Inter Process Communication (IPC)

Giorgio Richelli
giorgio.richelli@uniroma1.it

Introduction

Processes and their address spaces are kept isolated by OS

The purposes of IPC:

- Data transfer
- Sharing data
- Event notification
- Resource sharing
- Process control

Giorgio Richelli
giorgio.richelli@uniroma1.it

Introduction

IPC primitives:

- Signals
- Pipes, Named Pipes, FIFO
- SYSV: Shared Memory, Semaphores, Message Queues
- BSD: Sockets

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signals

Signal:

- A way to start a procedure (handler) when some events occur.

Generation:

- By the kernel, when the event occurs

Delivery:

- When the process recognizes the signal's arrival (handling)

Pending

- Between generation and delivery

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signals

Signals are identified by their number

- Could be different in different system or versions
  - POSIX defines 16 different signals

Signal	Portable number	Default action	Description
SIGABRT	6	Terminate (core dump)	Process abort signal
SIGALRM	14	Terminate	Alarm clock
SIGBUS	N/A	Terminate (core dump)	Access to an undefined portion of a memory object
SIGCHLD	N/A	Ignore	Child process terminated, stopped, or continued
SIGCONT	N/A	Continue	Continue executing, if stopped
SIGFPE	8	Terminate (core dump)	Erroneous arithmetic operation
SIGHUP	1	Terminate	Hangup
SIGILL	4	Terminate (core dump)	Illegal instruction
SIGINT	2	Terminate	Terminal interrupt signal
SIGKILL	9	Terminate	Kill (cannot be caught or ignored)
SIGPIPE	13	Terminate	Write on a pipe with no one to read it
SIGPOLL	N/A	Terminate	Pollable event
SIGPROF	N/A	Terminate	Profiling timer expired
SIGQUIT	3	Terminate (core dump)	Terminal quit signal
SIGSEGV	11	Terminate (core dump)	Invalid memory reference
SIGSTOP	N/A	Stop	Stop executing (cannot be caught or ignored)
SIGSYS	N/A	Terminate (core dump)	Bad system call
SIGTERM	15	Terminate	Termination signal
SIGTRAP	5	Terminate (core dump)	Trace/breakpoint trap
SIGTSTP	N/A	Stop	Terminal stop signal
SIGTTIN	N/A	Stop	Background process attempting read
SIGTTOU	N/A	Stop	Background process attempting write
SIGUSR1	N/A	Terminate	User-defined signal 1
SIGUSR2	N/A	Terminate	User-defined signal 2
SIGURG	N/A	Ignore	High bandwidth data is available at a socket
SIGVTALRM	N/A	Terminate	Virtual timer expired
SIGXCPU	N/A	Terminate (core dump)	CPU time limit exceeded
SIGXFSZ	N/A	Terminate (core dump)	File size limit exceeded
SIGWINCH	N/A	Ignore	Terminal window size changed

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signals

- Linux has 64 signals

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Handling

- Default actions:
  - What happens when the signal is delivered to a process
  - Each signal has one
- Possible values:
  - Abort: Terminate the process after generating a core dump.
  - Exit: Terminate the process without generating a core dump.
  - Ignore: Ignores the signal.
  - Stop: Suspend the process.
  - Continue: Resume the process, if suspended
- Default actions may be overridden by <signal handlers>

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Handling

A kernel routine, `issig()`, checks for pending signals

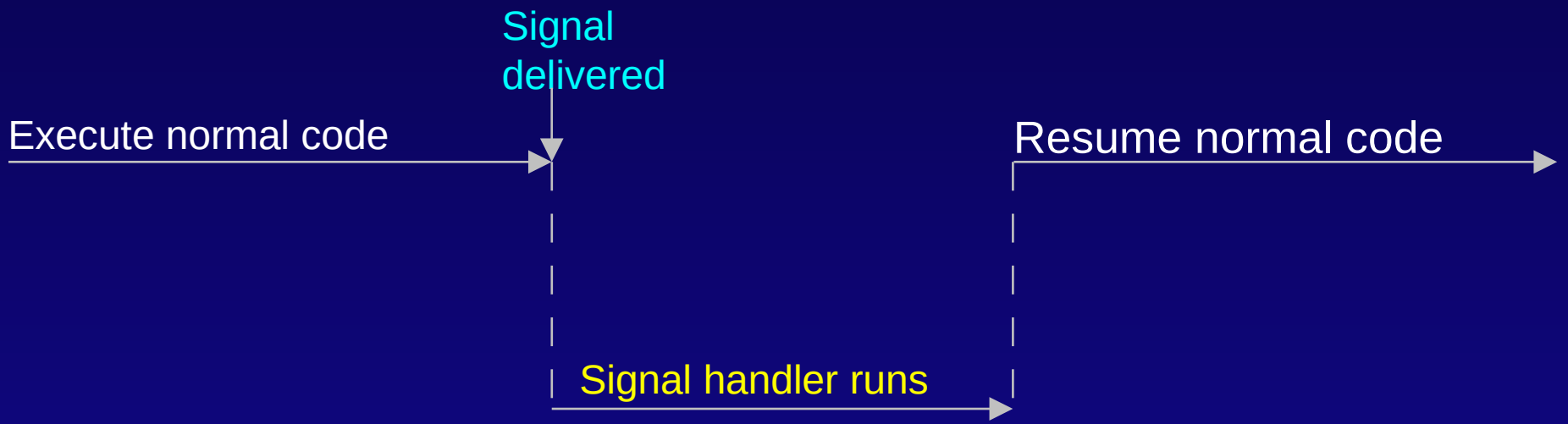
- Before returning to user mode from a system call or interrupt
- Just before blocking on an interruptible event
- Immediately after waking up from an interruptible event

Then:

- `psig()`: checks the signal action and if not ignored, defaulted, etc, calls:
- `sendsig()`: which invokes the user-defined handler

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Handling



Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Generation

Signal sources:

- Exceptions
- Terminal interrupts
- Job control
- Quotas
- Notifications
- Alarms
- Other processes

Giorgio Richelli
giorgio.richelli@uniroma1.it

Sleep and signals

Interruptible sleep:

- waiting for an event with indefinite time
- signals can be delivered

Uninterruptible sleep:

- waiting for a short term event such as disk I/O
- pending the signal
- Recognizing it before returning to user mode or blocking on another event

```
if (issig()) psig();
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Handlers

Signals handlers are installed by:

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

(now overridden by sigaction)

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signal Handlers

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act,struct
  sigaction *oldact);

struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t   sa_mask;
  int        sa_flags;
  void     (*sa_restorer)(void); /* Don't use this */
};
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Original Implementation: Unreliable Signals

- Signal handlers were not persistent and do not mask recurring instances of the same signal (SVR2)
- Race conditions: two ^C.
- Performance: SIG_DFL, SIG_IGN:
  - Kernel did not know the content of `u_signal[];`
  - Awake, check, and perhaps go back to sleep again (waste of time).

Giorgio Richelli
giorgio.richelli@uniroma1.it

Reinstalling a signal handler

```
void sigint_handler(int sig)
{
    signal(SIGINT, sigint_handler); /* first instruction */
  …
}
main()
{
    signal(SIGINT, sigint_handler);
    …
}
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Unreliable Signals

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

int cnt=0;
void handler(int sig)
{
    cnt++;
    printf("In the handler...\n");
    signal(SIGINT,handler);
}
main()
{
    signal(SIGINT,handler);
    while (1) {
        printf("In main\n");
        sleep(1);
    }
}
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Nowadays: Reliable Signals

Primary features:

- Persistent handlers: need not to be re-installed.
- Masking: A signal can be temporarily masked (it will be delivered later)
- Sleeping processes: let the signal disposition info visible to the kernel (now it is kept in the *proc* structure)
- Unblock and wait:
  - `sigpause()/sigsuspend()`
  - automatically unmask and suspend the process

Giorgio Richelli
giorgio.richelli@uniroma1.it

Signals in SVR4

- `sigprocmask(how, setp, osetp)`
  - `SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK`
- `sigaltstack(stack, old_stack):`
  - Specify a new stack to handle the signal
- `sigsuspend(sigmask)`
  - Set the blocked signals mask to *sigmask* and puts the process to sleep
- `sigpending(setp)`
  - *setp* contains the set of signals pending to the process

Giorgio Richelli
giorgio_richelli@it.ibm.com

Signals in SVR4

- `sigsendset(procset, sig)`
  - Sends the signal sig to the set of processes procset
- `sigaction(signo, act, oact)`
  - Specify a handler for signal signo.
  - act, oact pointers to sigaction structure
  - oact is the previous sigaction data
- Compatibility interface:
  - `signal, sigset, sighold, sigrelse, sigignore, sigpause`

Giorgio Richelli
giorgio_richelli@it.ibm.com

Signal flags

- SA_NOCLDSTOP: Do not generate SIGCHLD when a child is suspended
- SA_RESTART: Restart system call automatically if interrupted by this signal
- SA_ONSTACK: Handle this signal on the alternate stack, if one has been specified by sigaltstack
- SA_NOCLDWAIT: sleep until all terminate
- SA_SIGINFO: additional info to the handler.
- SA_NODEFER: do not block this signal
- SA_RESETHAND: reset the action to default

Giorgio Richelli
giorgio_richelli@it.ibm.com

Reliable Signals

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

static void hdl (int sig, siginfo_t *siginfo, void *context)
{
  printf("Signal %d From PID: %ld, UID: %ld\n",sig,siginfo->si_pid,siginfo->si_uid);
}

int main (int argc, char *argv[])
{
  struct sigaction act;
  memset (&act, 0, sizeof(act));
  act.sa_sigaction = &hdl;
  act.sa_flags = SA_SIGINFO;

  if (sigaction(SIGTERM, &act, NULL) < 0) {
    perror ("sigaction");
    return 1;
  }
  while (1) sleep (10);
  return 0;
}
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

Other IPC Facilities

- Signals are not enough
- Still useful for some purposes:
  - kill
  - User interaction (ctrl-c, ctrl-z) ..
- But :
  - <expensive> (timewise) and
  - Limited: only 32/64

Giorgio Richelli
giorgio.richelli@uniroma1.it

Other IPC Facilities

- Unidirectional Pipes
- FIFOs (named pipes)
- System V IPC
- Sockets
- ..

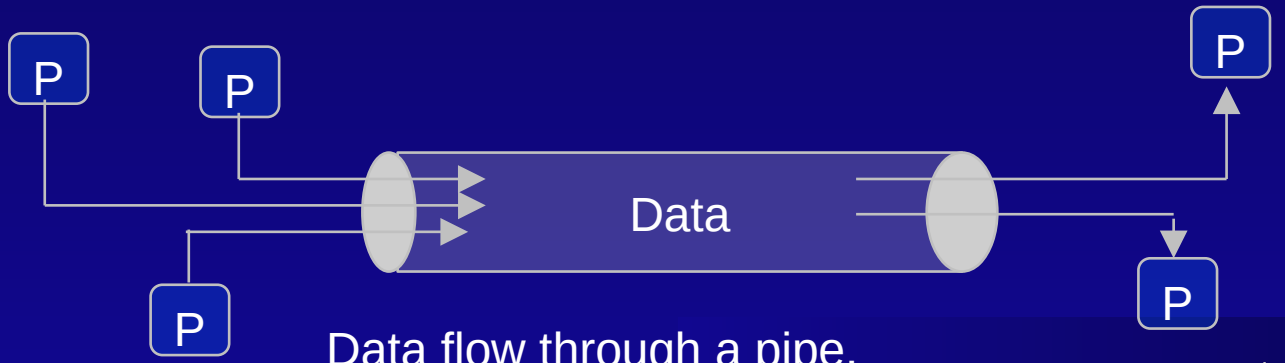
Giorgio Richelli
giorgio.richelli@uniroma1.it

Pipes

It is an:

- Unidirectional
- FIFO
- Unstructured data stream

```
int pipe (int filedes[2])
```



Data flow through a pipe.

Giorgio Richelli
giorgio.richelli@uniroma1.it

Pipes

- `*filedes` is an array of two file descriptors
- Using a pipe:
  - Write to `filedes[1]`
  - Read from `filedes[0]`

Giorgio Richelli
giorgio.richelli@uniroma1.it

Pipes

- Writing to a pipe would block for large I/O sizes
  - Limited bufferspace
  - Value is system dependent
- Un-named pipes can only used between “relatives”
  - using the inheritance mechanism available with `fork()/exec()`
- Often used in combination with `dup()/dup2()`

Giorgio Richelli
giorgio.richelli@uniroma1.it

Named Pipes

- Also called 'FIFO's
- Identified by their access point: path/filename

- Persistent
- Have a filesystem inode

- Created by:
  - `mkfifo` command
  - `int mkfifo(char *path, mode_t mode);`

Giorgio Richelli
giorgio.richelli@uniroma1.it

Named Pipes

- Cannot be opened for >both< reading and writing (in the same process)
- `read()` and `write()` to a named pipe are blocking,by default
- Seek operations (`lseek()`) cannot be performed

Giorgio Richelli
giorgio.richelli@uniroma1.it