



CS 241 Discussion Section (11/10/11)

Contents

- Signals
- MP7 - HTTP Headers

Review: signals

Asynchronous notification to a process
indicating some action should be taken

Sending signals to a process:

```
kill -<signal> <pid>  
int kill(pid_t pid, int sig);
```

We can signal individual threads, too:

```
int pthread_kill(pthread_t tid, int sig);
```

What can we do with signals?

- Handle them!
 - Default or Ignore
 - Custom function with sigaction
- Block them!
 - Delay delivery with masks
 - Then we can sigwait to get them.

Lots of signal 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);
int sigprocmask(int how, const sigset_t *restrict set, sigset_t
    *restrict oset)
int sigaction(int signo, const struct sigaction *act, struct
    sigaction *oact);
int sigwait(const sigset_t *restrict sigmask, int *restrict
    signo);
```

Process Signal Masks

Setting SIGINT to be blocked

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

SIG_BLOCK adds set to current mask
oldset will store the previous signal mask

Thread Signal Masks

`pthread_sigmask()`:

Takes same parameters as `sigprocmask`

Only affects the signal mask of a single thread

Signal mask is inherited on thread creation

Signal Handlers

Allow us to change what happens when a signal is received

```
void handler(int signo) { ... }
struct sigaction act;

act.sa_flags = 0;
act.sa_handler = handler;
// additional signals blocked in the handler
sigemptyset(&act.sa_mask);
sigaction(SIGUSR1, &act, NULL);
```

sa_handler can also be set to SIG_DFL
(default) or SIG_IGN (ignore)

`sa_handler` vs. `sa_sigaction`

We can get additional information about the signal

```
void handler(int signo, siginfo_t* info,  
             void* context);  
act.sa_flags = SA_SIGINFO;  
// fill sa_sigaction instead of sa_handler  
act.sa_sigaction = handler;
```

Extra information contains, e.g., the source of the signal (`info->si_code`):

`SI_USER` – user-created signal (with `abort`, `kill`, etc.)
`SI_TIMER` – a POSIX:RTS timer expired
etc.

One function to rule them all

```
void ( *signal(int signum, void  
(*handler)(int)) ) (int);
```

But what does it all mean?

One function to rule them all

Symbol

```
void ( *signal(int signum, void  
(*handler)(int)) ) (int);
```

But what does it all mean?

One function to rule them all

```
void ( *signal(int signum, void  
(*handler)(int)) ) (int);
```

But what does it all mean?

One function to rule them all

```
void ( *signal(int signum, void  
(*handler)(int)) ) (int);  Return Type
```

But what does it all mean?

One function to rule them all

```
void ( *signal(int signum, void  
(*handler)(int)) ) (int);
```

But what does it all mean?

```
typedef void (*sighandler_t)(int);
```

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

Code Examples

ds12/signals.c

ds12/http.c