Unix Process

Kien Nguyen Trung

Process id

- Each process has an unique process id (pid)
- Process id was generated incrementally by Operating System until maximum value
- Maximum value of Process id can be change

```
$> cat /proc/sys/kernel/pid_max
32768

$> echo 40000 > /proc/sys/kernel/pid_max
```

Process id

- Each process belong to a parent process (ppid).
- Process 0 create init process (process 1). After that, other processes are children of init process

Process resources

- Each process has it own memory space. Memory space is isolated between processes
- Process are allocated file descriptors

Process resources

```ruby
puts STDIN.fileno
puts STDOUT.fileno
puts STDERR.fileno

hosts = File.open('/etc/hosts')
puts hosts.fileno  # 3

temp = File.open("test_file", "w")
puts temp.fileno # 4

hosts.close

null = File.open("/dev/null")
puts null.fileno # 3
```

Forking

- fork allow a process create another processes
- process which call fork is parent process, process was generated by fork is child process
- child process inherit all resources from parent process
- memory between children and parent was shared by copy on write mechanism

Forking

```ruby
puts "Parent: #{Process.pid}"

Process.fork do
  puts "Child: #{Process.pid}"
end

puts "Parent: #{Process.pid}"
```

```ruby
puts "Parent: #{Process.pid}"

unless Process.fork then
  puts "Child: #{Process.pid}"
else
  puts "Parent: #{Process.pid}"
end
```

```
[kiennt-mac:~/processes]$ ruby fork2.rb
Parent process: 42296
Parent process: 42296
Child process: 42297
[kiennt-mac:~/processes]$
```

Orphaned process

```ruby
puts "Parent #{Process.pid}"

5.times do |i|
  Process.fork do
    sleep 1
    puts "Children process #{i+1}th with pid = #{Process.pid}"
  end
end

puts "Parent exit"
```

Orphaned process

- Parent process was killed before children process
- Children process was transfer to parent of parent process

Process.wait

```ruby
puts "Parent #{Process.pid}"

5.times do |i|
  Process.fork do
    sleep 1
    puts "Children process #{i+1}th with pid = #{Process.pid}"
  end
end

Process.wait
puts "Parent exit"
```

Zombie process

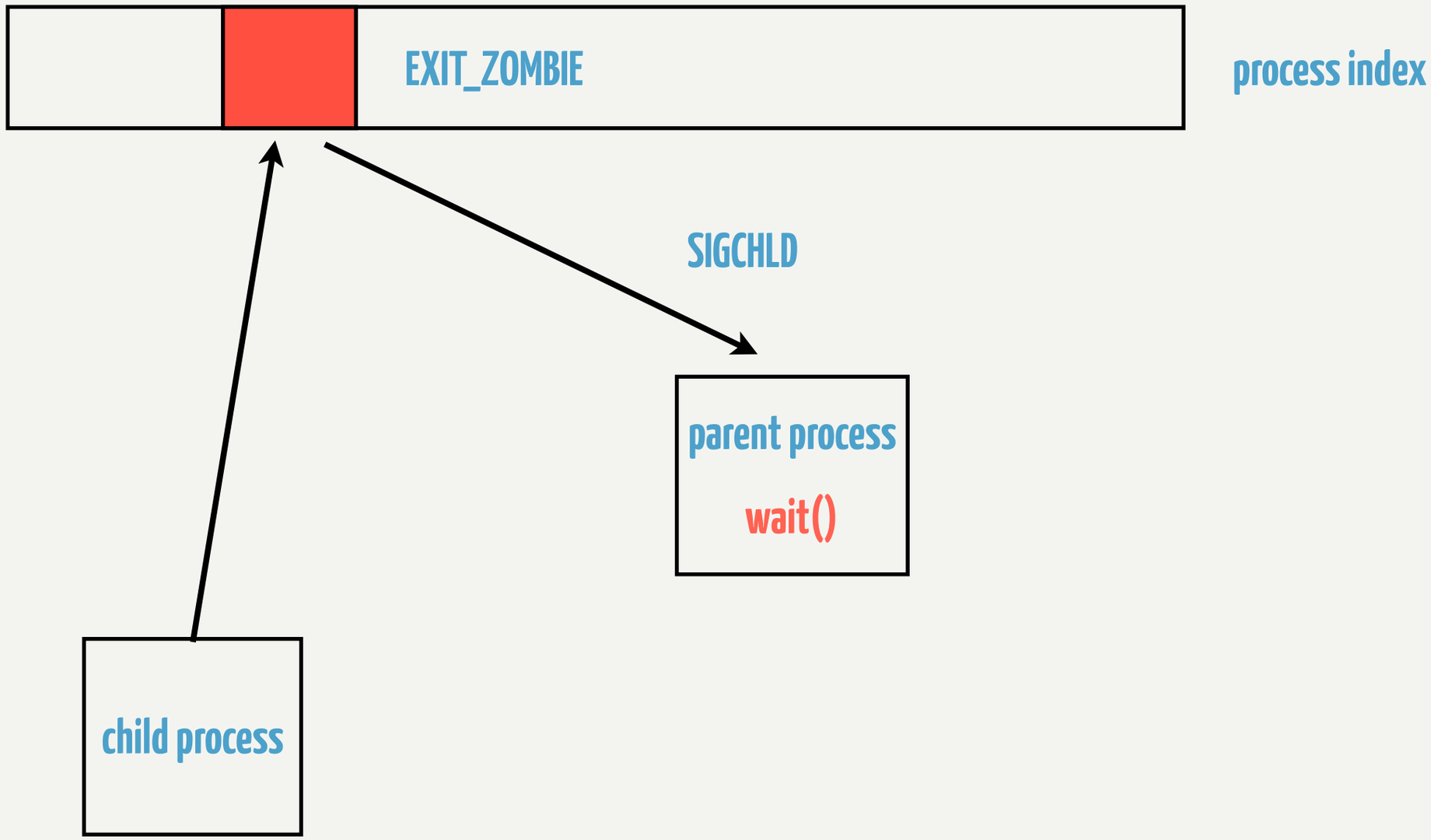
```ruby
puts "Parent process #{Process.pid}"

pid = Process.fork do
  puts "Child process #{Process.pid}"
end

sleep 1000
Process.wait
```

```
[kiennt-mac:~/processes]$ ps -ostate,pid,ppid,command | grep ruby | grep -v grep
S+   42438 40117 ruby zombie-process.rb
Z+   42439 42438 (ruby)
[kiennt-mac:~/processes]$
```

Zombie process



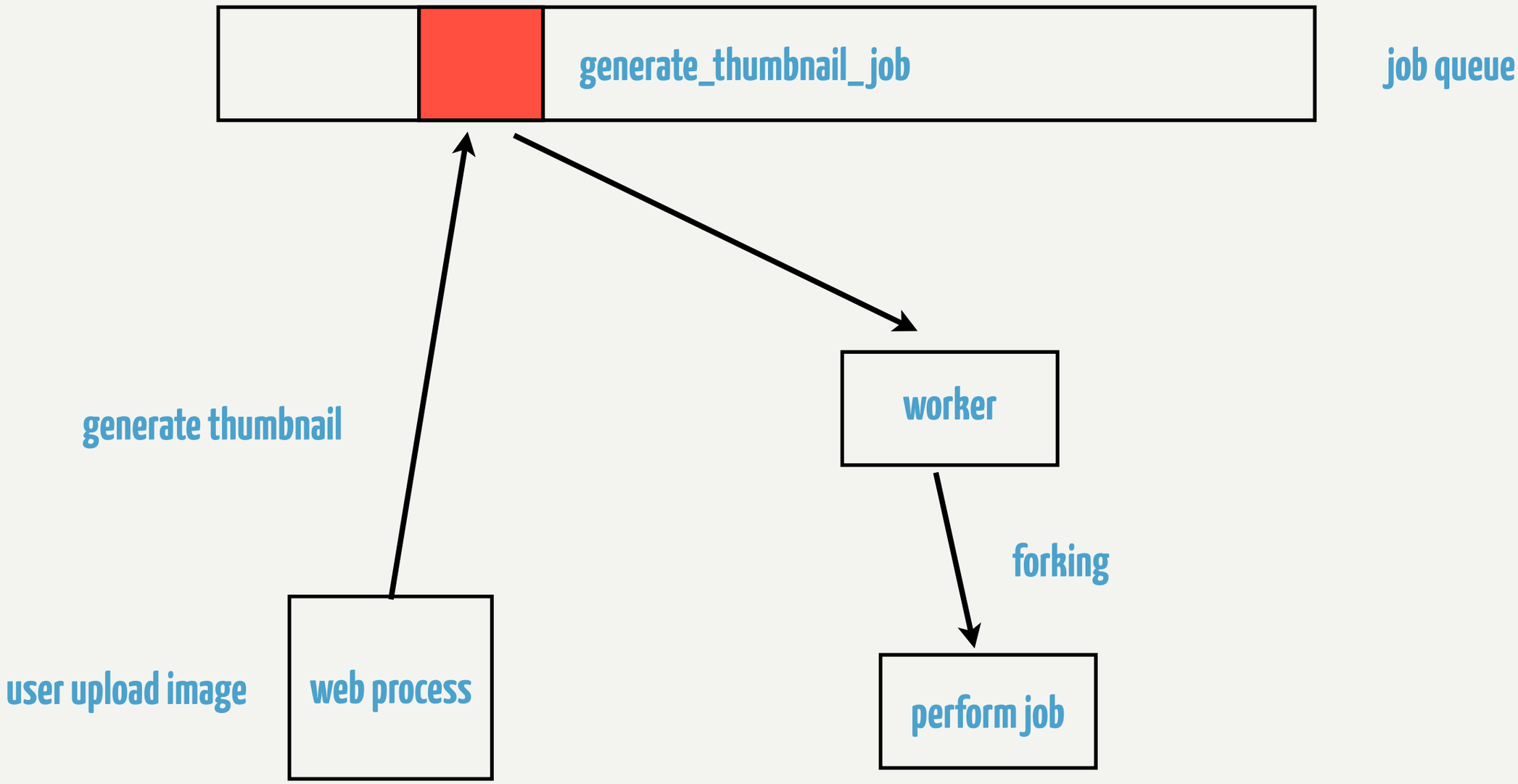
Zombie process

- Zombie processes do not use any system resources
- Each zombie process retain its process id
- To kill zombie process:
  - Make parent process call wait()
  - Kill parent process

Resque

What is Resque?

- Job queue server - open source from GITHUB
- Use REDIS to store job
- Use fork to perform job


generate_thumbnail_job
job queue
generate thumbnail
worker
forking
user upload image
web process
perform job

Why Resque?

- Resque assume chaos
- Resque assume your background workers will lock up, run too long or have unwanted memory growth
- Thanks for forking, jobs that use too much memory release that memory upon complete

https://github.com/defunkt/resque/blob/master/lib/resque/worker.rb

```ruby
if @child = fork(job) do
    unregister_signal_handlers
    procline "Processing #{job.queue} since #{Time.now.to_i}"
    reconnect
    perform(job, &block)
  end
  srand # Reseeding
  procline "Forked #{@child} at #{Time.now.to_i}"
  begin
    Process.waitpid(@child)
  rescue SystemCallError
    nil
  end
  job.fail(DirtyExit.new($?.to_s)) if $?.signaled?
else
  procline "Processing #{job.queue} since #{Time.now.to_i}"
  reconnect
  perform(job, &block)
end
```

Preforking Server

Preforking Server

- Use memory efficiently (thanks for copy on write)
- Load balance automatically
- Use 1 socket for all workers

```ruby
require 'socket'

socket = TCPServer.open('127.0.0.1', 8888)


5.times do |i|
  Process.fork do
    loop do
      connection = socket.accept
      connection.puts "Hello world from worker #{i}"
      connection.close
    end
  end
end


Process.waitall
```

```ruby
require 'socket'

def fork_worker(socket)
  Process.fork do
    loop do
      connection = socket.accept
      connection.puts "Hello world"
      connection.close
    end
  end
end

socket = TCPServer.open('127.0.0.1', 8888)
5.times do |i|
  fork_worker socket
end

trap (:CLD) do
  puts "Child died, restart..."
  fork_worker socket
end

Process.waitall
```

Preforking server

- Unicorn
- MRI is not copy on write friendly

Q&A