pipethread

It is what it does

ROY KEENE

Tcl 2016
Introduction to POSIX Shell Pipes

Example 1

```bash
#!/usr/bin/env sh
cat /etc/passwd | wc -l
```

Example 2

```bash
( cat /etc/passwd ) | ( wc -l )
```
Introduction to POSIX Shell Pipes

Example 1

```bash
#!/usr/bin/env sh
cat /etc/passwd; echo test | wc -l
```

Example 2

```bash
(cat /etc/passwd; echo test ) | ( wc -l )
```
More Pipes in the POSIX Shell

Script:

```bash
#!/usr/bin/env sh

now="`date +%s`"
read line
now=$now
line=$line
```

Output:

```
1479277948
1479277948
1479277948
```

Tcl 2016
pipethread

It is what it does

Implements the same idioms from the POSIX shell in pure Tcl -- and more.
```tcl
#!/usr/bin/env tclsh

set now [clock seconds]
pipethread::pipe {
    puts $outchan $now
} | {
    gets $inchan line; puts $now; puts $line
}
```

Output:

```
1479277948
1479277948
```
Comparison

POSIX Shell:

```bash
now="`date +%s`"; ( echo $now ) | ( read line; echo $now; echo $line )
```

pipethread:

```tcl
set now [clock seconds]; pipethread::pipe { puts $outchan $now } | { gets $inchan line; puts $now; puts $line }
```

What's going on over on the pipethread side?

- What are `inchan` and `outchan`?
- Are there threads involved here?
POSIX Shell: Multiprocessing

( echo hello ) | ( read salutation; echo $salutation )

^- process #1  ^^- process #2
\-- pipe connecting stdout from
    process #1 to stdin for process #2
pipethread::pipe {
    # Thread #1
    puts $outchan hello
} | {
    # <-- Pipe connecting $outchan
    ; # from thread #1 to $inchan for thread #2

    # Thread #2
    gets $inchan salutation; puts $salutation
}
pipethread: inchan and outchan

POSIX shell has stdin and stdout, pipethread has inchan and outchan.

Simple!
Can you hear me now?

POSIX Shell:

```bash
now="`date +%s`"; ( echo $now ) | ( read line; echo $now; echo $line )
```

pipethread:

```tcl
set now [clock seconds]; pipethread::pipe { puts $outchan $now } | { gets $inchan line; puts $now; puts $line }
```
Closure on this subject

The `now` variable is part of a closure. Kind of.
What now?

We can combine these ideas to form more complex structures.
POSIX Shell

Script:

```bash
#!/usr/bin/env bash

number="-1"
cat /etc/passwd | while IFS=' ' read -r line; do
  number=`expr $number + 1`
  echo "$number:$line"
done | tac
```

Output:

```
32:polkitd:x:87:87:PolicyKit ...
31:pulse:x:65:65:User for Pul...
30:usbmux:x:52:83:User for us...
...```
pipethread

Script:

```bash
#!/usr/bin/env bash

proc tac {inchan outchan} {
    set output [list]
    while {![eof $inchan]} {
        gets $inchan line
        set output [linsert $output 0 $line]
    }
    puts $outchan [join $output "\n"]
}

set number -1
pipethread::pipe exec cat /etc/passwd | foreach line {
    incr number
    puts $outchan "$number:$line"
} | tac
```
Case Studie

These contrived example
while true; do
    date +%s
    sleep 60
done | while read now; do
    for vmId in $(vmList); do
        if applicableSnapshot $vmId $now; then
            echo takeSnapshot $vmId $now
        fi
    done
done | while read command vmId now; do
    takeSnapshot --vm $vmId --id $now
    echo uploadSnapshots $vmId
done | while read command vmId; do
    uploadSnapshots "${vmId}"
done
pipethread::pipe {
    while true {
        puts $outchan [clock seconds]
        flush $outchan
        after 60000
    }
} | foreach now {
    foreach vmId [vmList] {
        if {[applicableSnapshot $vmId $now]} {
            puts $outchan [list takeSnapshot $vmId $now]
        }
    }
} | foreach line {
    set vmId [lindex $line 1]
    takeSnapshots $vmId
    puts $outchan [list uploadSnapshot $vmId]
} | foreach line {
    set vmId [lindex $line 1]
    takeSnapshots $vmId
}
POSIX Shell

```bash
ceph --watch | while IFS=' ' read -r line; do
  # spend a lot of time parsing the data
done | while IFS=' ' read -r sql; do
  # Use "sqlite3" to update a database in
  # a single transaction
done
```
pipethread

```
sqlite3 db ...
pipethread::pipe exec ceph --watch | foreach line {
    # Parse the data -- now easier in Tcl
}
| foreach infoDict {
    unset -nocomplain info
    array set info $infoDict
    db transaction {
        # Much easier to deal with SQL as a completely
        # different stage with a Tcl array
        db eval {...}
    }
}
```
Our princess is in another castle

pipethread supports a different kind of mode -- the asynchronous mode.
Asyncronous pipethread

```tcl
proc newConnection {sock addr port} {
    pipethread::pipe -inchan $sock -outchan $sock \ 
    -async [list close $sock] -- foreach cmd {
        switch -exact -- $cmd {
            "hello" {
                puts $outchan "Hi !"
            }
            "quit" {
                break
            }
        }
    } | foreach line {
        puts $outchan "[string length $line]:$line"
    }
}
socket -server newConnection 3030
vwait forever
```
One more thing...

```
pipethread::pipe -inch $sock -out $sock {...} | {...}
```

Looks simple... but threads can't share channels. 😞
One more thing, again...

A tale of two loops

```tcl
pipethread::pipe {
    for {set idx 0} {1} {incr idx} {
        puts $outchan $idx
    }
} | foreach line {
     puts $line
    after 1000
}
```

Enter pipethread::infiniteBuffer
Thank You!

For more information, see the pipethread Fossil repository: https://chiselapp.com/user/rkeene/repository/pipethread/

Wiki:
http://wiki.tcl.tk/pipethread

Questions?