file descriptors.

Eventually a call to `select()` will succeed.

I am going to see if I can write a test that causes the race.

You mentioned you are seeing this in tests, are you running waitress in a thread or a separate process?

oakkitten commented on Apr 25 Author

You don't iterate the map and delete from it in the code change you made, but I think you might be doing that in the already existing code?

- The main thread says `list(map.items())` .
- `close()` calls `del_channel()` , and `del_channel()` says `del map[fd]` . This can (and will) happen during the above `list()` call.

So the question is, is `list()` doing or calling anything that touches GIL in some kind of a guaranteed way. Well, I don't know, really. I asked on #python if it will call some magic thread-safe method for converting keys to a list and no-one was aware of such a method. And I don't see the documentation mention anything like that. Even if this is safe and indeed not an implementation detail, I know I wouldn't want to rely on this behavior, anyway. This is too complicated!

I am testing an addon for a Qt app that uses waitress internally. It launches it in a thread. This is a patch I made to fix the issue in a dumb way, which should be ok for tests. If you are willing, you can try grabbing the commit before that and running the tests (see tox.ini for instructions; the tests are slow and require a lot of dependencies so it won't be fun). The problem is rare, but you can force it by inserting some kind of a delay before the `select` call, then it manifests right away.

oakkitten mentioned this issue on May 24

closing webview windows causes crashes on some machines ankitects/anki#1879

Closed

bertjwregeer commented on May 24 — Member

I updated #377 (comment) with the new changes. Give this a shot please.

bertjwregeer commented on May 24 — Member

It removes any races, only the main thread can close the socket...

mmerickel closed this as completed in #377 on May 24

oakkitten commented on May 25 — Author

I pulled `4f6789b` and 4800 tests and 108 minutes later there were no crash. Thanks! 🎉

wesleybl added a commit to collective/collective.cover that referenced this issue on Jun 2

`Pinn waitress = 2.1.2 in Python 3.7 and 3.8 to fix robot tests` … ✓ ab681a9

lamby commented on Jun 21

I spot that the description for CVE-2022-31015 mentions that this affects "versions 2.1.0 and 2.1.1". However, a quick glance at the code suggests that this might be because it affects the `wasyncore` module which was vendored in version 1.2. Does this issue, then, affect these older versions as well? Thanks. :)

mmerickel commented on Jun 21 — Member

No. It’s not a bug in wasyncore but rather waitress began trying to invoke wasyncore methods like close() from other threads that caused the issue.

lamby commented on Jun 21

Getcha. However, the fix essentially requires the vendored version, no? Otherwise the `do_close` thing can't be passed around? :)

mmerickel commented on Jun 21 Member

It enables a performance optimization where waitress can write to the socket safely from a thread.

mmerickel reopened this on Jun 21

mmerickel closed this as completed on Jun 21

bertjwregeer commented on Jun 21 Member

@lamby the vendored version is always used, even on Python versions that have `asyncore` support built-in. Waitress does not attempt to use the non-vendored version and then fall back to the vendored version.

lamby commented on Jun 22

That makes sense. However, someone using a very very old version of waitress (prior to the module being vendored in, that is...) would be vulnerable to this issue?

mmerickel commented on Jun 22 Member

Again no because the bug was only due to a change in how we USED asyncore in the specified versions of waitress. Waitress used it differently and safely before then. And it does again after the cve fix. We documented the affected versions correctly in the cve.

lamby commented on Jun 22

Thanks, really appreciate it. :)

Assignees

No one assigned

Labels

bug

Projects

None yet

Milestone

No milestone

Development

Successfully merging a pull request may close this issue.

Bugfix: Don't close socket in the WSGI thread, delegate it back to the main thread!
Pylons/waitress

4 participants