Proposal:
Creating a independent signal thread for trap handler

Current signal/trap handling

<table>
<thead>
<tr>
<th>sighandler</th>
<th>timer thread</th>
<th>main thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recv signal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wakeup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interrupt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check signal and invoke trap</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Signal/trap handling with signal thread

[signal thread]

- Recv signal
- Wakeup

[sighandler]

- Recv signal

[timer thread]

- Wakeup signal thread directly
- Create
- Check signal
- Check signal and invoke trap

[main thread]

- Ruby thread
- Create
- Check signal
- Check signal and invoke trap

[signal thread]

- Ruby thread
- Check signal
- Check signal and invoke trap
- Invoked an exception if trap handler raise its exception such as Interrupt error

Advantage:
1. Signal thread is independent on main thread, this means that you can use thread synchronization between trap handler and main thread. In other words, you can run any program in trap handler.
2. Simplify a path from sighandler to trap invocation thread (after creation of a signal thread)
3. Doesn’t need a difficult implementation (modify is limited).

Disadvantage:
1. There is a small compatibility issue because "Thread.current" on a trap handler is not a main thread.

Discussion:
1. Create signal thread at first like timer thread is high cost. Without `trap`, we don’t need a signal thread any more.
2. In signal handler and timer thread, we can’t make a signal thread because creating “Ruby thread” (== signal thread) needs GVL. So the process path from timer thread to main thread is remained.