Signals: Interprocess Communication

Operating Systems I

Viktor Iakovlev (Victor Yacovlev)

The Process Termination

- By process itself:
  - return from **`main`**
  - call function **`exit`**
  - system call **`_exit`**

- Suitable for time-limited programs that have begin and end

The Process Termination

- Killing by signal:
  - command **`kill`**
  - command **`killall`**
  - launched by **`timeout`**
  - keys Ctrl+C
  - terminal tab closed
  - system shutdown

- Asynchronous events to be processed by application

The Process Termination

- Caused by error:
  - segnemtation fault
  - broken pipe
  - division by zero
  - illegal instruction
  - assertion failed

- Default action is to to kill process by sending a **signal**

Signal

- Asynchronous event sent by:
  - the kernel
  - or by some other process

- Event handling:
  - just ignore it
  - terminate process
  - change state: s**T**opped | **R**unning
  - custom handling by application

Signals

Number	Name	Default Action	Description
1	`SIGHUP`	Term	lost terminal connection
2	`SIGINT`	Term	Ctrl+C
3	`SIGQUIT`	Core	Ctrl+\
4	`SIGILL`	Core	illegal instruction
6	`SIGABRT`	Core	abort()

man 7 signal

Демонстрации: 1) посылка сигналов; 2) do_abort.c

Core Dump

- Process Memory Dump
- May be used for debugging purposes
- Managed by systemd in modern systems

```
/usr/sbin/sysctl kernel.core_pattern
ulimit -c
```

Демонстрация: corefile для do_abort.c

Number	Name	Default Action	Description
9	`SIGKILL`	Term	process kill
11	`SIGSEGV`	Core	memory error
13	`SIGPIPE`	Term	broken pipe
15	`SIGTERM`	Term	process termination request
17	`SIGCHILD`	Ign	child terminated
18	`SIGCONT`	Cont	command fg
19	`SIGSTOP`	Stop	Ctrl+Z
23	`SIGURG`	Ign	socket urgent data

Демонстрация: Ctrl+Z

Signals Handling [deprecated]

```
#include <signal.h>

// sighandler_t - Linux only
typedef void (*sighandler_t)(int);


sighandler_t
signal(int signum, sighandler_t handler);
```

Демонстрация: handle-sigint-sigterm.c

Standards ans Systems

- System-V (Solaris)
- BSD and Linux

gcc -std=c99 *v.s.* gcc -std=gnu99

~~#define _BSD_SOURCE~~

#define _DEFAULT_SOURCE

#define _GNU_SOURCE

Signals Handling [deprecated]

```
#include <signal.h>

// sighandler_t - Linux only
typedef void (*sighandler_t)(int);

// sig_t - BSD only
typedef void (*sig_t)(int);


sighandler_t
signal(int signum, sighandler_t handler);
```

Демонстрация: handle-sigint-sigterm.c

Signals Handling [modern-way]

```
#include <signal.h>

struct sigaction {
   void      (*sa_handler)(int);
   void      (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t sa_mask;
   int       sa_flags;
   void      (*sa_restorer)(void);
};

int
sigaction(int signum,
          const struct sigaction *act,
          struct sigaction *oldact /* might be NULL */);
```

Демонстрация: sigaction-handling.c

System call **signal**

- To be used for default signal handlers: `SIG_IGN (=1)` и `SIG_DFL (=0)`
- For Linux systems works mostly like System-V

Signal Handler

- Might be called at **arbitary time**:
  - uses the current stack
  - it is warranted for x86_64 to have a «RedZone» by size of 128 bytes
  - Might use only restricted set of functions: Async-Signal Safe (AS-Safe)

Async Signal Safety

- Functions safeties:
  - Unsafe
  - Multi-Threading Safe (MT-Safe)
  - Async-Signal Safe (AS-Safe)
- The sets of AS-Safe and MT-Safe are not equal! Example: **`fwrite`**
- See for a full list in POSIX:
  `man 7 signal-safety`

Signal Handler

- The extra integer type **`sig_atomic_t`**
  - implemented as **`int`** for most platforms
  - guaranteed to be atomic at signal-level, but not MT-level
- The **`volatile`** language keyword
  - tells to compiler not to optimize the variable

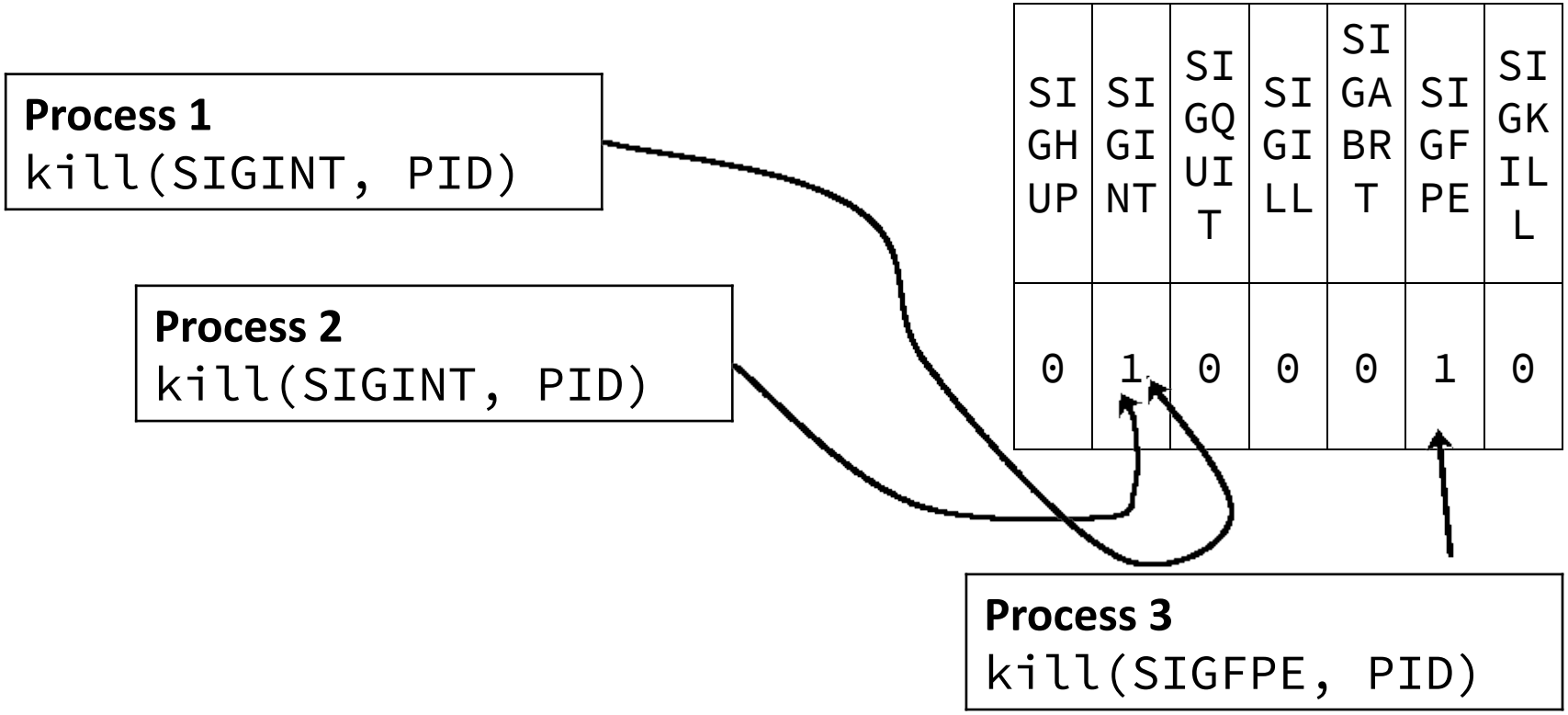
Демонстрация: good-signal-handling.c

Examples: what to handle

- SIGTERM and SIGINT - terminate correctly
- SIGINT might ask user if he sure to terminate
- SIGHUP - reload settings file(s) if any
- SIGCHILD - read child exit status
- SIGSEGV - might try to save critical data

Pending Signals

- The process attribute
- Not to be inherited by childs while `fork`



Signal Mask

Signals to be processed →	0	0	0	0	0	1	0
Mask of allowed/blocked signals →	1	0	1	1	1	1	1
	SIGHUP	SIGINT	SIGQUIT	SIGILL	SIGABRT	SIGFPE	SIGKILL
	0	1	0	0	0	1	0

Process 1
`kill(SIGINT, PID)`

Process 2
`kill(SIGINT, PID)`

Process 3
`kill(SIGFPE, PID)`

Signal Mask,
opposite to pending signals,
to be inherited by childs while fork

POSIX Signals

- Have standard behaviour for most UNIX-like variants
- Processes are noticed by signal facts, but not their count
- Signals might be processed in random order

Extented Signals

POSIX.1b real-time extensions; have been implemented in Linux

- Values from `SIGRTMIN` to `SIGRTMAX`
- Might be used as regular UNIX-signals (by system call `kill`)
- Might be queued and counted in specific order
- Might carry some trivial data

Sending Signals Using Queue

POSIX:

```
sigqueue(int pid,
         int signum,
         const union sigval value)
union sigval {
  int   sival_int;
  void* sival_ptr;
};
```

LINUX:

```
sys_rt_sigqueueinfo(
   int pid,
   int signum,
   const siginfo_t
         *uinfo
)
```