Daemons in Ruby

Jose Luis Salas

September 23, 2014

Presentation outline

Let's start

Consider this code

```
loop do
  # Do stuff here
  sleep 1
end
```

Let's start

Consider this code

```
loop do
  # Do stuff here
  sleep 1
end
```

Is this a daemon?

Let's start

Consider this code

```
loop do
  # Do stuff here
  sleep 1
end
```

Is this a daemon? No, it isn't

What's a daemon

Daemons are processes on a system which principal characteristics are:

What's a daemon

Daemons are processes on a system which principal characteristics are:

- Aren't directly controlled by the user

What's a daemon

Daemons are processes on a system which principal characteristics are:

- Aren't directly controlled by the user
- Run in background

What's a daemon

Daemons are processes on a system which principal characteristics are:

- Aren't directly controlled by the user
- Run in background
- Are spawned tipically by init manager

What's a daemon

Daemons are processes on a system which principal characteristics are:

- Aren't directly controlled by the user
- Run in background
- Are spawned tipically by init manager
- Tipically log data into logfiles

What's a daemon

Daemons are processes on a system which principal characteristics are:

- Aren't directly controlled by the user
- Run in background
- Are spawned tipically by init manager
- Tipically log data into logfiles
- Are detached from terminal

Presentation outline

Process

Process

- What is a process?

Process

- What is a process?
  Is an instance of a computer program that is being executed

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created?

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created? fork in UNIX

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created? fork in UNIX, CreateProcess on the rest

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created? fork in UNIX, CreateProcess on the rest
- What is shared between processes?

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created? fork in UNIX, CreateProcess on the rest
- What is shared between processes? Libraries?

Process

- What is a process?
  Is an instance of a computer program that is being executed
- How processes are created? fork in UNIX, CreateProcess on the rest
- What is shared between processes? Libraries? Memory?

What is a process consisted of?

What is a process consisted of?

- An image of the executable machine code.

What is a process consisted of?

- An image of the executable machine code.
- Virtual memory which includes the executable code, process-specific data, a call stack, and a heap to hold intermediate computation data generated during run time.

What is a process consisted of?

- An image of the executable machine code.
- Virtual memory which includes the executable code, process-specific data, a call stack, and a heap to hold intermediate computation data generated during run time.
- Operating system descriptors of resources, such as file descriptors.

What is a process consisted of?

- An image of the executable machine code.
- Virtual memory which includes the executable code, process-specific data, a call stack, and a heap to hold intermediate computation data generated during run time.
- Operating system descriptors of resources, such as file descriptors.
- Security attributes, such as the process owner and the process' set of permissions.

What is a process consisted of?

- An image of the executable machine code.
- Virtual memory which includes the executable code, process-specific data, a call stack, and a heap to hold intermediate computation data generated during run time.
- Operating system descriptors of resources, such as file descriptors.
- Security attributes, such as the process owner and the process' set of permissions.
- Processor state, such as the content of registers, physical memory addressing, etc. The state is typically stored in CPU registers when the process is executing, and in memory otherwise.

Executing a program

```
puts("Hello from process #{Process.pid}")
exec('uname -r')
puts("Bye from process")
```

Executing a program

```
puts("Hello from process #{Process.pid}")
exec('uname -r')
puts("Bye from process")
```

Hello from process 23191
3.14-2-amd64

Executing programs

- Kernel.exec

Executing programs

- Kernel.exec : Replaces the current process image
- Kernel.system

Executing programs

- Kernel.exec : Replaces the current process image
- Kernel.system : Executes cmd in a subshell
- Kernel.‘ (backticks)

Executing programs

- Kernel.exec : Replaces the current process image
- Kernel.system : Executes cmd in a subshell
- Kernel.‘ (backticks) : Executes cmd in a subshell
- IO.popen

Executing programs

- Kernel.exec : Replaces the current process image
- Kernel.system : Executes cmd in a subshell
- Kernel.‘ (backticks) : Executes cmd in a subshell
- IO.popen : Runs command as a subprocess
- Open3.popen3
- Process.spawn
- Process.daemon
- IO.popen4 (JRuby)

Creating a new process

```
puts("I am process #{Process.pid}")
fork
puts("I am process #{Process.pid}")
```

Creating a new process

```
puts("I am process #{Process.pid}")
fork
puts("I am process #{Process.pid}")
```

I am process 23823
I am process 23823
I am process 23825

Executing a new program in a new process

```
puts("I am process #{Process.pid}")
pid = fork do
  puts("I am process #{Process.pid}, my
     parent is #{Process.ppid}")
end

puts("I am process #{Process.pid}, I am
   waiting for process #{pid}")
Process.wait(pid)
```

Executing a new program in a new process

```
puts("I am process #{Process.pid}")
pid = fork do
  puts("I am process #{Process.pid}, my
     parent is #{Process.ppid}")
end

puts("I am process #{Process.pid}, I am
   waiting for process #{pid}")
Process.wait(pid)
```

I am process 11135
I am process 11135, I am waiting for process 11137
I am process 11137, my parent is 11135

Trying to become a daemon

```
def print_process_info
  puts("PID: #{Process.pid} PPID:
     #{Process.ppid} SID: #{Process.getsid}
     PGRP: #{Process.getpgrp}")
end

print_process_info
exit if fork
print_process_info
Process.getsid
print_process_info
exit if fork
print_process_info
```

Trying to become a daemon part 2

```
$ ruby process5.rb
PID: 5764 PPID: 4755 SID: 4755 PGRP: 5764
PID: 5805 PPID: 5764 SID: 4755 PGRP: 5764
PID: 5805 PPID: 5764 SID: 4755 PGRP: 5764
PID: 5808 PPID: 5805 SID: 4755 PGRP: 5764

$ ps -o ppid,pid,sid,pgid,command
PPID   PID   SID  PGID COMMAND
   1  5808  4755  5764 ruby process5.rb
```

Communicating processes with pipes

```
pipe_me_in, pipe_child_out = IO.pipe
pipe_child_in, _ = IO.pipe

fork do
  STDIN.reopen(pipe_child_in)
  STDOUT.reopen(pipe_child_out)
  exec("echo valencia.rb")
end

pipe_child_out.close
puts(pipe_me_in.read)
```

Communicating processes with pipes

```
pipe_me_in, pipe_child_out = IO.pipe
pipe_child_in, _ = IO.pipe

fork do
  STDIN.reopen(pipe_child_in)
  STDOUT.reopen(pipe_child_out)
  exec("echo valencia.rb")
end

pipe_child_out.close
puts(pipe_me_in.read)
```

valencia.rb

Signals

```
trap(:KILL) do
  puts('I am not going to die!')
end
puts('Trying to kill myself')
Process.kill(:KILL, Process.pid)
puts('I should be alive')
```

Signals

```
trap(:KILL) do
  puts('I am not going to die!')
end
puts('Trying to kill myself')
Process.kill(:KILL, Process.pid)
puts('I should be alive')
```

```
Trying to kill myself
Killed
```

Presentation outline

Steps to become a UNIX daemon

Steps to become a UNIX daemon

- Dissociate from the controlling tty

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader
- Execute as a background task by forking and exiting (once or twice), required sometimes to become a session leader.

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader
- Execute as a background task by forking and exiting (once or twice), required sometimes to become a session leader.
- Setting the root directory (/) as the current working directory so that the process does not keep any directory in use.

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader
- Execute as a background task by forking and exiting (once or twice), required sometimes to become a session leader.
- Setting the root directory (/) as the current working directory so that the process does not keep any directory in use.
- Changing the umask to 0 to allow operating system calls to provide their own permission masks and not to depend on the umask of the caller

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader
- Execute as a background task by forking and exiting (once or twice), required sometimes to become a session leader.
- Setting the root directory (/) as the current working directory so that the process does not keep any directory in use.
- Changing the umask to 0 to allow operating system calls to provide their own permission masks and not to depend on the umask of the caller
- Closing all inherited files at the time of execution that are left open by the parent process, including file descriptors 0, 1 and 2.

Steps to become a UNIX daemon

- Dissociate from the controlling tty
- Become a session leader
- Become a process group leader
- Execute as a background task by forking and exiting (once or twice), required sometimes to become a session leader.
- Setting the root directory (/) as the current working directory so that the process does not keep any directory in use.
- Changing the umask to 0 to allow operating system calls to provide their own permission masks and not to depend on the umask of the caller
- Closing all inherited files at the time of execution that are left open by the parent process, including file descriptors 0, 1 and 2.
- Using a logfile, the console, or /dev/null as stdin, stdout, and stderr

Daemonizing in Ruby 1.9

```
Process.daemon
```

Daemonizing in Ruby 1.9

```
Process.daemon
```

- Detach the process from controlling terminal and run in the background as system daemon.
- Unless the argument nochdir is true, it changes the current working directory to /.
- Unless the argument noclose is true, daemon() will redirect standard input, standard output and standard error to /dev/null.
- Return zero on success, or raise one of Errno::*.

proc_daemon

From https://github.com/ruby/ruby/blob/master/process.c

```
static VALUE proc_daemon(int argc, VALUE argv) {
  VALUE nochdir, noclose;
  int n;

  rb_secure(2);
  rb_scan_args(argc, argv, "02", &nochdir, &noclose);

  prefork();
  before_fork();
  n = daemon(RTEST(nochdir), RTEST(noclose));
  after_fork();
  if (n < 0) rb_sys_fail("daemon");
  return INT2FIX(n);
}

#define daemon(nochdir, noclose) rb_daemon(nochdir, noclose)
#endif
```

rb_process

From https://github.com/ruby/ruby/blob/master/process.c

```
static int rb_daemon(int nochdir, int noclose) {
  int n, err = 0;

  switch (rb_fork(0, 0, 0, Qnil)) {
    case -1:
      rb_sys_fail("daemon");
    case 0:
      break;
    default:
      _exit(EXIT_SUCCESS);
  }

  proc_setsid();

  switch (rb_fork(0, 0, 0, Qnil)) {
    case -1:
      return -1;
    case 0:
      break;
    default:
      _exit(EXIT_SUCCESS);
  }

  if (!nochdir) err = chdir("/");

  if (!noclose && (n = open("/dev/null", O_RDWR, 0)) != -1) {
    (void)dup2(n, 0);
    (void)dup2(n, 1);
    (void)dup2(n, 2);
    if (n > 2) (void)close (n);
  }
  return err;
}
```

rb_fork

From https://github.com/ruby/ruby/blob/master/process.c

```
rb_pid_t rb_fork(int *status, int (*chfunc)(void*), void *charg, VALUE fds) {
  if (chfunc) {
    struct chfunc_wrapper_t warg;
    warg.chfunc = chfunc;
    warg.arg = charg;
    return rb_fork_err(status, chfunc_wrapper, &warg, fds, NULL, 0);
  } else {
    return rb_fork_err(status, NULL, NULL, fds, NULL, 0);
  }
}
```

rb_fork_err

From https://github.com/ruby/ruby/blob/master/process.c

```
rb_pid_t rb_fork_err(int *status, int (*chfunc)(void*, char *, size_t),
    void *charg, VALUE fds, char *errmsg, size_t errmsg_buflen) {
  rb_pid_t pid;
  int err, state = 0;

#define prefork() (define\
    rb_io_flush(rb_stdout), \
    rb_io_flush(rb_stderr)rb_stderr\
    )
  prefork();
  // STUFF
  for (; before_fork(), (pid = fork()) < 0; prefork()) {
    after_fork();
    // STUFF
  }
  if (!pid) {
    forked_child = 1;
    if (chfunc) {
      if (!(*chfunc)(charg, errmsg, errmsg_buflen)) _exit(EXIT_SUCCESS);
#if EXIT_SUCCESS == 127
      _exit(EXIT_FAILURE);
#else
      _exit(127);
#endif
    }
  }
  after_fork();
  return pid;
}
```

proc_setsid

From https://github.com/ruby/ruby/blob/master/process.c

```
static VALUE proc_setsid(void) {
  rb_pid_t pid;

  rb_secure(2);
  pid = setsid();
  if (pid < 0) rb_sys_fail(0);
  return PIDT2NUM(pid);
}

#define setsid() ruby_setsid()
```

ruby_setsid.c

From https://github.com/ruby/ruby/blob/master/process.c

```
static rb_pid_t ruby_setsid(void) {
  rb_pid_t pid;
  int ret;

  pid = getpid();
#if defined(SETPGRP_VOID)
  ret = setpgrp();
  /* If 'pid_t setpgrp(void)' is equivalent to setsid(),
   * 'ret' will be the same value as 'pid', and following open() will fail.
   * In Linux, 'int setpgrp(void)' is equivalent to setpgid(0, 0). */
#else
  ret = setpgrp(0, pid);
#endif
  if (ret == -1) return -1;

  if ((fd = open("/dev/tty", O_RDWR)) >= 0) {
    ioctl(fd, TIOCNOTTY, NULL);
    close(fd);
  }
  return pid;
}
#endif
```

proc_setpgrp.c

From https://github.com/ruby/ruby/blob/master/process.c

```
static VALUE proc_setpgrp(void) {
  rb_secure(2);
  /* check for posix setpgid() first; this matches the posix */
  /* getpgrp() above.  It appears that configure will set SETPGRP_VOID */
  /* even though setpgrp(0,0) would be preferred. The posix call avoids */
  /* this confusion. */
#ifdef HAVE_SETPGID
  if (setpgid(0,0) < 0) rb_sys_fail(0);
#elif defined(HAVE_SETPGRP) && defined(SETPGRP_VOID)
  if (setpgrp() < 0) rb_sys_fail(0);
#endif
  return INT2FIX(0);
}
#endif
```

How to daemonize a program in Ruby

```ruby
def daemonize_app
  exit if fork
  Process.setsid
  exit if fork
  Dir.chdir("/")
  STDIN.reopen("/dev/null")
  STDOUT.reopen("/dev/null", "a")
  STDERR.reopen("/dev/null", "a")
end
```

Checking it live

```
$ ls -l /proc/2206/fd
total 0
lrwx------ 1 selu selu 64 Sep 21 16:52 0 -> /dev/null
lrwx------ 1 selu selu 64 Sep 21 16:52 1 -> /dev/null
lrwx------ 1 selu selu 64 Sep 21 16:52 2 -> /dev/null
lr-x------ 1 selu selu 64 Sep 21 16:52 3 -> pipe:[595663]
l-wx------ 1 selu selu 64 Sep 21 16:52 4 -> pipe:[595663]
lr-x------ 1 selu selu 64 Sep 21 16:52 5 -> pipe:[595664]
l-wx------ 1 selu selu 64 Sep 21 16:52 6 -> pipe:[595664]
```

```
$ ps xf -o pid,ppid,pgid,sid,command | grep -e [r]uby -e [P]ID
  PID  PPID  PGID   SID COMMAND
 2206     1  2203  2203 ruby -e Process.daemon; sleep 3600
```

Daemonize in daemons gem

From https://github.com/ghazel/daemons/blob/master/lib/daemons/daemonize.rb

```
def daemonize(logfile_name = nil, app_name = nil)
  srand
  safefork and exit

  unless sess_id = Process.setsid
    raise Daemons.RuntimeException.new('cannot detach from controlling
        terminal')
  end

  # Prevent the possibility of acquiring a controlling terminal
  trap 'SIGHUP', 'IGNORE'
  exit if pid = safefork

  $0 = app_name if app_name

  Dir.chdir "/"
  File.umask 0000

  ObjectSpace.each_object(IO) do |io|
    unless [STDIN, STDOUT, STDERR].include?(io)
      begin
        io.clode unless io.closed?
      rescue ::Exception
      end
    end
  end

  redirect_io(logfile_name)
  return sess_id
end
```

Presentation outline

TCP preforking daemon

```ruby
require 'socket'
Process.daemon
socket = TCPServer.open('0.0.0.0', 8080)
wpids = []

5.times do
  wpids << fork do
    loop do
      connection = socket.accept
      connection.puts "Hello from #{Process.pid}"
      connection.close
    end
  end
end

[:INT, :QUIT].each do |signal|
  Signal.trap(signal) do
    wpids.each { |wpid| Process.kill(signal, wpid) }
  end
end
Process.waitall
```

```
$ nc localhost 8080;nc localhost 8080
Hello from 8349
Hello from 8361
```

Improving signals

```ruby
def handle_signal(signal)
  puts("Received #{signal}")
end

self_read, self_write = IO.pipe

%w(INT TERM).each do |sig|
  trap(sig) { self_write.puts(sig) }
end

while readable_io = IO.select([self_read])
  signal = readable_io.first[0].gets.strip
  handle_signal(signal)
end
```

Our robust daemon

```
Process.daemon
socket = TCPServer.open('0.0.0.0', 8080)
wpids = []

5.times do
  wpids << fork do
    run = true
    [:INT, :QUIT].each do |signal|
      Signal.trap(signal, lambda { run = false })
    end
    while run
      connection = socket.accept
      connection.puts("Hello from #{Process.pid}")
      sleep 10
      connection.puts("Goodbye from #{Process.pid}")
      connection.close
    end
  end
end

[:INT, :QUIT].each do |signal|
  Signal.trap(signal) do
    wpids.each { |wpid| Process.kill(signal, wpid) }
  end
end
Process.waitall
```

```
$ nc localhost 8080; killall -INT ruby
Hello from 8328
Goodbye from 8328
```

Our robust daemon with logging

```ruby
Process.daemon
socket = TCPServer.open('0.0.0.0', 8080)
wpids = []

logger = Logger.new('/tmp/logger.log')
logger.level = Logger::INFO

5.times do
  wpids << fork do
    run = true
    [:INT, :QUIT].each do |signal|
      Signal.trap(signal, lambda { run = false })
    end
    while run
      connection = socket.accept
      connection.puts("Hello from #{Process.pid}")
      logger.info("Connection in #{Process.pid}")
      sleep 10
      connection.puts("Goodbye from #{Process.pid}")
      connection.close
    end
  end
end

[:INT, :QUIT].each do |signal|
  Signal.trap(signal) do
    wpids.each { |wpid| Process.kill(signal, wpid) }
  end
end
Process.waitall
logger.close
```

```
$ tail -f /tmp/logger.log
# Logfile created on 2014-09-21 19:46:25 +0200 by logger.rb/v1.2.7
I, [2014-09-21T19:47:12.696949 #9452]  INFO -- : Connection in 9452
I, [2014-09-21T19:47:22.702008 #9449]  INFO -- : Connection in 9449
```

Logs rotation with a signal

From https://github.com/kennethkalmer/daemon-kit/blob/master/lib/daemon_kit/application.rb

```
configuration.trap("HUP") {
  DaemonKit::Application.reopen_logs
}

module DaemonKit
  class Application
    def reopen_logs
      nr = 0
      append_flags = File::WRONLY | File::APPEND
      DaemonKit.logger.info "Rotating logs" if DaemonKit.logger

      ObjectSpace.each_object(File) do |fp|
        next if fp.closed?
        next unless (fp.sync && fp.path[0..0] == "/")
        next unless (fp.fcntl(Fcntl::F_GETFL) & append_flags) == append_flags

        begin
          a, b = fp.stat, File.stat(fp.path)
          next if a.ino == b.ino && a.dev == b.dev
        rescue Errno::ENOENT
        end

        open_arg = 'a'
        if fp.respond_to?(:external_encoding) && enc = fp.external_encoding
          open_arg << ":#{enc.to_s}"
          enc = fp.internal_encoding and open_arg << ":#{enc.to_s}"
        end
        DaemonKit.logger.info "Rotating path: #{fp.path}" if DaemonKit.logger
        fp.reopen(fp.path, open_arg)
        fp.sync = true
        nr += 1
      end # each_object
      nr
    end
  end
end
```

The end

The end

Further info:

- http://codeincomplete.com/posts/2014/9/15/ruby_daemons/
- Working with unix processes by Jesse Storimer
- lib/unicorn/launcher.rb in Unicorn gem
- daemons, dante and daemon-kit gems
- Linux System Programming: Talking Directly to the Kernel and C Library
- Understanding the Linux Kernel

The end

Further info:

- http://codeincomplete.com/posts/2014/9/15/ruby_daemons/
- Working with unix processes by Jesse Storimer
- lib/unicorn/launcher.rb in Unicorn gem
- daemons, dante and daemon-kit gems
- Linux System Programming: Talking Directly to the Kernel and C Library
- Understanding the Linux Kernel

Questions?

The end

Further info:

- http://codeincomplete.com/posts/2014/9/15/ruby_daemons/
- Working with unix processes by Jesse Storimer
- lib/unicorn/launcher.rb in Unicorn gem
- daemons, dante and daemon-kit gems
- Linux System Programming: Talking Directly to the Kernel and C Library
- Understanding the Linux Kernel

Questions? Thanks!