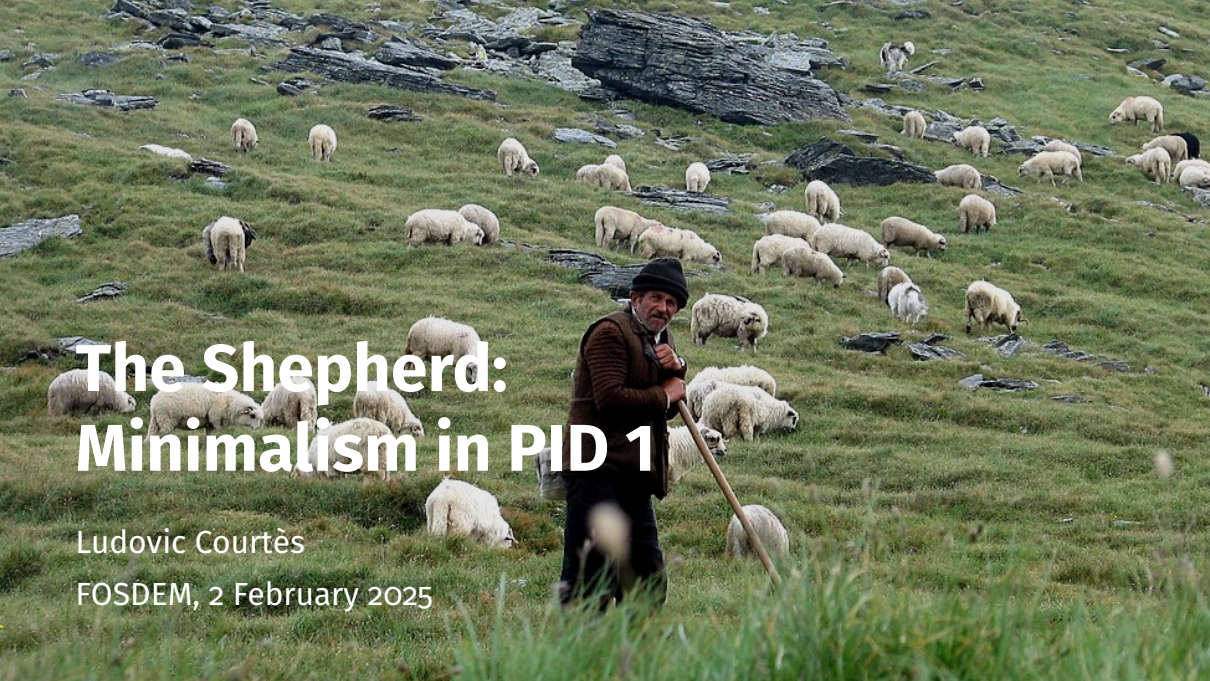


The Shepherd: Minimalism in PID 1

Ludovic Courtès

FOSDEM, 2 February 2025

From: Wolfgang Jaehrling
Subject: Announcement: dmd -0.7
Date: Wed, 2 Apr 2003 21:44:52 +0200

```
;; Announcement: dmd -0.7                                -*- scheme -*-

(define about-this-release
  "This is the second public release of dmd, the ‘Daemon managing
Daemons’ (or ‘Daemons-managing Daemon’?).  This version adds many
convenient features and is thus a huge step forward, but still it is
experimental software and you should expect it to break.")

(define about-the-software
  "The dmd program is a service manager, i.e. on the GNU system (which
is the primary target), it replaces /sbin/init completely, on systems
which are similar to Unix (e.g. GNU/Linux) it replaces the part of
/sbin/init that is responsible for switching runlevels (/etc/rc?.d and
/etc/init.d come to mind), respawning services (/etc/inittab comes to
mind) and similar things.")
```

From: Wolfgang Jaehrling
Subject: Announcement: dmd -0.7
Date: Wed, 2 Apr 2003 21:44:52 +0200
;; Announcement: dmd -0.7 -*- scheme -*-

(define about-this-release
"This is the second public release of dmd, the 'Daemon managing Daemons' (or 'Daemons-m... This version adds many convenient features and..., but still it is experimental software and ...break.")

(define about-the-software
"The dmd program is a service manager, i.e. on the GNU system (which is the primary target), it replaces /sbin/init completely, on systems which are similar to Unix (e.g. GNU/Linux) it replaces the part of /sbin/init that is responsible for switching runlevels (/etc/rc?.d and /etc/init.d come to mind), respawning services (/etc/inittab comes to mind) and similar things.")

Revived in 2013!

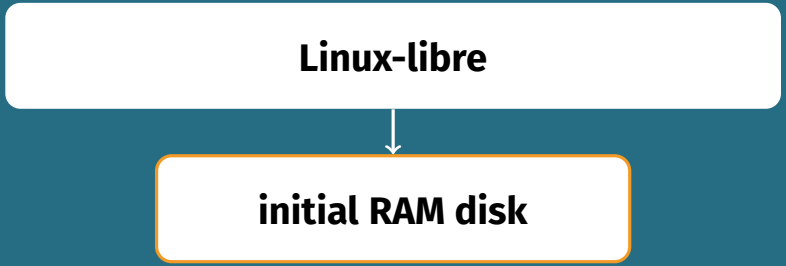
From: Wolfgang Jaehrling
Subject: Announcement: dmd -0.7
Date: Wed, 2 Apr 2003 21:44:52 +0200

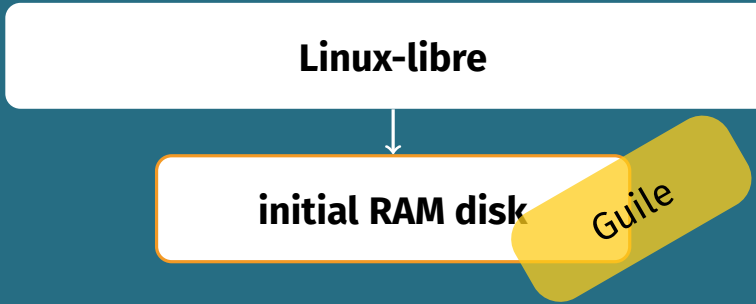
```
;; Announcement: dmd -0.7                                -*- scheme -*-

(define about-this-release
  "This is the second public release of dmd, the 'Daemon managing
Daemons' (or 'Daemons-m                    This version adds many
convenient features and                    , but still it is
experimental software a                    break.")

(define about-the-software
  "The dmd program is a service manager, i.e. on the GNU system (which
is the prim                                systems
which are s                                art of
/sbin/init                                 /rc?.d and
/etc/init.d come to mind), respawning services (/etc/inittab comes to
mind) and similar things.")
```

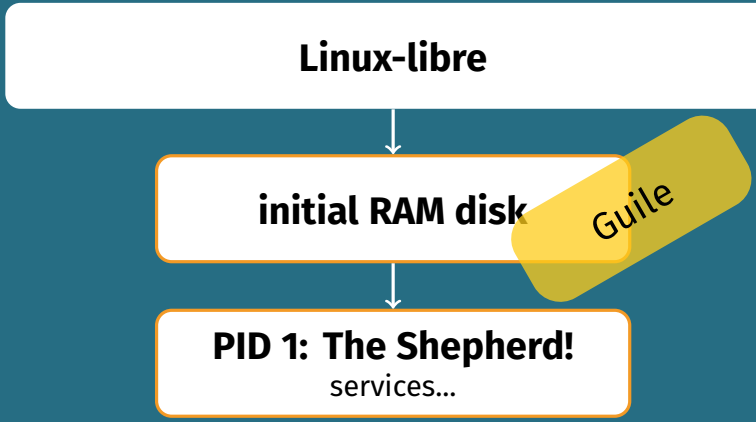
Revived in 2013!

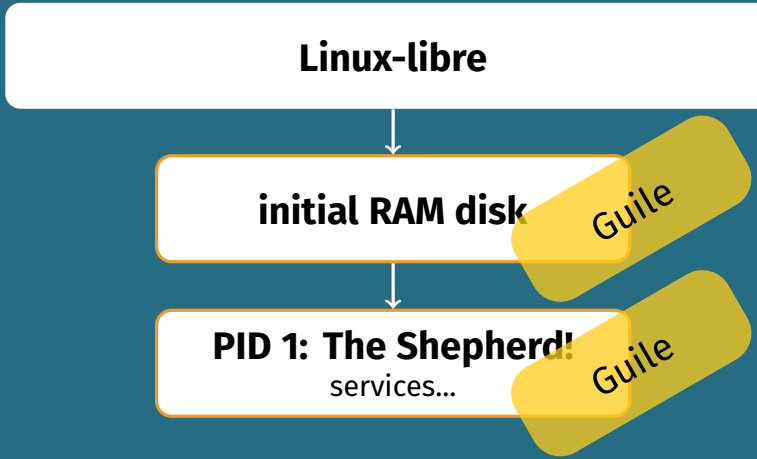
1.0 in December 2024!

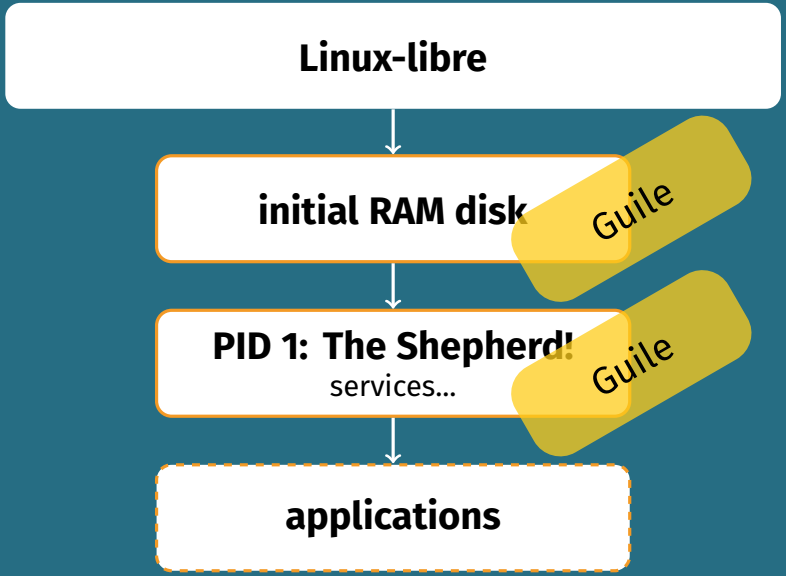
Linux-libre


Linux-libre
initial RAM disk


Linux-libre
initial RAM disk
Guile


Linux-libre
initial RAM disk
Guile
PID 1: The Shepherd!
services...


Linux-libre
initial RAM disk
Guile
PID 1: The Shepherd!
services...
Guile


Linux-libre
initial RAM disk
Guile
PID 1: The Shepherd!
services...
Guile
applications

Demo time!

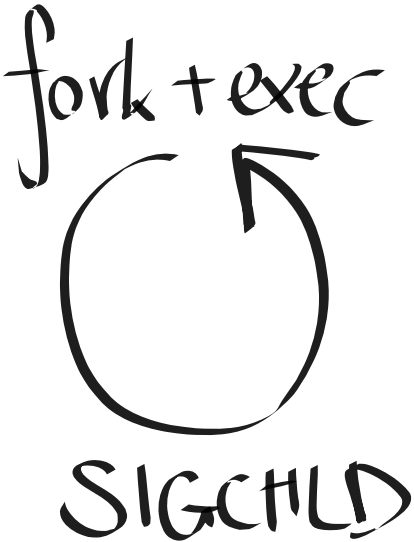
```
(define sshd
  (service
    '(sshd ssh-daemon)  ;for convenience, give it two names
    #:start (make-forkexec-constructor
             '("/usr/sbin/sshd" "-D")
             #:pid-file "/etc/ssh/sshd.pid")
    #:stop (make-kill-destructor)
    #:respawn? #t))

(register-services (list sshd))
(start-in-the-background '(sshd))
```

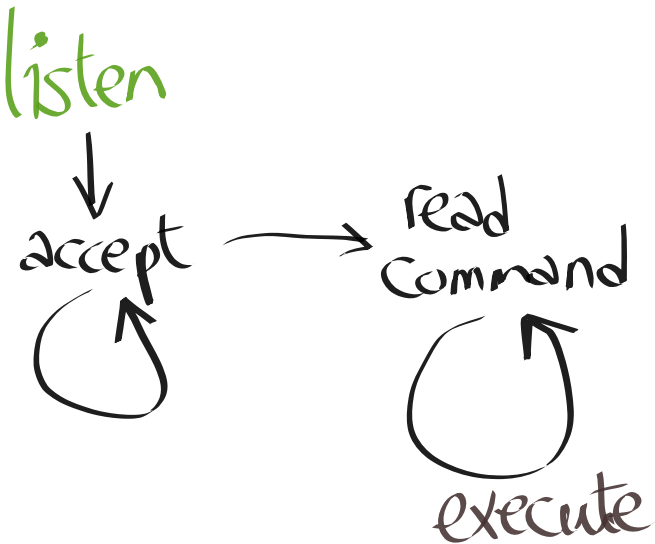
```
(define sshd
  (service
    '(sshd ssh-daemon)
    #:start (make-inetd-constructor
             '("/usr/sbin/sshd" "-D" "-i")
             (list (endpoint
                    (make-socket-address AF_INET INADDR_ANY 22))
                   (endpoint
                    (make-socket-address AF_INET6 IN6ADDR_ANY 22)))
             #:max-connections 10)
    #:stop (make-inetd-destructor)
    #:respawn? #t))
```

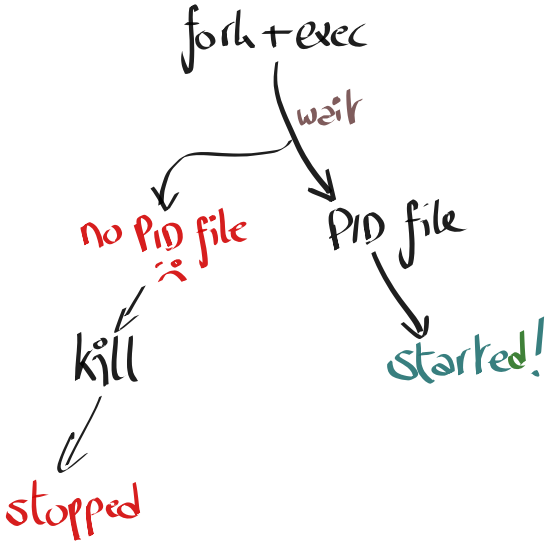
```
(use-modules (shepherd service timer))

(define updatedb                ;for 'locate'
  (service
    '(updatedb)
    #:start (make-timer-constructor
             ;; Fire at midnight and noon everyday.
             (calendar-event #:hours '(0 12) #:minutes '(0))
             (command '("/usr/bin/updatedb"
                        "--prunepaths=/tmp")))
    #:stop (make-timer-destructor)
    #:actions (list timer-trigger-action)))
```

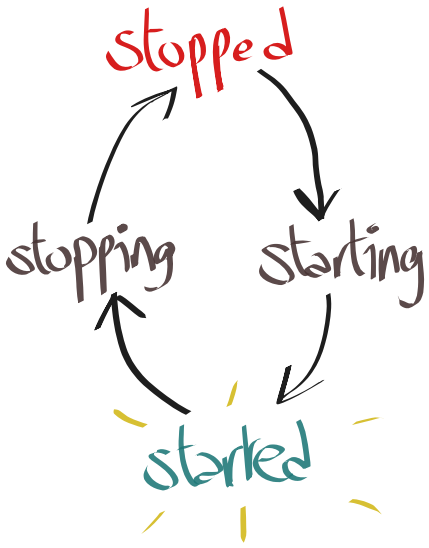
```
(define file-system-/mnt/disk
  (service
    '(file-system-/mnt/disk)
    #:start (lambda _
              (mount "/dev/sdb2" "/mnt/disk" "ext4"))
    #:stop (lambda _
             (umount "/mnt/disk"))))
```

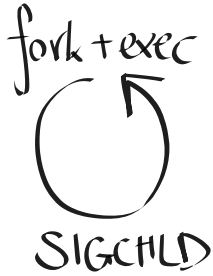
Blurring the line
between users and developers
to increase user autonomy.

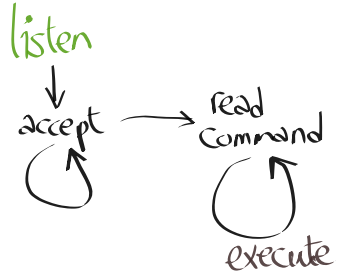

fork + exec
SIGCHLD

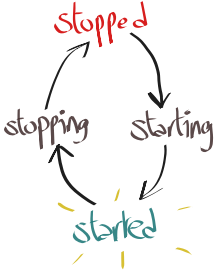

listen
accept
read
command
execute


fork + exec
wait
no PID file
PID file
kill
started!
stopped


stopped
starting
started
stopping


fork + exec
SIGCHLD


listen
accept
read
command
execute


stopped
starting
started
stopping

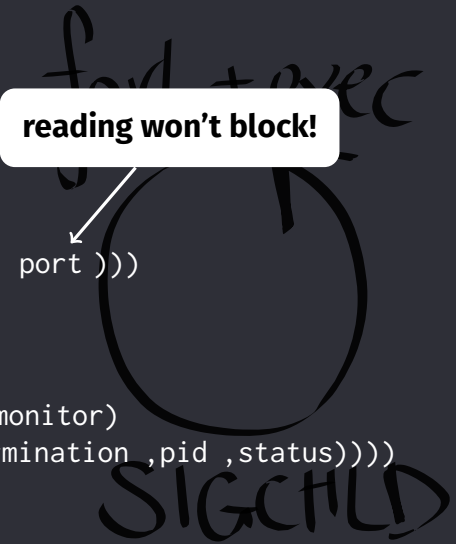

fork + exec
wait
no PID file
PID file
kill
started!
stopped

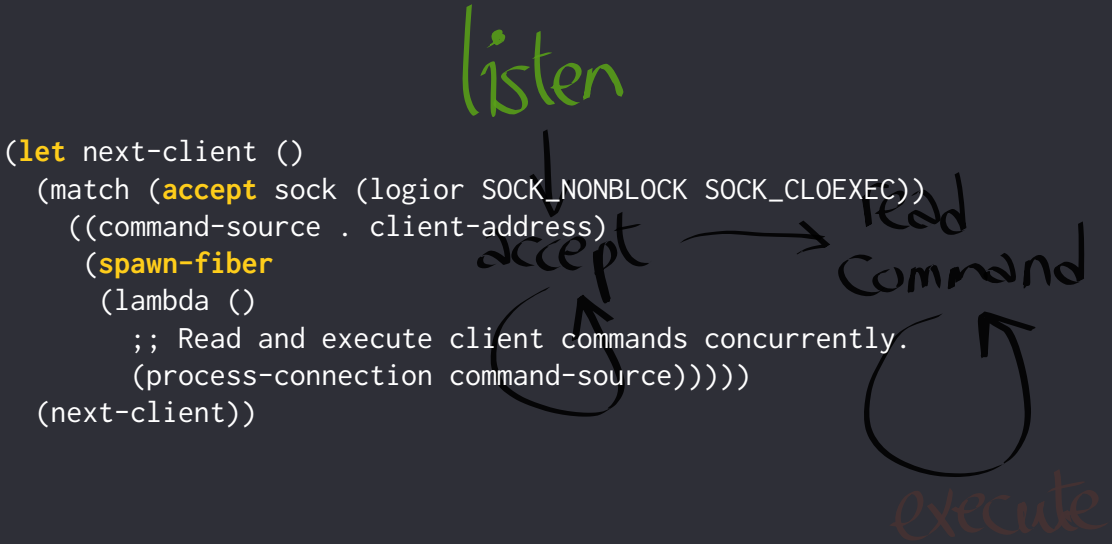
After a **fork()** in a multithreaded program, the child can safely call only async-signal-safe functions (see **signal-safety(7)**) until such time as it calls **execve(2)**.

```
   low-level states for this though so that software can distinguish the perma
 * state from this transitionary UNIT_INACTIVE state by looking at the low-lev
if (s->restart_mode != SERVICE_RESTART_MODE_DIRECT)
        service_set_state(s, restart_state);

restart_usec_next = service_restart_usec_next(s);

r = service_arm_timer(s, /* relative= */ true, restart_usec_next);
if (r < 0) {
        log_unit_warning_errno(UNIT(s), r, "Failed to install restart timer: %
        service_enter_dead(s, SERVICE_FAILURE_RESOURCES, /* allow_restart= */
        return;
}

log_unit_debug(UNIT(s), "Next restart interval calculated as: %s", FORMAT_TIME

service_set_state(s, SERVICE_AUTO_RESTART);
```

```
(let loop ()
  (handle-signal-port signal-port)
  (loop))
```

```
(let loop ()
  (handle-signal-port signal-port)
  (loop))

(define (handle-signal-port port)
  (let ((signal (consume-signalfd-siginfo  port )))
    (if (= SIGCHLD signal)
        (match (waitpid WAIT_ANY WNOHANG)
          ((pid . status)
           (put-message (current-process-monitor)
                        `(handle-process-termination ,pid ,status))))
        ...)))
```

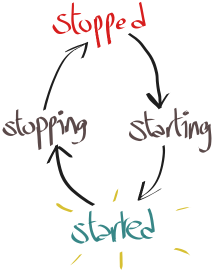


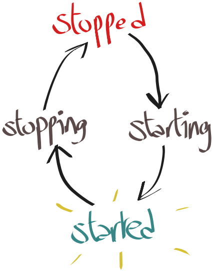


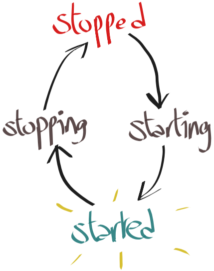

```
(let next-client ()
  (match (accept sock (logior SOCK_NONBLOCK SOCK_CLOEXEC))
    ((command-source . client-address)
     (spawn-fiber
      (lambda ()
        ;; Read and execute client commands concurrently.
        (process-connection command-source)))))
  (next-client))
```

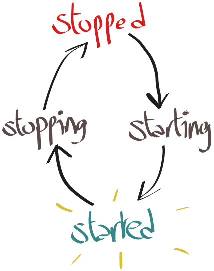
```
(define* (terminate-process pid signal
                            #:key (grace-period 5))
  ;; Send signal to PID, or SIGKILL after grace period.
  (let ((reply (make-channel)))
    (put-message (current-process-monitor)
                 `(await ,pid ,reply))
    (catch-system-error (kill pid signal))

    (match (get-message* reply grace-period #f)
      (#f
       ;; Grace period is over, send SIGKILL.
       (catch-system-error (kill pid SIGKILL))
       (get-message reply))
      (status
       ;; Terminated on time!
       status))))
```

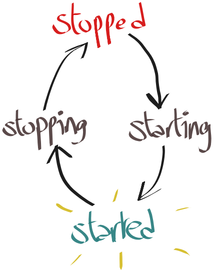
```
(define (service-controller channel)
  (let loop ((status 'stopped)
             (value #f)
             ...)
    (match (get-message channel)
      (('status reply)     ;return the current status
       (put-message reply status)
       (loop status value ...))
      (('start reply)      ;start the service
       ...)
      (('stop reply)       ;stop it
       ...)
      ...)))
```

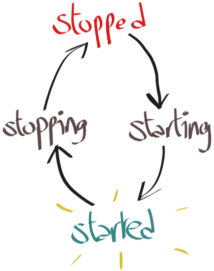

stopped
stopping
starting
started

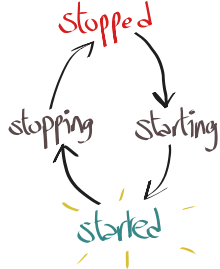

stopped
stopping
starting
started

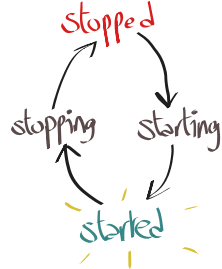

stopped
stopping
starting
started


stopped
stopping
starting
started

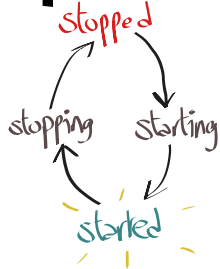

stopped
stopping
starting
started

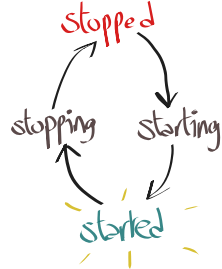

stopped
stopping
starting
started





Concurrent sequential processes.





Sweat & tears.

```
(define (alice channel)
  (let loop ()
    (match (get-message channel)
      (('say-hi-to-me reply)
       (put-message reply 'hi!)))))
```

```
(define (bob friend)
  (let ((reply (make-channel)))
    (put-message friend
      `(say-hi-to-me ,reply))
    'bye-bye!))
```

```
(define (alice channel)
  (let loop ()
    (match (ge
      (('say-h
       (put-message reply "hi!")))))
```

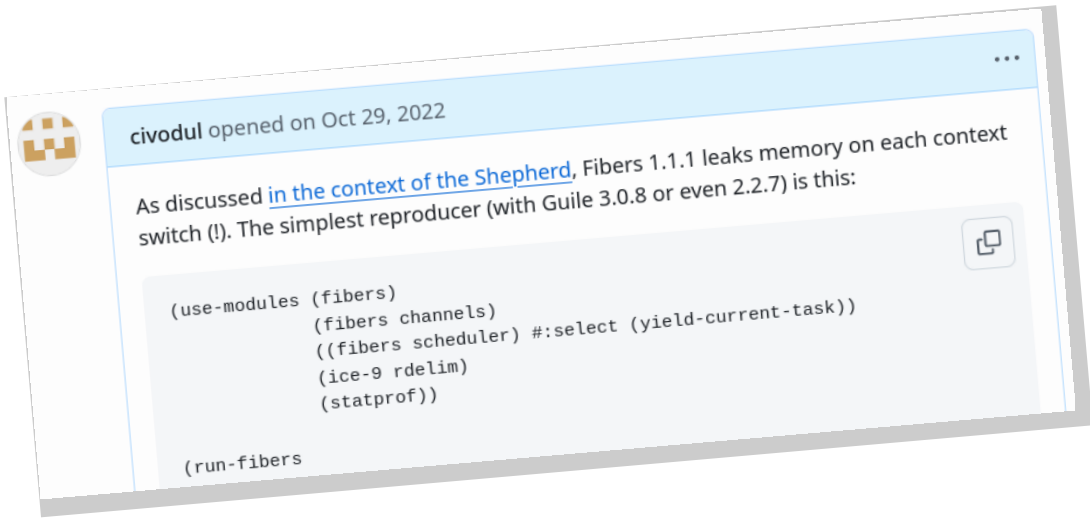
```
(define (bob friend)
  (let ((reply (make-channel)))
    friend
    o-me ,reply))
    bye bye!))
```

Do not miss a rendez-vous.

“Hello, I wrote my own service and now `herd status` *hangs.”*

```
thread apply all bt
```

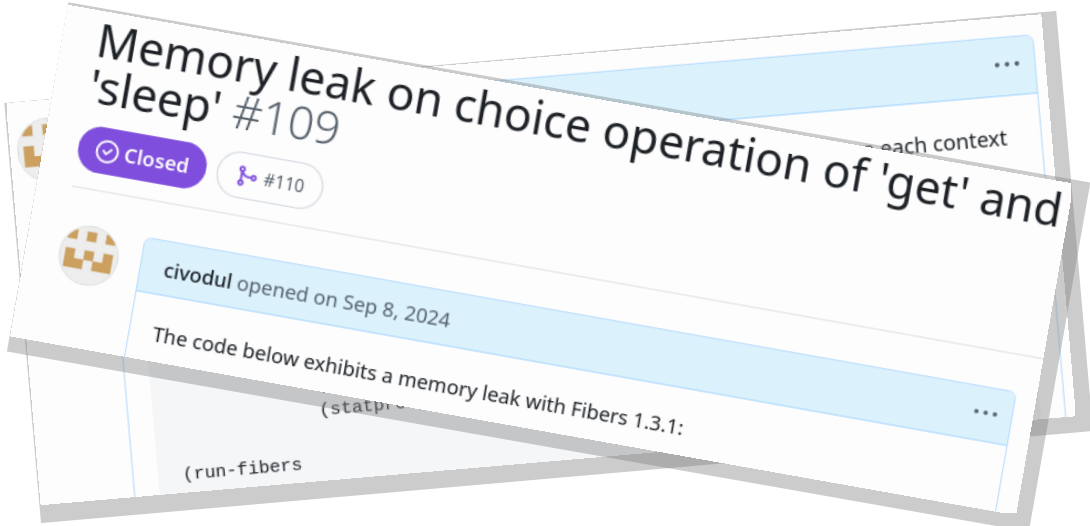
... would be nice.

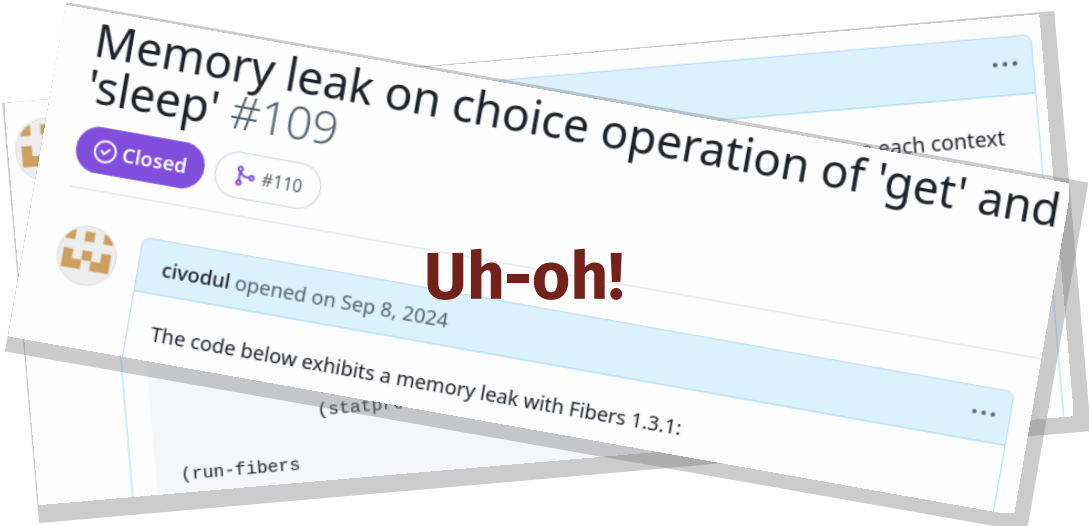


civodul opened on Oct 29, 2022

As discussed in the context of the Shepherd, Fibers 1.1.1 leaks memory on each context switch (!). The simplest reproducer (with Guile 3.0.8 or even 2.2.7) is this:

```
(use-modules (fibers)
             (fibers channels)
             ((fibers scheduler) #:select (yield-current-task))
             (ice-9 rdelim)
             (statprof))


(run-fibers
```


Memory leak on choice operation of 'get' and 'sleep' #109
Closed
#110
civodul opened on Sep 8, 2024
The code below exhibits a memory leak with Fibers 1.3.1:
(run-fibers


Memory leak on choice operation of 'get' and 'sleep' #109
Closed
#110
Uh-oh!
civodul opened on Sep 8, 2024
The code below exhibits a memory leak with Fibers 1.3.1:
(run-fibers

```
herd status
```

Systemd Fixes Bug While Facing New Challenger in GNU Shepherd

Dec 17, 2024

Share

Jack Wallen

The systemd developers have fixed a really nasty bug amid the release of the new GNU Shepherd init system.

- **1.0.x** bug-fix releases rolling
- **Guix integration**: replacing rottlog, mcron, syslogd
- **Fibers** is solid! (and needs ♡)
- **7.4K lines of code!**
- miss something from systemd? **let's hack!**


THE
GOBLIN
AND THE
SHEPHERD
A TALE AS TOLD BY
JULIANA SIMS

https://gnu.org/s/shepherd

ludo@gnu.org | @civodul@toot.aquilenet.fr

Picture of Shepherd in Făgăraș Mountains, Romania, by “friend of Darwinek”, under CC-BY-SA 3.0, https://commons.wikimedia.org/wiki/File:Rumunia_5806.jpg

Goblins + Spritely drawing by Spritely Institute, https://spritely.institute/news/spritely-nlnet-grants-december-2023.html

Shepherd logo by Luis Felipe López Acevedo, under CC-BY-SA 4.0, https://git.savannah.gnu.org/cgit/shepherd/graphics.git.

The source of this document is available from https://git.sv.gnu.org/cgit/guix/maintenance.git.