Using Expect with Coroutines

Colin Macleod

colin.macleod@mailbox.org

CGM on Tcl Wiki

Background

- I work in a financial data processing company, where production machines are tightly locked down because they are processing sensitive and confidential data. Logging in requires a special procedure and then you get a restricted shell to enforce read-only operations.
- Increasingly, processing is distributed over groups of machines, each running multiple instances of various server processes, with each processes logging to files which roll periodically to new files.
- When a problem occurs, we need to quickly find and study the relevant log to debug the problem.

Expect

- Expect is a well-known tool based on Tcl, which is very good for scripting interactions with remote systems. So we can use it to automate logging in to multiple production machines, finding the set of relevant logs, and searching them for some identifier or relevant pattern.
- But we have many machines to check, some of which might respond slowly or not at all. So we want to check all the machines in parallel, not one-by-one.

Expecting in parallel

- Expect supports handling multiple connections in parallel through the expect_background command.
- But using this forces us to write each step of processing as a handler for the event which triggers it. So now we have the well-known problem of “inversion of control”.
- In particular, maintaining state for each connection (e.g. the list of relevant files found on that machine) is awkward. The event handlers only know which connection they were called for, so typically we need to store state in global arrays indexed by connection (spawn_id in Expect).

Coroutines

- Coroutines can provide a helpful alternative to inversion of control for event-driven code.
- So using them together with Expect seems like a natural development. But I could not find any record of previous work in this area. Please tell me if did not look hard enough.
- However Expect is a flow-control command with many options, this makes it difficult to combine with coroutines in a fully general way.
- Luckily I did not need a fully general solution, so was able to write something simple but which still covers a number of use-cases.

Example of code run per-connection

```
exp_send "ls -1 --color=never $::filepat\n"

   set files {}

   co_expect $spawn_id op1> {\n([^\r]+)\r}
   while {[yield]} {
      set file $::expect_out(1,string)
      lappend files $file
   }

   foreach file $files {
      exp_send "gzgrep -i -n '$::target' $file | cut -c-1000\n"

      co_expect $spawn_id op1> {\n(\d+):([^\r]+)\r}
      while {[yield]} {
         set line $::expect_out(1,string)
         set text $::expect_out(2,string)
         add_match $machine $file $line $text
      }
   }
```

co_expect: Expect a set of patterns, resuming the current coroutine when one matches

```
proc co_expect {sid args} {

    set expect_args [list -i $sid]
    set pos -1
    foreach pattern $args {
        lappend expect_args -re $pattern [list [info coroutine] [incr pos]]
    }

    after idle expect_background $expect_args
}
```