



Signal Handling

Lecture 15

Klara Nahrstedt

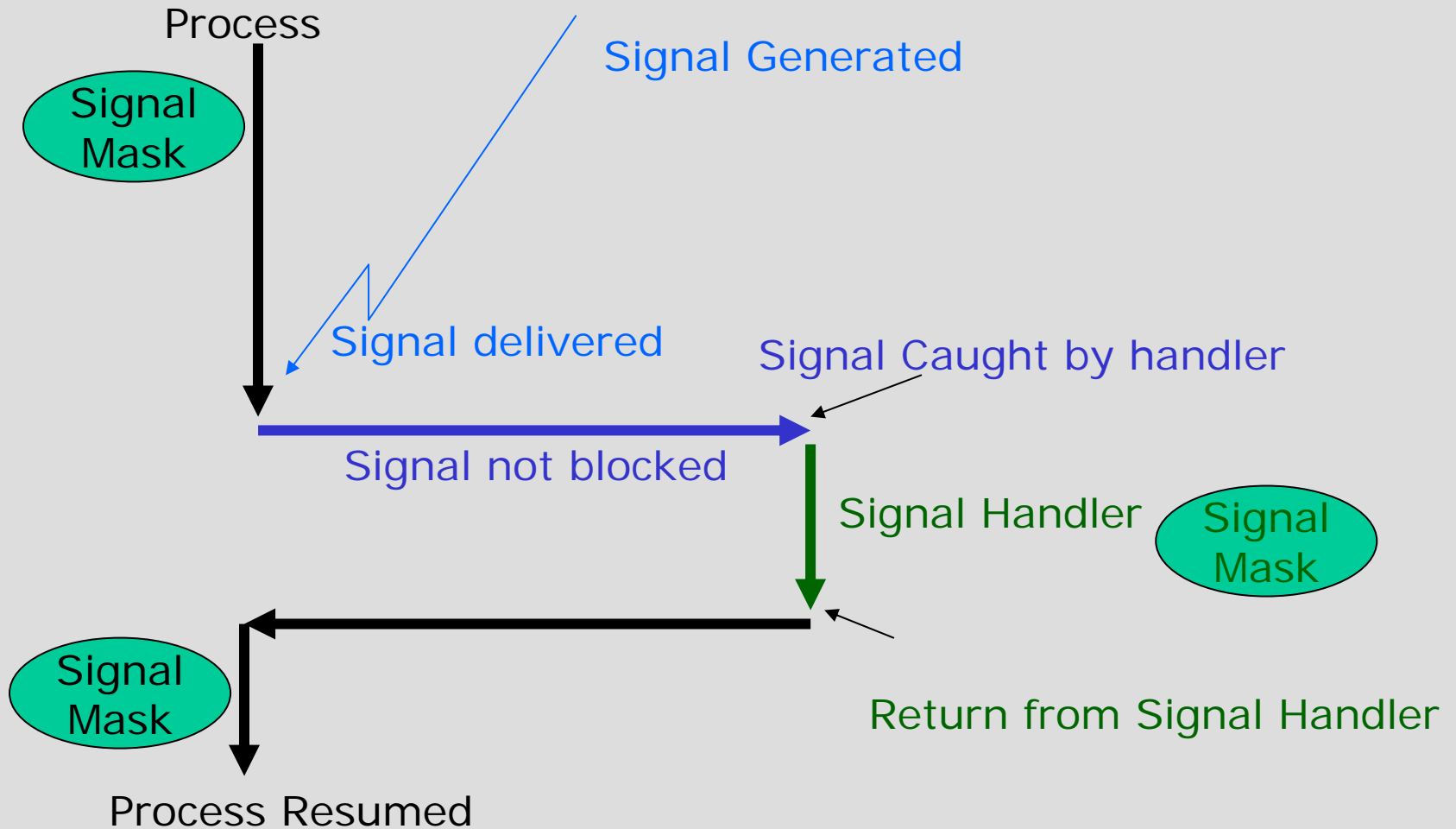
CS241 Administrative

- Read Chapter 8.1-8.4 in R&R

Outline

- Learn fundamentals of signal handling
- Experiment with signals for control
- Explore POSIX signal facilities
- Use signal masks and handlers
- Signals and threads

Review: How Signals Work



Review: Generating Signals

- Signal has a symbolic name starting with SIG
- Signal names are defined in signal.h

Users can generate signals (e.g., SIGUSR1)

OS generates signals when certain errors occur
(e.g., SIGSEGV – invalid memory reference)

Specific calls generate signals such as alarm
(e.g., SIGALARM)

Examples: Programming Signals

From a program you can use the `kill` system call:

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

Example 8.4: send SIGUSR1 to process 3423:

```
if (kill(3423, SIGUSR1) == -1)
    perror("Failed to send the SIGUSR1 signal");
```

Example 8.5: a child kills its parent:

```
if (kill(getppid(), SIGTERM) == -1)
    perror("Failed to kill parent");
```

Programming Signals

Example 8.5: a process sends a signal to itself:

```
if (raise(SIGUSR1) != 0)  
    perror("Failed to raise SIGUSR1");
```

Example 8.8: kill an infinite loop after 10 seconds:

```
int main(void) {  
    alarm(10);  
    for ( ; ; ) ;  
}
```

Signal Masks

- Process can temporarily prevent signal from being delivered by *blocking* it.
- *Signal Mask* contains a set of signals currently blocked.
- **Important!** Blocking a signal is different from ignoring signal. Why?
- When a process blocks a signal, the OS does not deliver signal until the process unblocks the signal
 - A *blocked* signal is not delivered to a process until it is unblocked.
- When a process ignores signal, signal is delivered and the process handles it by throwing it away. 8

Signal Sets

- ❑ Signal set is of type `sigset_t`
- ❑ Signal sets are manipulated by five functions:

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

```
int sigemptyset(sigset_t *set);
```

```
int sigfillset(sigset_t *set);
```

```
int sigaddset(sigset_t *set, int signo);
```

```
int sigdelset(sigset_t *set, int signo);
```

```
int sigismember(const sigset_t *set, int  
    signo);
```

Signal Sets

`sigemptyset` initializes the set to contain no signals

`sigfillset` puts all signals in the set

`sigaddset` adds one signal to the set

`sigdelset` removes one signal from the set

`sigismember` tests to see if a signal is in the set

Signal Masks

SigMask

SigInt	SigQuit	SigKill	...	SigCont	SigAbrt
0	0	1	...	1	0

Signal SigInt Bit 2, Signal Sigkill Bit 9,
Signal SigChld Bit 20

A SIGSET is a collection of signals:
#000003 is SIGHUP + SIGINT

SIGPROCMASK

- The function **sigprocmask** is used to modify the signal mask.

```
#include <signal.h>  
int sigprocmask(int how,  
    const sigset_t *restrict set,  
    sigset_t *restrict oset)
```

'how' specifies the manner in which the signal mask is to be modified

SIG_BLOCK – add collection of signals to those currently blocked

SIG_UNBLOCKED – delete collection of signals from the currently blocked

SIG_SETMASK – set collection of signals being blocked to the specified set

Example: Initialize Signal Set:

```
if ((sigemptyset(&twosigs) == -1) ||  
    (sigaddset(&twosigs, SIGINT) == -1) ||  
    (sigaddset(&twosigs, SIGQUIT) == -1))  
    perror("Failed to set up signal mask");
```

Example: Add SIGINT to Set of Blocked Signals

```
sigset_t newsigset;
```

```
if ((sigemptyset(&newsigset) == -1) ||  
    (sigaddset(&newsigset, SIGINT) == -1))  
    perror("Failed to initialize the signal  
set");  
else if (sigprocmask(SIG_BLOCK, &newsigset,  
NULL) == -1)  
    perror("Failed to block SIGINT");
```

If SIGINT is already blocked, the call to sigprocmask has no effect.

Summary

- Signals – asynchronous events
- Generating signals
- Programming signals
- Signal masks