Java Nio Debugging Note

Chen Luo chenluo.cn@gmail.com

October 8, 2020

1 Motivation

Recently, I'm learning java nio with some sample code. At first, I forget handle the case that SocketChannel.read return -1. The behavior becomes Selector.select() is always returning an event with SocketChannel.read() is -1. Then the program enters a endless loop.

It is mandatory to handle the above case: un-register SelectionKey or close the SocketChannel. It solves the problem. But I'd like to see what is happening without it.

It's not enough only debugging at Java side. When stepping into Selector.select(), EPoll.wait() is eventually a native method. And some other related method are relatively low-level.

I referenced the blog[2], many QAs which are not listed here and official manual[1] to debugging the jvm itself.

2 Preparation

2.1 Environment

I'm using Ubuntu 20.04.1 LTS. The OS dependent implementation won't be fully discussed.

2.2 Download source code

The OpenJDK uses Mercurial and the manual has a quick start guide. I'm not familiar with this vcs. I cloned the OpenJDK from `https://github.com/openjdk/jdk` instead. At the time I cloned, the version in master branch is 16-ea.

2.3 Build JDK

As the blog shows, it need to specify the debug level to slowdebug when building the jdk. During the following instructions, some additional packages maybe missed. It's easy to follow the log and install the missed packages.

```
bash ./configure --with-target-bits=64 --with-debug-level=slowdebug
--disable-warnings-as-errors --with-native-debug-symbols=internal
--with-boot-jdk=[PATH_TO_EXISTING_JDK]

make clean

make all
```

2.4 Import to VSC

I'm using Visual Studio Code to read the C code. After run the make all command, it produces a vsc workspace file. Just open it.

2.5 Using built JDK to run Java program

After 2.3, it would be a JDK in

```
[JDK_SRC_DIRECTORY]/build/linux-x86_64-server-slowdebug/jdk
```

3 Run Java program and debugging by gdb

3.1 Run Java program

Server

Here is the sample code. It starts a server and handle incoming connections.

```
public void testSocketNio() throws IOException {
        ServerSocketChannel channel = ServerSocketChannel.open();
        channel.configureBlocking(false);
        ServerSocket serverSocket = channel.socket();
        serverSocket.bind(new InetSocketAddress(10901));
        Selector selector = Selector.open();
        channel.register(selector, SelectionKey.OP_ACCEPT);
        try {

            while (true) {
                int eventNum = selector.select();
                if (eventNum == 0) {
                    continue;
                }
                logger.info("{} events come", eventNum);
                Set<SelectionKey> selectionKeys = selector.
                    selectedKeys();
                logger.info("{}", selectionKeys);
                Iterator<SelectionKey> iterator = selectionKeys.
                    iterator();
                while (iterator.hasNext()) {
                    SelectionKey key = iterator.next();
                    if (key.isAcceptable() && (key.readyOps() &
                        SelectionKey.OP_ACCEPT)
                            == SelectionKey.OP_ACCEPT) {
                        ServerSocketChannel serverSocketChannel =
                                (ServerSocketChannel) key.channel();

                        SocketChannel socketChannel =
                            serverSocketChannel.accept();
                        logger.info("OP_ACCEPT from {}", socketChannel
                            .socket().getPort());
                        socketChannel.configureBlocking(false);
                        SelectionKey selectionKey = socketChannel.
                            register(selector,
                                SelectionKey.OP_READ);

                    } else if (key.isReadable() &&
                            (key.readyOps() & SelectionKey.OP_READ) ==
```

```
                        SelectionKey.OP_READ) {
                    SocketChannel socketChannel = (SocketChannel)
                        key.channel();
                    logger.info("OP_READ from {}", socketChannel.
                        socket().getPort());
                    ByteBuffer buffer = ByteBuffer.allocate(16);

                    int readResult = socketChannel.read(buffer);
                    if (readResult == -1) {
                        socketChannel.socket().close();
                        socketChannel.close();
                        logger.warn("channel closed: {}",
                            socketChannel.socket().getPort());
                    } else {
                        buffer.flip();
                        if (buffer.hasRemaining()) {
                            logger.info("{}:read string:{}",
                                socketChannel.socket().getPort(),
                                    new String(buffer.array()).trim
                                        ());
                        }
                    }
                    buffer.clear();

                }
                iterator.remove();
            }
            selectionKeys.clear();
        }
    } catch (IOException e) {
        logger.error("", e);
    } finally {
        channel.close();
    }
```

Client

I used **nc** as the client to connect and send bytes.

3.2 Attach Java program to gdb

After running the server, it's possible to attach the process to gdb by **gdb -p** [**PID**]. **jps** is helpful to find the **PID**. It will prevent the server from running when connected.

3.3 Set breakpoint and some instructions

After connected to the java process, gdb can set breakpoints by **break** instruction. It accepts many kinds formats, like method name (which is available in auto completion) and line number. It's pretty enough to break the method first and then break at line level for me. Also, gdb supports save and reload breakpoints by **save** and **source** instructions.

3.4 Attach source of glibc

During debugging, taking Selector.select() as example, it eventually invokes a method epoll_wait. But it's in glibc rather JDK. To see the source code, following steps are necessary:

1. download glibc source code via apt source glibc
2. attach source code in gdb via directory `PATH_TO_GLIBC_SRC_CODE`

Here is an sample about breakpoint in glibc:

```
(gdb) source breakpoints_nio.pts
Breakpoint 4 at 0x7f242c0bc8b2: file /home/chen/Applications/
    openjdk_15_src/jdk/src/java.base/linux/native/libnio/ch/EPoll.c,
     line 84.
Breakpoint 5 at 0x7f242c0bc903: file /home/chen/Applications/
    openjdk_15_src/jdk/src/java.base/linux/native/libnio/ch/EPoll.c,
     line 87.
Breakpoint 6 at 0x7f2456260574: /home/chen/glibc-2.31/sysdeps/unix/
    sysv/linux/epoll_wait.c:30. (2 locations)
Breakpoint 7 at 0x7f242c0bd716: file /home/chen/Applications/
    openjdk_15_src/jdk/src/java.base/unix/native/libnio/ch/
    FileDispatcherImpl.c, line 299.

(gdb) list epoll_wait.c:30
```

```
26     int
27     epoll_wait (int epfd, struct epoll_event *events, int
   maxevents, int timeout)
28     {
29     #ifdef __NR_epoll_wait
30       return SYSCALL_CANCEL (epoll_wait, epfd, events, maxevents
   , timeout);
31     #else
32       return epoll_pwait (epfd, events, maxevents, timeout, NULL
   );
33     #endif
34     }
```

4 TODO

4.1 What's SYSCALL_CANCEL

It invokes SYSCALL_CANCEL at last. But the source code is out of glibc again. Next step is to debug into this line.

4.2 How does event/SelectionKey updated

Event Type: address + offset
FileDescriptor: Unsafe.get

References

[1] OpenJDK. Openjdk official.

[2] Alexey Pirogov. https://medium.com/@pirogov.alexey/gdb-debug-native-part-of-java-application-c-c-libraries-and-jdk-6593af3b4f3f.