Overview

- *Administrative*
  - HW 1 grades
  - HW 2 Due
- *Topics*
  - 5.1 What is a Signal?
  - 5.2-3 Dealing with Signals - masks, handlers
  - 5.4 Synchronization: pause(), sigsuspend()
  - 5.6 Interaction with other systems calls
  - problems with signals for communications
  - 5.7-9 Rest of signals
- *Readings: Ch. 5. (p. 167-204)*

What is a Signal?

- *Motivation*

  * Get immediatereal-time attention, ˆC and infinite loop

  * Increase concurrency, e.g. disk controller, CPU

  * Attend to unpredictable events, e.g. errors

  - But, Asynchronous = Hard to understand

- *Signals*

  * Software notification of an event to a process

  * Lifecycle: event

  - generate signal

  - OS queues blocked one, delivers others to the process

  - Process catches it and executes its handler

5.1 Value Domain for Signals

- *Symbolic Names for Signals (Tables 5.1-2, pp. 170-1)*
  - * Defined in signal.h, example of required ones
  - * SIGKILL - terminate
  - * SIGFPE - error in arithmetic or divide by zero
  - * SIGSEGV - invalid memory address
  - * SIGINT - interactive attention signal (e.g. ˆC)
- *Generating Signals by command line’:*
  - * ˆC (SIGINT) or ˆ| (SIGQUIT)
  - * Q? How to determine SIGINT character on your terminal?
  - * stty -a | grep intr ; stty -a | grep quit (pp 174)

    kill -s signal pid, Ex. Kill -INT 3423

    kill -l [exit_status] # list names of signals

    kill [-signal] pid

5.1 Value Domain for Signals

- *Generating signals by System calls:*

  int kill(pid_t pid, int sig);

  * (Pid > 0)

  -   Send signal to process pid

  * (Pid < 0)

  -   Send signal to process group id = |pid|

  * (Pid = 0)

  -   Send signal to process group of sender

  * Returns 0 on success

  -   Returns -1 if User-id of caller and receiver differ

- *System call:*

```
 int raise(int sig);
unsigned int alarm(unsigned int seconds); /* SIGALARM */
```

- *Example Code Segment:*

```
#include signal.h
kill( 3423, SIGKILL);
raise(SIGUSER1)
alarm(10);
for( ; ; ) { }
```

Exercises on "What is a Signal?"

- *Q? Compare signals and pipes for coomunication b/w processes P1 & P2*

  * relationship b/w P1, P2

  * bandwidth

  * possibility of blocking

  * buffer size

- *Q? Classify signals into synchronous and asynchronous ones:*

  * timer expiry, file does not exist, mouse click

  * end-of-file found, ˆC on keyboard,

- *Q? Which tasks can signals be used for?*

  * Exception handling, e.g. bad pointer, divide by 0

  * Process termination in abnormal circumtances
  - parent terminates a child process
  - a child process terminates its parent

  * Process notification of asynchronous events
  - e.g. I/O complete, timer expiration

  * Interprocess communication (message passing)

  * Emulation of multitasking

Dealing with Signals

- *What can a process do with signals?*

  * block for a while: postpone delivery

  * ignore signals as if they never arrived

  * handle signal- set up a routine to be called
  - whenever s particular signal arrives

- *Implementation of Process preferences*

  * Signal mask

  * Table mapping signal-type to handler

- *Signal Mask = list of currently blocked signals*

  * Changed by sigprocmask()

- *Signal Handler*

  * A user defined procedure or "SIG_DFL" or "SIG_IGN"

  * "SIG_IGN" will throw it away with no queueing

  * Installed via sigaction()

  * Invocation: implicit at signal delivery to process

5.2 Handling Signals - masks

- *Signal Mask = list of currently blocked signals*

  * Blocked signals are queued, i.e.not lost

- *Create a signal-mask : system calls (pp. 175)*

- return 0 if successful, -1 on error

```
int sigemptyset( sigset_t *set); /* no signal */
int sigfillset( sigset_t *set); /* all signals */
int sigaddset(sigset_t *set, int signo);
int sigdelete(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
```

- Return 1 if member, 0 otherwise

- *Example 5.8 (pp. 175)*

  * Create a mask with 2 signals

Dealing with Signals - mask

- *Change signal mask for a process:*
  - * - examine or modify signal mask
  - * - add/delete a set of signals
  - * - union of two blocked sets
- *sigprocmask(), pp. 176*
  - * Parameter 1: how, i.e. add/delete/assignment
  - * Parameter 2: new sigset_t
  - * Parameter 3: old sigset_t
- *Example 5.9 (pp. 176)*
  - * Add SIGINT to blocked set of signals
  - * Simple usage
- *Example 5.12 (pp. 178)*
  - * Typical use of blocking - protect crucial sections!
  - * signals are masked during fork()
- *Alternative: sigaction() as shown later*

Dealing with Signals - mask

- *Which is closer to maksing signals?*

  * Telephone: block calls from certain numbers

  - calls are lost

  * Post Office: place a hold on delivery for a few days

  - mail delivery is postponed but mail is not lost.

- *Masks and fork()*

  * Is fork() signal safe?

  * Does child process inherit mask of parent?

  * Does a child share mask with its parent?

  * Can a parent process change masks for its child process?

- *What can Masks be used for?*

  * Postpone signals of certain types

  * Ignore signals of specific types

  * Block signals from specific processes

5.3 Dealing with Signals - handler routines

- *Handler is a C function / subroutine*

  * Returns no value

  * Gets the signal number as input

  * Asynchronous invocation

- *Installing signal handlers: sigaction() - pp. 180*

  * Parameter 1: signal number

  * Parameter 2: new handler structure

  * Parameter 3: old handler structure

- *Handler structure (struct sigaction)*

  * Field 1: pointer to handler function
  - or SIG_DFL - default handler function
  - or SIG_IGN - ignore signal, i.e do nothing
  - Example 5.17 (pp. 182) - testing for ignored signal

  * Field 2: mask
  - additional signals to be blocked during
  - execution of the signal handler subroutine

  * Field 3: special flags (0 for now!)
  - e.g. automtic restart of system call interuppted by signal
  - in spec 1170 not in POSIX

Dealing with Signals - handler routines

- *Example 5.13 (pp. 180)*
    * Install handler for SIGINT
- *Example 5.15 (pp. 181)*

```
char message = "I found ˆC 0 ;

void catch_ctrl_c( int signo); {
        write(stderr, message, strlen(message));
}

struct sigaction act;
act.sa_handler = catch_ctrl_c;
sigemptyset(act.sa_mask);
act.sa_flags = 0;
if (sigaction(SIGINT, act, NULL)  0) {  }
```

Exercises on Dealing with Signals

- *Compare and contrast the following:*
  - * (a) Postpone signals vs. Ignore signals
  - * (b) mask set by sigprocmask() vs. mask set by sigaction()
- *Q? Is write() signal safe? (Table 5.3, pp. 191)*
  - * Is fprintf() signal safe?
  - * Why use signal safe system calls within a handler?
- *How would one simulate the following policies for*

  - for signals arriving during execution of a handler:
  - * Telephone: call waiting

    - attended to new signal immediately
  - * Telephone: disable call waiting w/ no voice mailbox

    - new signals are lost
  - * Telephone: disable call waiting + voice mailbox

    - new signals are saved for later processing
- *Compare the above policies for masking signal inside handlers.*
  - * When would you use each policy?

Handling Signals- Process synchronization

- *5.4 Waiting for a signal*

  * Motivation: recall parent-child synchronization

  * Chapter 2: exit() and wait()

  * Chapter 5.4: kill() and pause()/sigsuspend()

- *system call pause(); (pp. 182)*

  * wait till a unblocked signal comes

  * Example 5.18 (pp. 183)

  * Notice external variable signal_received

  * signal must arrive during pause() to set signal_received

  * window of vulnerability
  - b/w testing of signal_received and call to pause()

- *new system call sigsuspend(); (pp. 183)*

  * Closes window of vulnerability

  * Atomic step to unblock signal and start wait
    int sigsuspend(const sigset_t *sigmask);
    /* unblocked signals (change mask) and wait for them */

- *Example 5.20 (pp. 184)*

  * Wait for signal number signum

Exercises on Signals + Process synchronization

- *Compare and contrast synchronization methods*
  - * exit() - wait()
  - * kill() - pause()
  - * kill() - sigsuspend()
- *Can the window of vulnerability for pause() be closed*

- by masking signals during test of signal_received?

- *Analyze window of vulnerability for Ex. 5.20.*
  - * Who sets signal_received to non-zero value?
  - * What is mask during sigsuspend()?
  - * What is mask during test (signal_received == 0)?
- *Compare and contrast the following:*
  - * mask set by sigprocmask()
  - * mask set by sigaction()
  - * mask set by sigsuspend()

5.6-7 Implications for System calls

- *Interaction b/w signals and system calls*

  * Example 5.22 (pp. 189-90)
  - Limit wait on input to 10 second

  * restart the system calls interuppted by signals?

  * non-reentrant system calls

- *Restart issues*

  * Q? What happens if a process gets a signal
  - while executing a system call?

  * Interuptt "Slow" system calls
  - e.g. terminal I/O has indefinite wait
  - interuptted call return -1 with ’errno’ = EINTR
  - program can restart te system if needed

  * Other system calls are not interrupted
  - e.g. disk I/O, getpid() - finite or no wait

- *Example 5.21 (pp. 189)*

  * while loop restarts read() if interuppted by signal

5.6-7 Implications for System calls

- *Non-reentrant system call issue*
  - * Use of global data, e.g. errno, signal_received
  - * static data-structure - malloc(), free()
  - * executing 2 occurrence of subroutine => problems
  - * e.g. signal handler and main program
- *Async-signal safe function =*

- can be called safely with a handler

- * Does not use static data structures or malloc()
- * Does not use global data in a non-entrant way
- * Table 5.3 (pp. 191) lists async-signal safe system calls

5.6-7 Exercises on Implications for System calls

- *Q? Why following guidline for signal hanling? (pp. 190)*
  - * explicitly restart system calls within a program
  - * use async-signal safe system calls within a handler
  - * block signals to prevent unwanted interactions
- *What are following? What are those used for?*

- List a few system calls for operating on each.

  - * (a) signals, (b) masks, (a) candlers
- *Organizing the knowledge*
  - * List the system calls and structures learned in ch. 5.
  - * Group these into C++/Java like classes
  - * Identify inheritance and part-of relationships
- *Q? Where does a program return to after executing handler*

- for a signal arriving during a system call

  - * (1) next machine language instruction
  - * (2) next high-level language statement
  - * (3) end of current function or system call
  - * (4) end of current process (i.e. program)

Problems with communicating with signals

- *POSIX.1*

  * Lack of signals for application use
  - only 2, ie SIGUSR1, SIGUSR2

  * Lack of signal queueing
  - 5 signals of same type during blocked period
  - process may get 1 or 2 after unblocking

  * Signal delivery order
  - multiple pending signals -> no priority scheme

  * Information content is minimal
  - bit or an integer

  * Asynchrony
  - Must block signals during crucial sections

- *POSIX.4 real-time signals*

  * Address some of the problems

  * queued, delivered in order, carry extra data

Rest of Signals

- *5.7 Explicit control of return place after handler*

  * System calls: Siglongjmp(), sigsetjmp() (pp. 192)

  * Like "goto" and "set label" but

  * Unravel function call stack properly!

  * Ex. Program 5.2, pp. 192-3

- *5.8 Real-Time Signals (POSIX.1b)*

- *Expands ’sigaction’ structure*

- with special member function ’sa_sigaction’
- Which takes 3 parameters: (a) signal number,
- (b) info structure = signal no., cause of signal, signal value
- (c) Context - no defined
- Cause of signal = user, queue, timer, asyncIO, mesgQ
- Signal Value allows an interger /pointer parameter to handler

- *Ex. Program 5.4 (pp. 196)*

- *5.9 Asynchronous I/O*

  aio.h, aio_read(), aio_write(), ...