6. Processes

- Kernel Abstraction
- Processes Genealogy
- Daemon Processes

Process Tree

Simplified Tree From $ pstree | more

```
init-cron
   |-dhclient3
   |-gdm---gdm-+-Xorg
   |           `-x-session-manag---ssh-agent
   |-5*[getty]
   |-gnome-terminal-+-bash-+-more
   |                |      `-pstree
   |                |-gnome-pty-helper
   |                `-{gnome-terminal}
   |-klogd
   |-ksoftirqd
   |-kthread-+-ata
   |         |-2*[kjournald]
   |         `-kswapd
   |-syslogd
   `-udevd
```

6. Processes

- Kernel Abstraction
- Processes Genealogy
- Daemon Processes

Process Descriptor

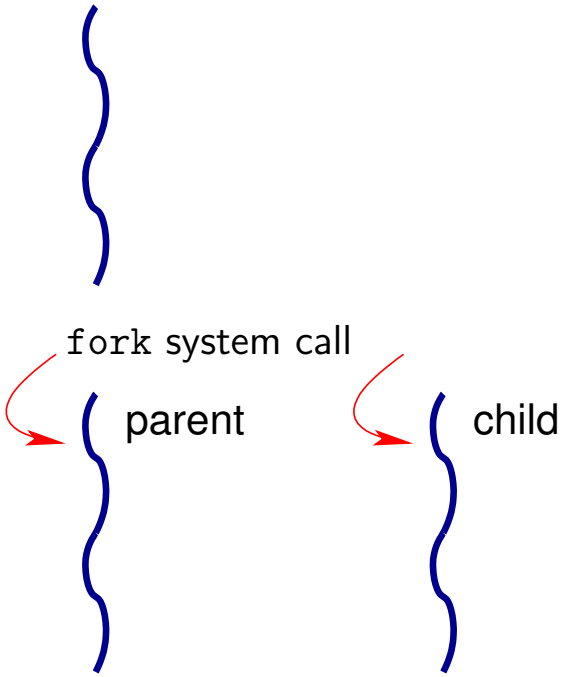
Main Information

State	ready/running, stopped...
Memory map	table of memory regions (pages) attributed to the process
Parent	ID of parent process (allow to obtain PPID)
TTY	control terminal (if any)
Thread	control thread information
Files	current directory and table of file descriptors
Limits	resource limits
Signals	signal handlers, masked and pending signals
Environment	array of pairs (variable name, value string)

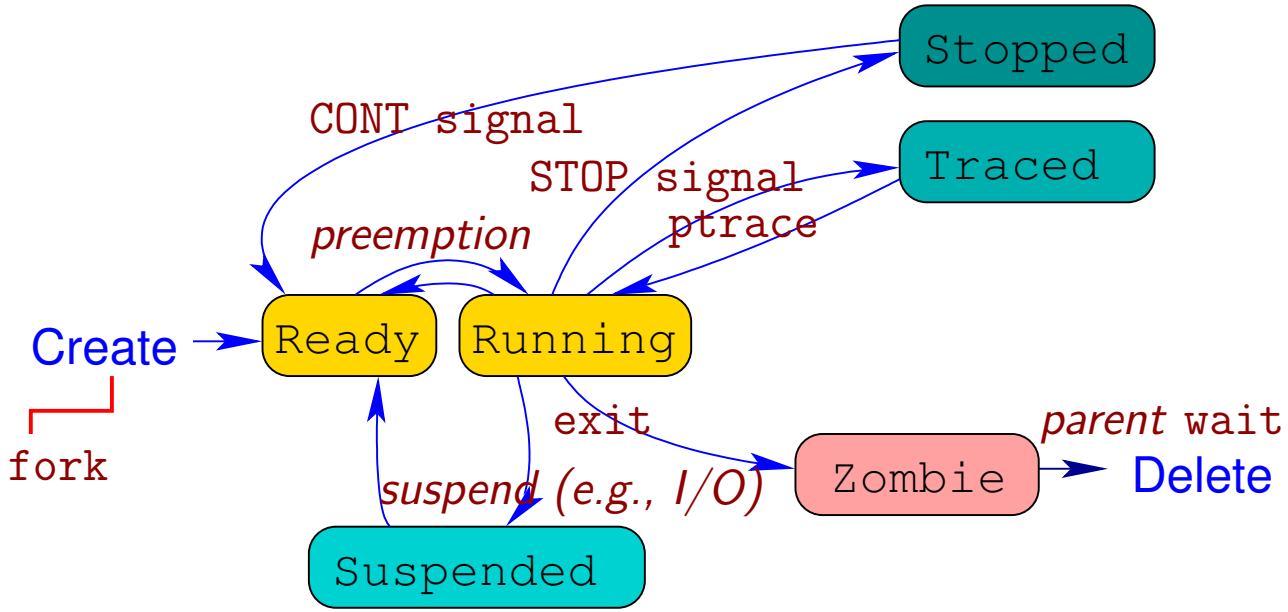
Process Creation

Process Duplication

- Generate a clone of the *parent* process
- The *child* is almost identical
  - It executes the same program
  - In a copy of its virtual memory space



Process States

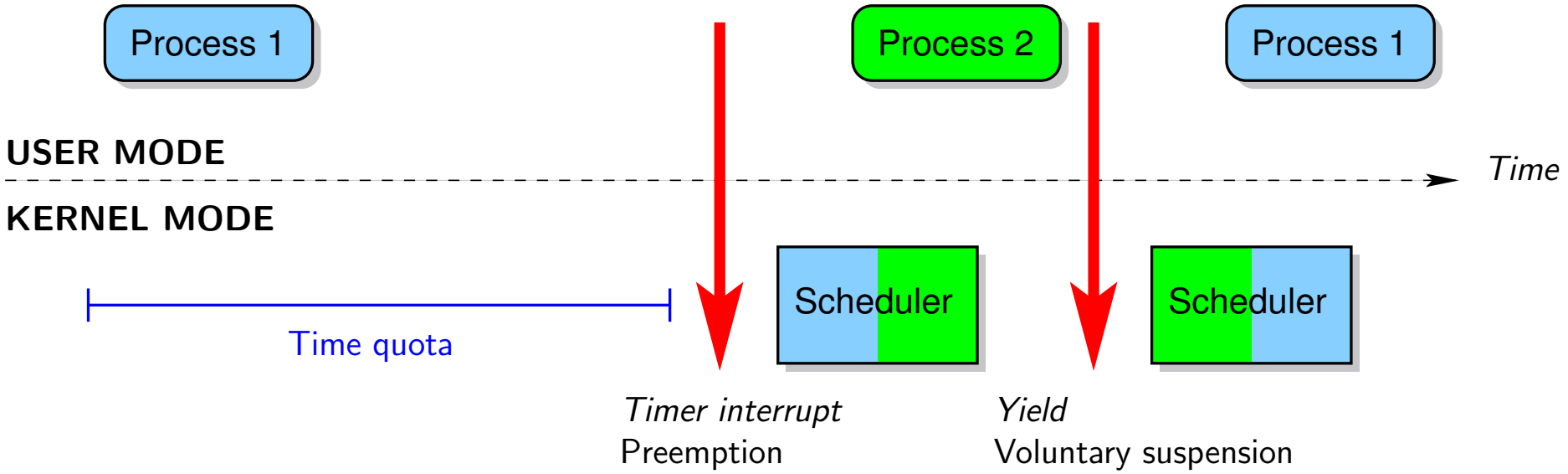


- `Ready` (runnable) process waits to be scheduled
- `Running` process make progress on a hardware thread
- `Stopped` process awaits a continuation signal
- `Suspended` process awaits a wake-up condition from the kernel
- `Traced` process awaits commands from the debugger
- `Zombie` process retains termination status until parent is notified

Process Scheduling

Preemption

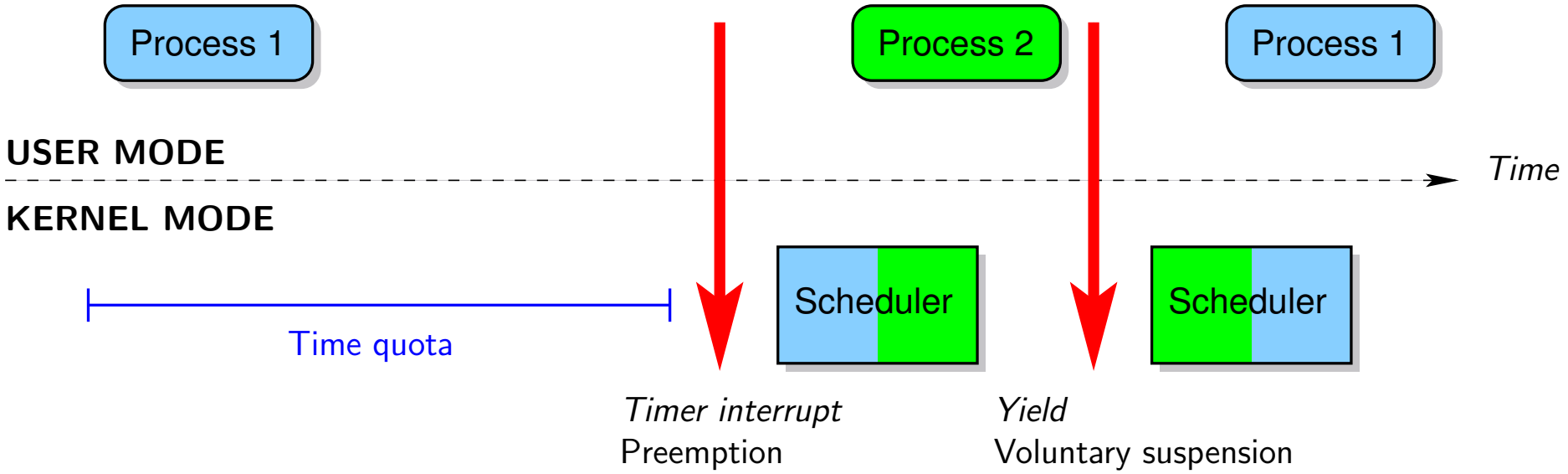
- Default for multiprocessing environments
- Fixed *time quota* (typically $\mathbf{1}\mathrm{ms}$ to $\mathbf{10}\mathrm{ms}$)
- Some processes, called *real-time*, may not be preempted



Process Scheduling

Voluntary Yield

- Suspend execution and yield to the kernel
  - E.g., I/O or synchronization
  - Only way to enable a context switch for *real-time* processes



Process Termination

Java Syntax

```
System.exit(int code);
```

UNIX Command

```
exit [code]
```

Purpose

- Terminates the calling process
  - Closes any open file descriptor
  - Frees all memory pages of the process address space (except shared ones)
  - Any child processes are inherited by process **1** (`init`)
  - If the process is a *session leader* and its *controlling terminal* also controls the session, disassociate the terminal from the session and send a `HUP` signal to all processes in the *foreground group* (terminate process by default)

Process Termination

Java Syntax

```
System.exit(int code);
```

UNIX Command

```
exit [code]
```

Exit Code

- The *exit code* is a *signed byte* (from -128 to 127)
- **0** means normal termination, non-zero indicates an error/warning
- There is no standard list of exit codes

6. Processes

- Kernel Abstraction
- Processes Genealogy
- Daemon Processes

Bootstrap — Kernel Mode

Swapper Process

Process 0

- *One per CPU* (if multiprocessor)
- Built from scratch by the kernel and runs in kernel mode
- Uses *statically*-allocated data
- Constructs memory structures and initializes virtual memory
- Initializes the main kernel data structures
- Creates kernel threads (swap, kernel logging, etc.)
- Enables interrupts, and creates a kernel thread with $\mathrm{PID} = \mathbf{1}$

Bootstrap — User Mode

Init Process

Process 1

- *One per machine* (if multiprocessor)
- Shares all its data with process **0**
- Completes the initalization of the kernel
- Switch to user mode
- Executes `/sbin/init`, becoming a regular process and burying the structures and address space of process **0**

Executing `/sbin/init`

- Builds the OS environment
  - From `/etc/inittab`: type of bootstrap sequence, control terminals
  - From `/etc/rc*.d`: scripts to run system *daemons*
- Adopts all orphaned processes, continuously, until the system halts
- `$ man init` and `$ man shutdown`

Command-Line Operations on Processes

- Cloning and executing
  `$ program arguments &`
- Joining (waiting for completion)
  `$ wait [PID]`
- Signaling events
  `$ kill [-signal] PID`
  `$ killall [-signal] process_name`
  Default signal `TERM` terminates the process
- `$ nohup`: run a command immune to *hang-up* (`HUP` signal)

6. Processes

- Kernel Abstraction
- Processes Genealogy
- Daemon Processes

Sessions

- Orthogonal to process hierarchy
- Session ID = PID of the leader of the session
- Typically associated to user *login*, interactive *terminals*, *daemon* processes
- The *session leader* sends the `HUP` *hang-up* signal to every process belonging to its session, and only if it belongs to the *foreground* group associated to the *controlling terminal* of the session

- `$ setsid`: create a *daemon* (or service) process
  - Leader of its own session
  - Detached from terminals (on I/O, no job control from terminals)
  - Often associated with periodic (`crond`) or network service (`inetd`) tasks

Network Service Daemons

Internet "Super-Server"

- `inetd`, initiated at boot time
- Listen on specific ports — listed in `/etc/services`
  - Each configuration line follows the format:
    service_name port/protocol ⌈*aliases ...*⌋
    E.g., `ftp 21/tcp`
- Dispatch the work to predefined daemons — see `/etc/inetd.conf` — when receiving incomming connections on those ports
  - Each configuration line follows the format:
    service_name socket_type protocol flags user_name daemon_path arguments
    E.g., `ftp stream tcp nowait root /usr/bin/ftpd`