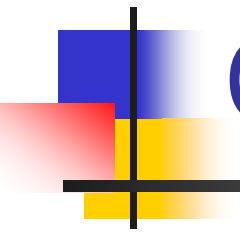
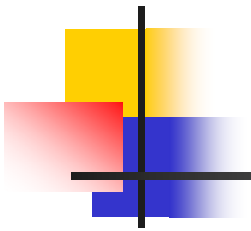


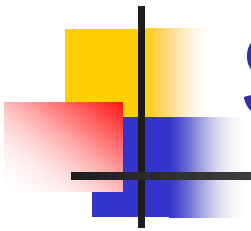
Chapter 10: Signals

CMPS 105: Systems Programming
Prof. Scott Brandt
T Th 2-3:45
Soc Sci 2, Rm. 167



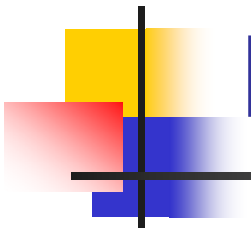
Introduction

- Signals are software interrupts
- Signals
  - Allow processes to deal with asynchronous events
    - Disk activity completion, process termination, ...
  - Allow processes to synchronize or otherwise coordinate their activities
- Each process can set up signal handlers to specify what action to take when a signal arrives



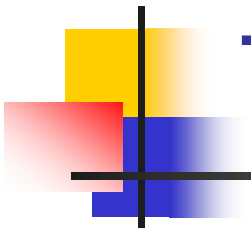
Signal Concepts

- Every signal has a name
  - They all begin with SIG
  - SIGABRT, SIGALRM, ...
  - Defined in signal.h
- Lots of conditions can generate a signal
  - Terminal generated (CTRL-C)
  - Hardware exceptions (e.g. divide by 0)
  - kill() function (allows one process to kill another)
  - kill command (kill a process from the command line)
  - Software conditions (e.g., alarm or pipe conditions)



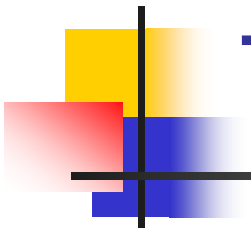
More Signal Concepts

- Signals are completely asynchronous
  - From the perspective of the receiving process
  - They arrive at apparently random times
  - Regardless of what the process is currently doing
- The process can do three things
  - Ignore the signal
    - Except SIGKILL and SIGSTOP
  - Catch the signal with a signal handler
  - Let the default action take place
    - Usually terminate



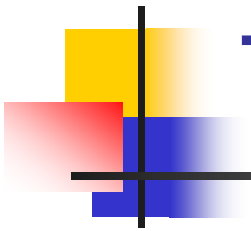
The Signals

- SIGABRT
  - Generated by calling the abort function
  - The process terminates abnormally
  - Default: terminate w/core
- SIGALRM
  - Generated when a timer (set by the process) goes off
  - Also generated when the interval timer goes off
  - Default: terminate



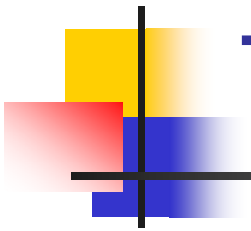
The Signals II

- SIGBUS
  - Implementation-defined hardware fault
  - Default: terminate w/core
- SIGCHLD
  - Child process has terminated or stopped
  - default: ignore
  - typical: call wait()



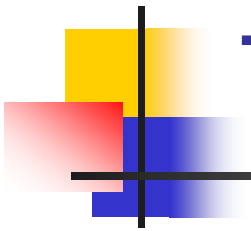
The Signals III

- SIGCONT
  - Sent to a stopped process when it is continued
  - Default: continue the process/ignore, possibly redraw (e.g. vi)
- SIGEMT
  - Implementation-defined hardware fault
  - Default: terminate w/core



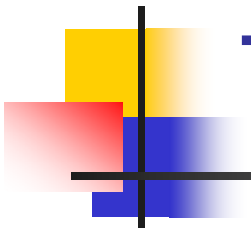
The Signals IV

- SIGFPE
  - Signals an arithmetic exception
  - Examples: divide by zero, floating-point overflow, etc.
  - Default: terminate w/core
- SIGHUP
  - Sent to the controlling process associated with a controlling terminal when a disconnect occurs
  - Also generated if session leader terminates
  - Also used to tell daemon processes to reread their configuration files
  - Default: terminate



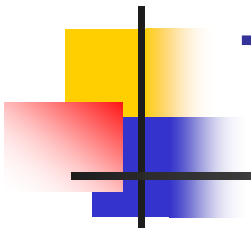
The Signals V

- SIGILL
  - Indicates that a process has executed an illegal hardware instruction
  - Default: terminate w/core
- SIGINFO
  - Generated by the terminal driver when the status key (CTRL-T) is typed
  - Sent to all processes in the foreground group
  - Causes process status to be displayed
  - Default: ignore



The Signals VI

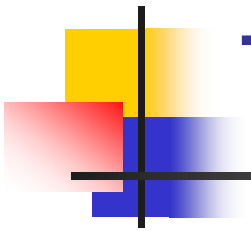
- SIGINT
  - Generated by the terminal driver when the interrupt key (DELETE or CTRL-C) is typed
  - Sent to all processes in the foreground process group
  - Often used to terminate a rogue process
  - Default: terminate
- SIGIO
  - Indicates an asynchronous I/O event
  - Default: terminate or ignore (system specific)



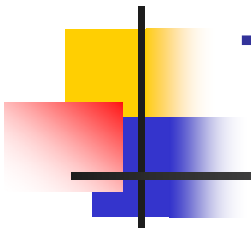
The Signals VII

- SIGIOT
  - Implementation-defined hardware fault
  - Default: terminate w/core
- SIGKILL
  - Terminates the process
  - Can’t be caught or ignored
- SIGPIPE
  - Generated when a process writes to a PIPE or socket when the other end has terminated
  - Default: terminate

The Signals VIII

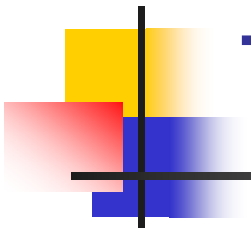


- SIGPOLL
  - Generated when a specific event occurs on a pollable device
  - Default: terminate
- SIGPROF
  - Generated when the profiling interval timer goes off
  - Default: terminate
- SIGPWR
  - System dependent UPS signal
  - Default: ignore



The Signals IX

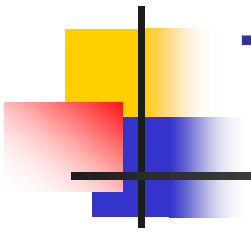
- SIGQUIT
  - Generated by the terminal driver when the quit key (CTRL-\) is typed
  - Sent to all processes in the foreground group
  - Default: terminate w/core
- SIGSEGV
  - Generated by the kernel on an invalid memory reference
  - Default: terminate w/core



The Signals X

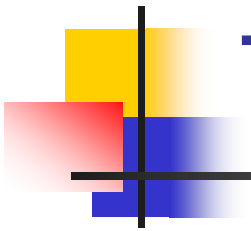
- SIGSTOP
  - Job-control signal to stop a process
  - Cannot be ignored or caught
  - Default: stop process
- SIGSYS
  - Signals an invalid system call
    - trap() instruction with bad parameters
  - Default: terminate w/core

The Signals XI

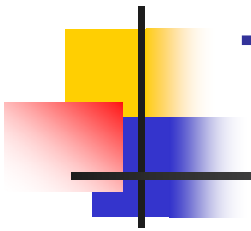


- SIGTERM
  - Generated by kill() by default
  - Defaut: terminate
- SIGTRAP
  - Implementation-defined hardware fault
  - Default: terminate w/core

The Signals XII



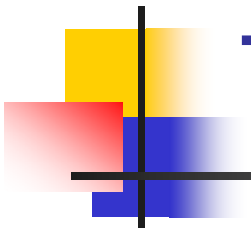
- SIGTSTP
  - Generated by the terminal driver when the suspend key (CTRL-Z) is typed
  - Default: stop process
- SIGTTIN
  - Generated by the terminal when a background process tries to read from the terminal
  - Default: stop process
- SIGTTOU
  - Generated by the terminal when a process in the background tries to write to the terminal
  - Default: stop process (optional)



The Signals XIII

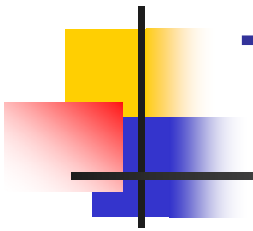
- SIGURG
  - An urgent condition has occurred
  - Optionally generated when out-of-band data is received on a network connection
  - Default: terminate
- SIGUSR1 and SIGUSR2
  - User-defined signals for use by application processes
  - Default: terminate

The Signals XIV



- SIGVTALRM
  - Generated when a virtual interval timer expires
  - Default: terminate
- SIGWINCH
  - Generated when terminal window size changes
  - Default: ignore

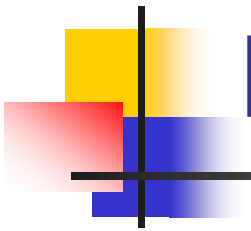
The Signals XV



- SIGXCPU
  - Generated when a process exceeds its soft CPU limit
  - Default: terminate w/core
- SIGXFSZ
  - Generated when a process exceeds its soft file size limit
  - Default: terminate w/core

The signal() function

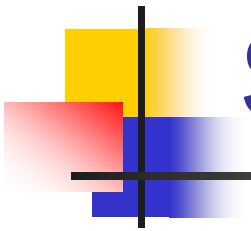
- #include <signal.h>
- void (*signal(int signo, void (*func)(int)))(int);
  - Function returning a pointer to a function that takes an int as a parameter and returns nothing
- Sets up a signal handler to be executed when the specified signal occurs
- Returns pointer to old signal handler
- Cannot be used for all signals



Program Startup

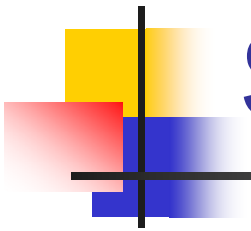
- All signals are set to the defaults, except
  - Any signal ignored by the parent is ignored by the child
  - After fork(), all signal dispositions are identical
  - After exec() any signals being caught by parent are set to the default for the child
    - The parent's handler wouldn't make any sense in the child
- Example: Shell handling of interrupt, kill, etc.

Signals during System Calls



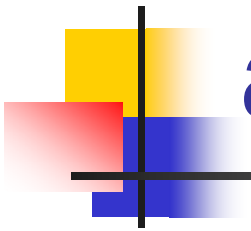
- Signals are assumed to be important
- In many Unixes, the signal will interrupt a system call to execute the handler (or default action)
- In some Unixes, the system call will restart afterwards
  - In others, not
- May depend on how the signal was set up, and which system call is interrupted

Reentrant Functions and Signals



- A reentrant function is one that can be called without harm a second time, while the first call is still active
- Recursive functions are reentrant
  - They call themselves before they have finished running
- Rule of thumb: Writes to globals => non-reentrant
- Issue: non-reentrant functions called from signal handlers may collide with previous calls from main body of code
- Solution: Write simple signal handlers, and only call reentrant functions
- This same issue comes up with threads

Reliable Signal Terminology and Semantics

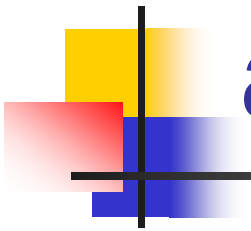


- Signal is *generated* when the event causing the signal occurs
  - Hardware exception, software condition, terminal-generated, or a call to kill()
  - Usually indicated by a flag in the process table
- Signal is *delivered* when the action for a signal is taken
- Between generation and delivery, a signal is *pending*
- A process can *block* a signal, delaying its delivery until it is unblocked
- Signal mask defines what signals are blocked

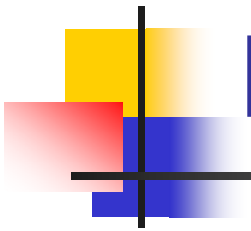
kill() and raise()

- #include <sys/types.h>
- #include <signal.h>
- int kill(pid_t *pid*, int *signo*);
  - sends a signal to another process or group of processes
  - Four conditions for the pid argument
    - pid > 0, signal sent to process with PID=pid
    - pid == 0, signal sent to all processes in same group
    - pid < 0, signal sent to all process whose process group ID = |pid|
    - pid == -1, undefined
    - Permissions have to match (basically = user IDs match)
- int raise(int *signo*);
  - sends a signal to the calling process

alarm() and pause() functions

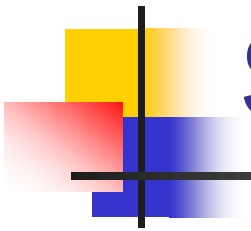


- #include <unistd.h>
- unsigned int alarm(unsigned int *seconds*);
  - Allows us to set a timer that will expire in the specified number of seconds, at which time we will receive a SIGALRM signal
  - Returns 0 or seconds remaining until a previously specified alarm
  - Usually we catch the signal with a specified handler function
- int pause(void);
  - Suspends the process until a signal arrives
  - Returns after handler runs



Problematic Example

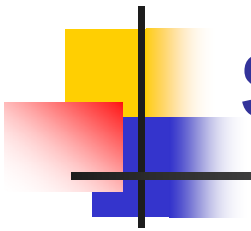
```
#include <signal.h>
#include <unistd.h>
static void sig_alrm(int signo) {
    printf("Signal handler is running");
}
unsigned int sleep1(unsigned int nsecs) {
    if(signal(SIGALRM, sig_alrm) == SIG_ERR)
        return(nsecs);
    alarm(nsecs);
    pause();
    printf("sleep1 running after pause");
    return (alarm(0));
}
// Better example uses semaphores, as discussed in class
```



Signal Sets

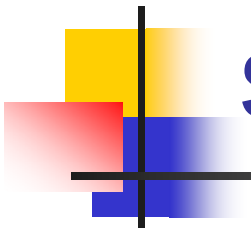
- #include <signal.h>
- int sigemptyset(sigset_t *set);
- int sigfillset(sigset_t *set);
- int sigaddset(sigset_t *set, int signo);
- int sigdelset(sigset_t *set);
- int sigismember(const sigset_t *set, int signo);
- sigset_t is a data type that can refer to multiple signals at once

sigprocmask()



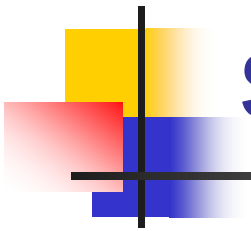
- #include <signal.h>
- int sigprocmask(int *how*, const sigset_t **set*, sigset_t **oset*);
  - Changes the signal mask for the process
- The signal mask determines which signals are blocked
- Set specifies the signals (if non-null)
- Current signal mask returned in *oset* (if non-null)
- How determines what is done with *set*
  - SIG_BLOCK – specified signals are blocked
  - SIG_UNBLOCK – specified signals are unblocked
  - SIG_SETMASK – only specified signals are blocked

sigpending()



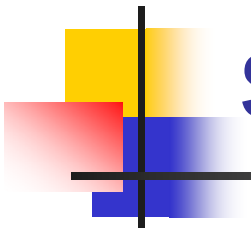
- #include <signal.h>
- int sigpending(sigset_t **set*);
- Returns the set of signals that are currently pending
  - Must have been blocked to be pending

sigaction()



- #include <signal.h>
- int sigaction(int *signo*, const struct sigaction **act*, struct sigaction **oact*);
- Supercedes signal();
  - Allows handler to temporarily block new signals
  - Adds other options

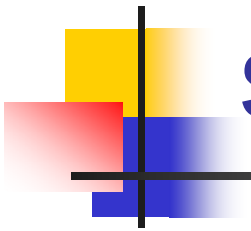
```
struct sigaction {
    void (*sahandler)();
    sigset_t sa_mask;
    int sa_flags;
}
```



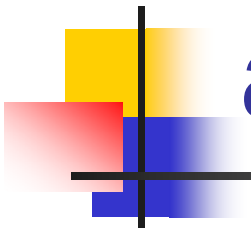
sa_flags

- sa_flags determine certain aspects of signal handling
  - SA_NOCLDSTOP – don't deliver signal on child job-control stop
  - SA_RESTART – restart interrupted system calls
  - SA_ONSTACK – use a different stack for the signal handler
  - SA_NOCLDWAIT – don't let child processes become zombies
  - SA_NODEFER – don't block this signal while executing handler
  - SA_RESETHAND – revert to default action for this signal after the handler executes
  - SA_SIGINFO – provides additional information to a signal handler

sigsuspend()

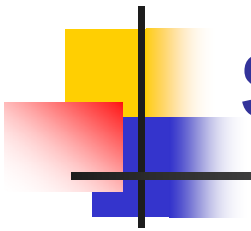


- #include <signal.h>
- int sigsuspend(const sigset_t **sigmask*);
- Atomically sets up a signal mask and suspends the process
- Gets rid of possible race condition between enabling a signal and pausing to await its arrival



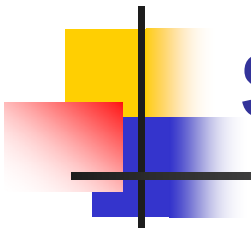
abort()

- #include <stdlib.h>
- void abort(void);
- Shouldn’t be ignored
- Can be caught, but can’t be returned from
- Used to perform cleanup before exit



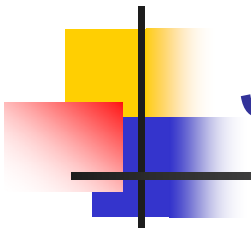
system()

- System forks/execs a program
- Should ignore SIGINT, SIGQUIT, and SIGCHLD
  - Why?
  - Because SIGINT and SIGQUIT go to all foreground processes
  - SIGCHLD doesn't have a handler



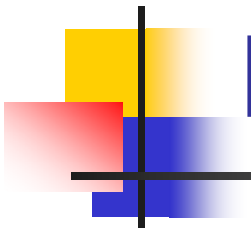
sleep()

- #include <unistd.h>
- unsigned int sleep(unsigned int *seconds*);
- Causes the calling process to be suspended until either
  - The specified number of seconds has elapsed; or
  - Any signal is caught by the process and the signal handler returns
- Actual time that the process starts running may be any time after *seconds* has elapsed
  - Can be delayed due to other system activity
- sleep() can be implemented with alarm(), but this can cause unwanted interactions



Job-Control Signals

- SIGCHLD – child process has stopped or terminated
- SIGCONT – continue process, if stopped
- SIGSTOP – Stop signal (can't be caught or ignored)
- SIGTSTP – Interactive stop signal
- SIGTTIN – Read from controlling terminal by member of background process group
- SIGTTOU – Write to controlling terminal by member of background process group



Extra Features

- In some systems:
- Signal names
  - extern char *sys_siglist[];
- void psignal(int signo, const char *msg);
  - prints out msg: description of signal