Signal Masks

- Sometimes we need to block some signals, so that critical sections are not interrupted.
- Every process maintains a signal mask telling which signals are blocked.
- If a signal type is blocked, and signals of this type are received, they are suspended until process termination or until the signal type is unblocked.
- Signal masks are stored in the data type `sigset_t`. 
#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);
return 0 if OK, -1 on error
int sigismember(const sigset_t *set, int signo);
returns 1 if true, 0 if false

• The above are used to set the set value, not to set the process signal mask.
• Call sigemptyset() or sigfillset() at least once.
• sigset_t is guaranteed to be able to hold all signals supported by the UNIX implementation.
Signal Masks (cont)

```c
#include <signal.h>
int sigpending(sigset_t *set);
    returns 0 if OK, -1 on error
```

- `sigpending()` tells us what signals are blocked and currently pending.
- The list of signals is returned inside `set`.
- Use `sigismember()` to find out what signals are present in `set`. 
Masking Signals

#include <signal.h>
int sigprocmask(int how, const sigset_t *set,
                 sigset_t *oset);
returns 0 if OK, -1 on error

**how values**

- SIG_BLOCK (union)
- SIG_UNBLOCK (intersection)
- SIG_SETMASK (equality)

- If *oset* is non-NULL, the old signal mask is returned in it.
- *set* defines the signals we want to block or unblock.
- If there are any pending signals, and we unblock it with `sigprocmask()`, one of these signals is received before `sigprocmask()` returns.
Critical Sections

Setup signal mask
Call `sigprocmask()` to block signals
/* critical section */
Call `sigprocmask()` to unblock signals
Signals will be handled, etc.

- Blocking signals makes sure that critical section are executed atomically.
- Yet what if we want to wait for a signal after unblocking the signal mask.
  - Calling `pause()` could make process wait forever!!
**sigsuspend()**

```c
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
returns −1 with errno set to EINTR
```

- **sigsuspend() execution:**
  1. Sets the signal mask to `sigmask`.
  2. Then it calls the `pause()` function.
  3. If `pause()` returns, the signal mask to set back to its original value.

- All the above steps are guaranteed to be performed atomically and we thus get no lost signals.
#include <signal.h>
int sigaction(int signo, const struct sigaction *act, const struct sigaction *oset);
returns −1 on error

struct sigaction {
  void (*sa_handler)();
  sigset_t sa_mask;
  int sa_flags;
};

Values for sa_flags

- SA_NOCLDSTOP
- SA_RESTART
- SA_ONSTACK
- SA_NODEFER
- SA_RESETHAND
- SA_SIGINFO

- A more modern version of signal().
- sa_mask specify the additional signals to block if the sa_handler is a user defined signal handler.
- Not all sa_flags values are implemented.
Exercises

• Protect a section of your program from being interrupted by signals.
• Disable the CTRL-C keyboard termination signal for a critical section.
• Send signals to terminate children processes and reap their termination status. See that no signal is lost.
• Re-implement sleep() using sigsuspend().