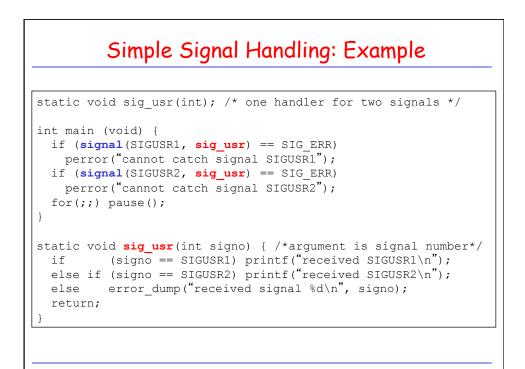


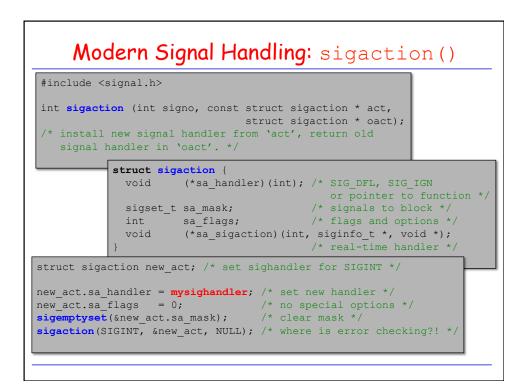
No	Name	Default Action	Description	No	Name	Default Action	Description
1 2	SIGHUP SIGINT	terminate process terminate process	terminal line hangup interrupt program	17	SIGSTOP	stop process	stop (cannot be caught or ignored)
3 4	SIGQUIT	create core image create core image	quit program illegal instruction	18	SIGTSTP	stop process	stop signal generated from keyboard
-	SIGTRAP	create core image	trace trap	19	SIGCONT	discard signal	continue after stop
	SIGABRT	create core image	abort program (formerly SIGIOT)	20	SIGCHLD	discard signal	child status has changed
7	SIGEMT	create core image		21	SIGTTIN	stop process	background read
,	0100111	create core image	executed			attempte	ed from control termina
8	SIGFPE	create core image	floating-point exception		SIGTTOU		background write pted to control termina
9	SIGKILL	terminate process	kill program	23	SIGIO	discard signal	I/O is possible on a escriptor (see fcntl(2)
10	SIGBUS	create core image	bus error	24	SIGXCPU	terminate process	* · · · · ·
11	SIGSEGV	create core image		24	DIGNOLO	*	eeded (see setrlimit(2)
12	SIGSYS	create core image	violation non-existent system call invoked	25	SIGXFSZ	terminate process exce	file size limit eded (see setrlimit(2)
13	SIGPIPE	terminate process		26	SIGVTALRM	terminate process	virtual time alarm (see setitimer(2))
		*	no reader	27	SIGPROF	terminate process	profiling timer alarm
14	SIGALRM	terminate process	real-time timer expired				(see setitimer(2))
15	SIGTERM	terminate process	software termination signal		SIGWINCH SIGINFO	discard signal discard signal	Window size change status request from keyboard
16	SIGURG	discard signal	urgent condition	30	STGUSB1	terminate process	-
			present on socket		SIGUSB2	terminate process	-
						-	

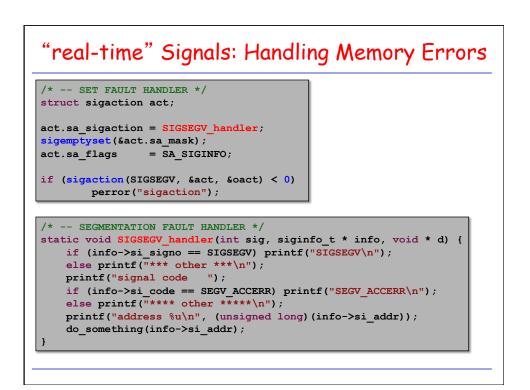
Generating Signals: kill(2) and raise(3) #include <signal.h> int kill(pid_t pid, int sig); /* send signal `sig' to process `pid' */ //* example: send signal SIGUSR1 to process 1234 */ if (kill(1234, SIGUSR1) == -1) perror("Failed to send SIGUSR1 signal"); /* example: kill parent process */ if (kill(getppid(), SIGTERM) == -1) perror("Failed to kill parent"); #include <signal.h>

int raise(int sig);
 /* Sends signal 'sig' to itself.
 Part of ANSI C library! */

#include <signal.h></signal.h>	defining signal handlers the old-fashioned way
void (* signal (int sign	no, void (*func)(int)))(int);
	In English: "The function signal takes two arguments: an integer and a pointer to a function that takes an integer and returns nothing. The function signal itself returns a pointer to a
The prototype can be	function that takes an integer as argument and returns nothing."
The prototype can be simplified through the use of	returns nothing."

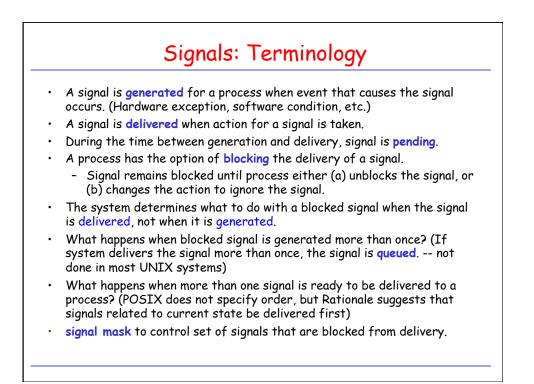


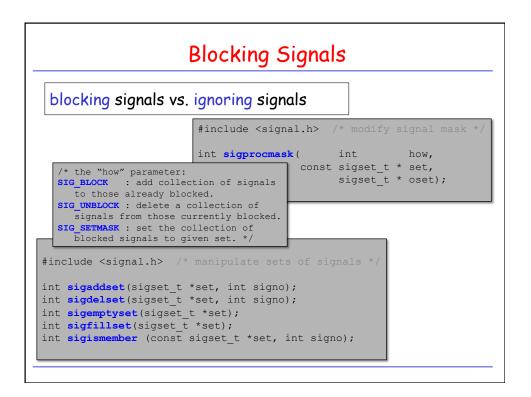


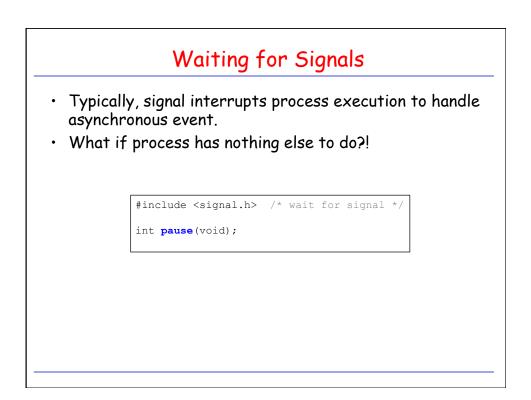


Need more Details?!! : ucontext

```
/* -- SEGMENTATION FAULT HANDLER */
static void
SIGSEGV handler(int sig, siginfo t * info, ucontext t * uc) {
    [. . . ]
    /* -- IDENTIFY INSTRUCTION THAT CAUSED FAULT */
   unsigned long pc, *pcptr, instruction;
#if defined(SOLARIS)
             = (unsigned long) uc->uc_mcontext.gregs[1];
   pc
   pcptr
              = (unsigned long *) pc;
   instruction = *pcptr;
#endif
   /* -- READ OR WRITE OPERATION? */
   read_fault = LOAD_INSTRUCTION(instruction);
   write fault = STORE INSTRUCTION (instruction);
    [...]
```







How do we wait for particular Signal? /* Approach 1, using a global variable (buggy!) */ /* Have the signal handler set quitflag to 1. */ static volatile sig atomic t quitflag = 0; while (quitflag == 0) pause(); ′* ?! */́ /* Approach 2, using global variable (also buggy!) */ /* Have the sighandler set quitflag to 1. */ static volatile sig atomic t quitflag = 0; signum; int sigset t sigset; sigemptyset(&sigset); sigaddset(&sigset, signum); sigprocmask(SIG_BLOCK, &sigset, NULL); while (quitflag == 0) pause(); /* ?! */

