Select and poll and Signals

CS 241

April 6, 2012

University of Illinois

Review: Interprocess communication

Shared address space

- Shared memory
- Memory mapped files

Via OS

- Files
- Pipes
- FIFOs (named pipes): Review today
- Signals: New today

SurveyMonkey

Review: FIFOs and dup()

How could we read from a FIFO as if it were stdin?

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char** argv) {
    mkfifo(argv[1], S_IRWXU | S_IRWXG | S_IRWXO);

    int fifo = open(argv[1], O_RDONLY);

    dup2(fifo, 0); /* 0 is the file descriptor of stdin */

    char line[1024];
    while (fgets(line, 1024, stdin))
        printf("I got this: %s\n", line);
}
```

pipestdin.c

Select & Poll

Waiting for *any one* of a set of inputs

Examples

- Multiple children to compute in parallel; wait for output from any
- Network server connected to many clients; take action as soon as any one of them sends data

Problem

- Can use read / write scanf, but ..... problem?
- Blocks waiting for that one file, even if another has data ready & waiting!

Solution

- Need a way to wait for any one of a set of events to happen
- Something similar to wait() to wait for any child to finish, but for events on file descriptors

Select and Poll: Waiting for input

Similar parameters

- Set of file descriptors
- Set of events for each descriptor
- Timeout length

Similar return value

- Set of file descriptors
- Events for each descriptor

Notes

- Select is slightly simpler
- Poll supports waiting for more event types
- Newer variant available on some systems: epoll

Select

```
int select (int num_fds, fd_set* read_set,
            fd_set* write_set, fd_set* except_set,
            struct timeval* timeout);
```

Wait for readable/writable file descriptors.

Return:

- Number of descriptors ready
- -1 on error, sets **`errno`**

Parameters:

- **`num_fds`**:
  - number of file descriptors to check, numbered from 0
- **`read_set`**, **`write_set`**, **`except_set`**:
  - Sets (bit vectors) of file descriptors to check for the specific condition
- **`timeout`**:
  - Time to wait for a descriptor to become ready

File Descriptor Sets

Bit vectors

- Often 1024 bits, only first **num_fds** checked
- Macros to create and check sets

```
fds_set myset;
void FD_ZERO(&myset);      /* clear all bits   */
void FD_SET(n, &myset);    /* set bits n to 1 */
void FD_CLEAR(n, &myset); /* clear bit n      */
int FD_ISSET(n, &myset);  /* is bit n set?    */
```

File Descriptor Sets

Three conditions to check for

- Readable
  - Data available for reading
- Writable
  - Buffer space available for writing
- Exception
  - Out-of-band data available (TCP)

Select: Example

```
fd_set my_read;
FD_ZERO(&my_read);
FD_SET(0, &my_read);

if (select(1, &my_read, NULL, NULL) == 1) {
      ASSERT(FD_ISSET(0, &my_read);
      /* data ready on stdin */
```

Poll

```
#include <poll.h>
```

```
int poll (struct pollfd* pfds, nfds_t nfds, int timeout);
```

Poll file descriptors for events.

Return:

- Number of descriptors with events
- -1 on error, sets **`errno`**

Parameters:

- **`pfds`**:
  - An array of descriptor structures. File descriptors, desired events and returned events
- **`nfds`**:
  - Length of the **`pfds`** array
- **`timeout`**:
  - Timeout value in milliseconds

Descriptors

Structure

```
struct pollfd {
    int fd;                         /* file descriptor */
    short events;                   /* queried event bit mask */
    short revents;                  /* returned event mask */
```

Note:

- Any structure with `fd` < 0 is skipped

Event Flags

`POLLIN`:

- data available for reading

`POLLOUT`:

- Buffer space available for writing

`POLLERR`:

- Descriptor has error to report

`POLLHUP`:

- Descriptor hung up (connection closed)

`POLLVAL`:

- Descriptor invalid

Poll: Example

```
struct pollfd my_pfds[1];

my_pfds[0].fd = 0;
my_pfds[0].events = POLLIN;

if (poll(&my_pfds, 1, INFTIM) == 1) {
      ASSERT (my_pfds[0].revents & POLLIN);
      /* data ready on stdin */
```

Signals

Signals

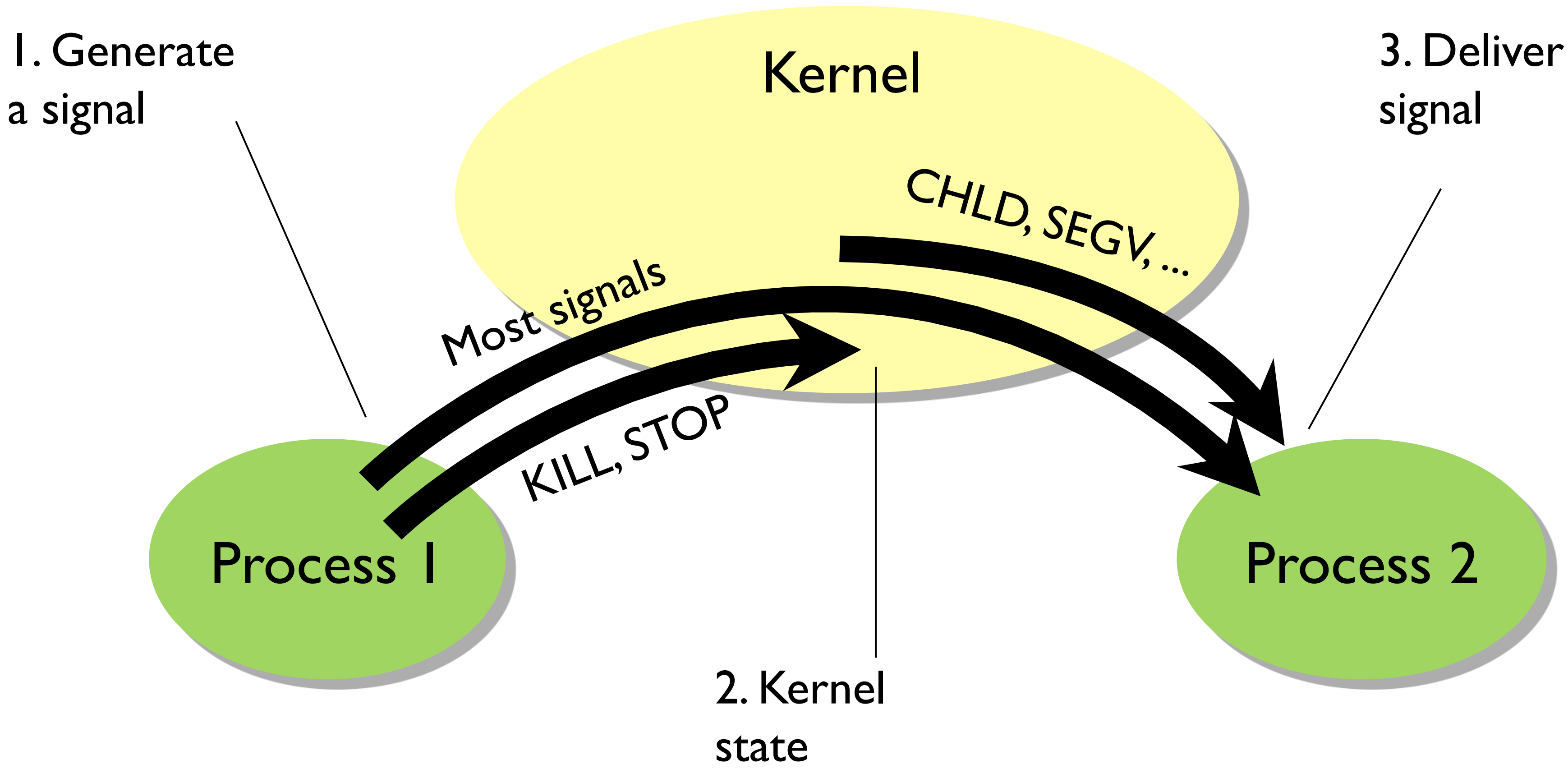
A signal is an asynchronous notification of an event

- Asynchronous: could occur at any time
- Interrupts receiving process; jumps to signal handler in that process
- A (limited) menu of event types to pick from

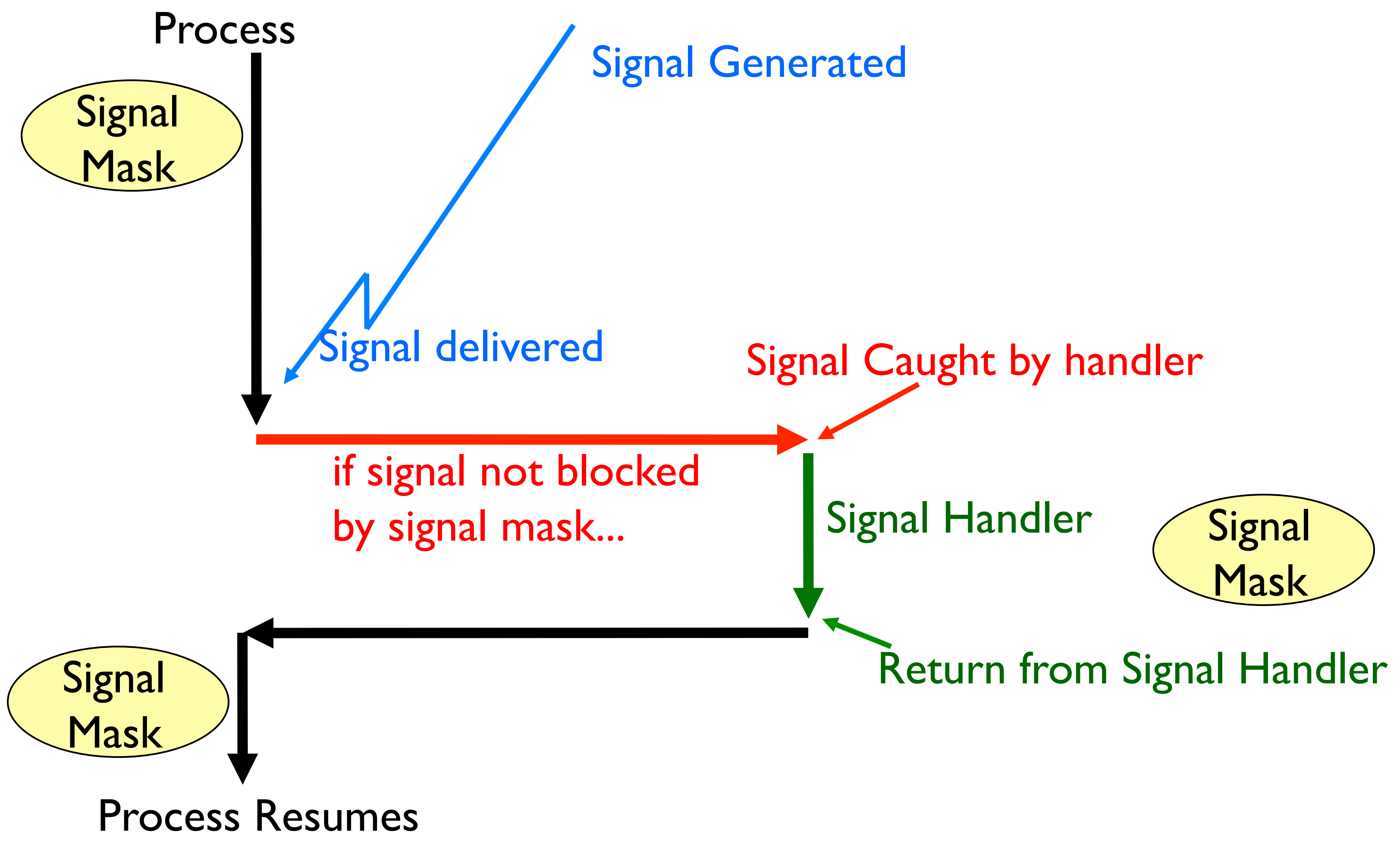
What events could be asynchronous?

- Email message arrives on my machine
  - Mailing agent (user) process should retrieve it
- Invalid memory access
  - OS should inform scheduler to remove process from the processor
- Alarm clock goes off
  - Process which sets the alarm should catch it

Signaling overview



Signaling: Inside Process 2



Example: Catch SIGINT

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

void handle(int sig) {
    char handmsg[] = "Ha! Handled!!!\n";
    int msglen = sizeof(handmsg);
    write(2, handmsg, msglen);
}

int main(int argc, char** argv) {
    struct sigaction sa;
    sa.sa_handler = handle;   /* the handler function!! */
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);

    sigaction(SIGINT, &sa, NULL);

    while (1) {
        printf("Fish.\n");
        sleep(1);
    }
}
```

Note: Need to check for error conditions in all these system & library calls!

Run Demo

Some POSIX signals (see signal.h)

NAME	Default Action	Description
SIGHUP	terminate process	terminal line hangup
SIGINT	terminate process	interrupt program
SIGQUIT	create core image	quit program
SIGILL	create core image	illegal instruction
SIGTRAP	create core image	trace trap
SIGABRT	create core image	abort(3) call (formerly SIGIOT)
SIGEMT	create core image	emulate instruction executed
SIGFPE	create core image	floating-point exception
SIGKILL	terminate process	kill program
SIGBUS	create core image	bus error
SIGSEGV	create core image	segmentation violation
SIGSYS	create core image	non-existent system call invoked
SIGPIPE	terminate process	write on a pipe with no reader
SIGALRM	terminate process	real-time timer expired
SIGTERM	terminate process	software termination signal
SIGURG	discard signal	urgent condition present on socket
SIGSTOP	stop process	stop (cannot be caught or ignored)
SIGTSTP	stop process	stop signal generated from keyboard
SIGCONT	discard signal	continue after stop

Some POSIX signals (see signal.h)

NAME	Default Action	Description
SIGCHLD	discard signal	child status has changed
SIGTTIN	stop process	background read attempted
SIGTTOU	stop process	background write attempted
SIGIO	discard signal	I/O is possible on a descriptor
SIGXCPU	terminate process	cpu time limit exceeded
SIGXFSZ	terminate process	file size limit exceeded
SIGVTALRM	terminate process	virtual time alarm
SIGPROF	terminate process	profiling timer alarm
SIGWINCH	discard signal	Window size change
SIGINFO	discard signal	status request from keyboard
SIGUSR1	terminate process	User defined signal 1
SIGUSR2	terminate process	User defined signal 2
SIGWAKE	start process	Wake upon reaching end of long, boring list of signals

A little puzzle

Signals are a kind of interprocess communication

Q: Difference between signals and pipes or shared memory?

A:

- Asynchronous notification
- Doesn’t send a “message” as such; just a signal number
- Puzzle: Then how could I do *this.....?*

Run demo