



# Operating Systems CS F372

## Signals Additional Reading

**BIJU K RAVEENDRAN**

# Signal

- Is a software interrupt delivered to a process.
- The events that generate signals fall into three major categories:
  - errors, external events, and explicit requests
  - error means that a program has done something invalid and cannot continue execution
    - Most of the error will not generate signals
  - external event generally has to do with I/O or other processes. (arrival of input, expiration of a timer, and termination of a child process)
  - explicit request means the use of a library function such as kill whose purpose is specifically to generate a signal.



# Signal

- Signals may be generated *synchronously* or *asynchronously*.
- A synchronous signal pertains to a specific action in the program, and is delivered (unless blocked) during that action.
  - Errors and explicit requests by a process to the same process generate signals synchronously.
- Asynchronous signals are generated by events outside the control of the process that receives them.
  - arrives at unpredictable times during execution.
  - External events and explicit requests that apply to some other process generate asynchronous signals





# Signal

- Linux support
  - POSIX reliable signals and
  - POSIX real time signals
- Some signals are architecture dependent
- Action for the signal
  - Term (action - terminate the process)
  - Ign (action – ignore the signal)
  - Core (action – terminate the process with core dump)
  - Stop (action – stop the process)



1	SIGHUP	2	SIGINT	3	SIGQUIT	4	SIGILL
5	SIGTRAP	6	SIGABRT	7	SIGBUS	8	SIGFPE
9	SIGKILL	10	SIGUSR1	11	SIGSEGV	12	SIGUSR2
13	SIGPIPE	14	SIGALRM	15	SIGTERM	16	SIGSTKFLT
17	SIGCHLD	18	SIGCONT	19	SIGSTOP	20	SIGTSTP
21	SIGTTIN	22	SIGTTOU	23	SIGURG	24	SIGXCPU
25	SIGXFSZ	26	SIGVTALRM	27	SIGPROF	28	SIGWINCH
29	SIGIO	30	SIGPWR	31	SIGSYS	33	SIGRTMIN
34	SIGRTMIN+1	35	SIGRTMIN+2	36	SIGRTMIN+3	37	SIGRTMIN+4
38	SIGRTMIN+5	39	SIGRTMIN+6	40	SIGRTMIN+7	41	SIGRTMIN+8
42	SIGRTMIN+9	43	SIGRTMIN+10	44	SIGRTMIN+11	45	SIGRTMIN+12
46	SIGRTMIN+13	47	SIGRTMIN+14	48	SIGRTMIN+15	49	SIGRTMAX-14
50	SIGRTMAX-13	51	SIGRTMAX-12	52	SIGRTMAX-11	53	SIGRTMAX-10
54	SIGRTMAX-9	55	SIGRTMAX-8	56	SIGRTMAX-7	57	SIGRTMAX-6
58	SIGRTMAX-5	59	SIGRTMAX-4	60	SIGRTMAX-3	61	SIGRTMAX-2
62	SIGRTMAX-1	63	SIGRTMAX				

# Signals described in the original POSIX.1 standard

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm
SIGTERM	15	Term	Termination signal
SIGUSR1	10	Term	User-defined signal 1
SIGUSR2	12	Term	User-defined signal 2
SIGCHLD	17	Ign	Child stopped or terminated
SIGCONT	18	Cont	Continue if stopped
SIGSTOP	19	Stop	Stop process
SIGTSTP	20	Stop	Stop typed at tty
SIGTTIN	21	Stop	tty input for background process
SIGTTOU	22	Stop	tty output for background process

### Signals not in POSIX.1 standard but described in SUSv2 and SUSv3 / POSIX 1003.1-2001

Signal	Value	Action	Comment
SIGBUS	7	Core	Bus error (bad memory access)
SIGPOLL		Term	Pollable event (Sys V). Synonym of SIGIO
SIGPROF	27	Term	Profiling timer expired
SIGTRAP	5	Core	Trace/breakpoint trap
SIGURG	23	Ign	Urgent condition on socket (4.2BSD)
SIGVTALRM	26	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	24	Core	CPU time limit exceeded (4.2BSD)
SIGXFSZ	25	Core	File size limit exceeded (4.2BSD)

### Various other signals

Signal	Value	Action	Comment
SIGIOT	6	Core	IOT trap. A synonym for SIGABRT
SIGSTKFLT	16	Term	Stack fault on coprocessor (unused)
SIGIO	29	Term	I/O now possible (4.2BSD)
SIGPWR	30	Term	Power failure (System V)
SIGWINCH	28	Ign	Window resize signal (4.3BSD, Sun)
SIGUNUSED	31	Term	Unused signal (will be SIGSYS)



Friday, September 13,

- Program error signals
  - SIGFPE
  - SIGILL
  - SIGSEGV
  - SIGBUS
  - SIGABRT
- TERMINATION SIGNALS
  - SIGUP
  - SIGINT
  - SIGQUIT
  - SIGTERM
  - SIGKILL
- Asynchronous IO signals
  - SIGIO
  - SIGURG
- Alarm signals
  - SIGALRM
  - SIGVTALRM
  - SIGPROF
- Job control signals
  - SIGCHLD
  - SIGCONT
  - SIGSTOP
  - SIGTSTP
  - SIGTTIN
  - SIGTTOU
- Miscellaneous signals
  - SIGPIPE
  - SIGUSR1
  - SIGUSR2



# Signal

- Linux supports 32 real time signals
  - Signal number 33 (SIGRTMIN) to 63 (SIGRTMAX)
  - Programs should refer to real time signals using SIGRTMIN+n as range of real time signal numbers changes in different distribution
  - Real time signal have no predefined meaning
  - The entire set can be used for application defined purposes
  - Default action for an unhandled real time system is to terminate the receiving process



# Signal

- Real time signals are distinguished by
  - Multiple instances of real-time signals can be queued
    - If multiple instances of a standard signal are delivered while that signal is currently blocked, then only one instance is queued.
  - If the signal is sent using sigqueue, an accompanying value (either an integer or a pointer) can be sent with the signal.
    - If the receiving process establishes a handler for this signal using the SA\_SIGINFO flag to sigaction then it can obtain this data via the si\_value field of the siginfo\_t structure passed as the second argument to the handler. Furthermore, the si\_pid and si\_uid fields of this structure can be used to obtain the PID and real user ID of the process sending the signal.

# Signal

- Real-time signals are delivered in a guaranteed order. Multiple real-time signals of the same type are delivered in the order they were sent.
  - If different real-time signals are sent to a process, they are delivered starting with the lowest-numbered signal.
  - I.e., low-numbered signals have highest priority.
- If both standard and real-time signals are pending for a process, POSIX leaves it unspecified which is delivered first.
  - Linux, like many other implementations, gives priority to standard signals in this case.
- POSIX implementation permits at least `_POSIX_SIGQUEUE_MAX` (32) real-time signals to be queued to a process

# Signal

- `sighandler_t signal(int signum, sighandler_t handler)`
  - Installs a new signal handler for the signal with number `signum`
  - Signal handler is set to `sighandler` which may be
    - A user specified function
    - `SIG_IGN`
    - `SIG_DFL`
  - When a signal with `signum` arrives
    - If the handler is set to `SIG_IGN` then the signal is ignored
    - If the handler is set to `SIG_DFL` then default action associated with the signal occurs
    - If the handler is set to a function then first either the handler is reset to `SIG_DFL` or an implementation dependant blocking of the signal is performed and next `sighandler` is called with `signum`.



# Signal


- The signals SIGSTOP and SIGKILL can not be caught or ignored.
- Return value
  - Returns the previous value of the signal handler
  - Or SIF\_ERR on error
- The behavior of a process is undefined after ignoring SIGFPE, SIGILL or SIGSEGV.





# Signal

```
main() {  
    (void) signal(SIGINT,leave);  
    temp_file = fopen("tmp","w");  
    for(;;) { /* Do things.... */  
        printf("Ready...\n");  
        (void) getchar();  
    }  
    /* can't get here ... */  
    exit(EXIT_SUCCESS);  
}  
  
void leave(int sig) {  
    fprintf(temp_file,"\nInterrupted..\n");  
    fclose(temp_file);  exit(sig);  
}
```



```
#include <stdio.h>  
#include <stdlib.h>  
#include <signal.h>  
FILE *temp_file;  
void leave(int sig);
```

# Signal

- `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
  - Change the action taken by the process on receipt of a specific signal
  - Signum specifies the signal (can be any valid signal except SIGKILL and SIGSTOP)
  - If act is not NULL, the new action of the signal signum is installed from act
  - If oldact is not NULL, the previous action is saved in oldact.
  - Return 0 on success and -1 on error.



# Signal

- `int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);`
  - Used to change the list of currently blocked signals.
  - how values
    - SIG\_BLOCK (set of blocked signal is the union of the current set and the set argument)
    - SIG\_UNBLOCK (the signal in set are removed from the current set of blocked signals. It is legal to attempt to unblock signal which is not blocked)
    - SIG\_SETMASK (the set of blocked signal is set to the argument set)
    - If oldset is not NULL, the previous value of the signal mask is stored in oldset
  - Return 0 on success and -1 on error

# Signal

- `int sigpending(sigset_t *set)`
  - Allows execution of pending signals (one which have been raised while blocked). The signal mask of pending signals is stored in `set`.
  - Return 0 on success and `-1` on error.
- `int sigsuspend(const sigset_t *mask);`
  - Temporarily replaces a signal mask for the process with that given by `mask` and then suspends the process until a signal is received.
  - Always return `-1` normally with the error `EINT`





# Signal

- `sigqueue(pid_t pid, int sig, const union signal value);`
  - Sends the signal specified in `sig` to the process whose PID is given in `pid`.
  - The null signal (0) can be used to check if a process with a given PID exists.
  - Value is used to specify an accompanying item of data (integer or pointer) to be sent with the signal
  - Returns 0 on success (signal has successfully queued to the receiving process). -1 on error.





# Signal

- `int sigwaitinfo(const sigset_t *set, siginfo_t *info);`
  - Suspends execution of the calling process until one of the signals in set is delivered
  - Will return immediately if one of the signals in the set is already pending
  - Removes the delivered signal from the calling process' list of pending signals and returns the signal number as its function result.

# Signal

- `int siginterrupt(int sig, int flag);`
  - Changes the restart behavior when a system call is interrupted by the signal `sig`.
  - If flag value is false (0) then system call will be restarted if interrupted by the specified signal `sig`.
  - If the flag value is true (1) and no data has been transferred then a system call interrupted by the signal `sig` will return `-1` (err – `EINTR`)
  - If the flag value is true (1) and data transfer has started, the system call will be interrupted and will return the actual amount of data transferred.



# Signal

- `int sighold(int sig)` -- adds sig to the calling process's signal mask
- `int sigrelse(int sig)` -- removes sig from the calling process's signal mask
- `int sigignore(int sig)` -- sets the disposition of sig to SIG\_IGN
- `int sigpause(int sig)` -- removes sig from the calling process's signal mask and suspends the calling process until a signal is received



# Signal

- `Int sigemptyset(sigset_t *set)`
  - Initializes the signal set (set) to empty
- `int sigfillset(sigset_t *set);`
  - Initializes the signal set (set) to full
- `int sigaddset(sigset_t *set, int signum);`
  - Add signum to signal set (set)
- `int sigdelset(sigset_t *set, int signum);`
  - Delete signum from signal set (set)
- `Int sigismember(const sigset_t *set, int signum);`
  - Tests whether signum is a member of set.
  - Return 1 if signum is a member, 0 if signum is not a member and -1 on error





# Signal

- `Void psignal(int sig, const char *s);`
  - Displays a message on stderr consisting of the string `s`, a colon, a space, and a string describing the signal number `sig`.
  - If `sig` is invalid, the message displayed will indicate an unknown signal.





# Signal

- `int kill(pid_t pid, int sig);`
  - Can be used to send any signal to any process group or process
  - If no signal is specified, the term signal is sent.
  - If pid is positive, then the sig is sent to pid
  - If pid equals 0, then sig is sent to every process in the process group of the current process
  - If pid equals -1, then sig is sent to every process in the process group -pid
  - If sig is 0, then no signal is sent but error checking is still performed



# Signal

- It is not possible to send a signal to init process.
  - To assure the system is not brought down accidentally
- Kill(-1, sig) will send signal to all processes
  - Does not signal the current process
- Kill options
  - -s signal (specify the signal to send)
  - -l (print a list of signal names)
  - -a (Do not restrict the command to pid conversion to processes with the same uid as the present process)
  - -p (Specify that kill should only print the process id of the named process, not send any signal)



# Signal

- `int killpg(int pgrp, int sig);`
  - Sends signal `sig` to process group `pgrp`.
  - If `pgrp` is 0, `killpg` sends the signal to the sending process's process group.
  - The sending process and the members of the process group must have the same effective user id, or the sender must be a super user
  - Special case: `SIGCONT` may be sent to any process that is descendant of the current process.



# Signal

- Other commands related to kill
  - pkill
    - Will send the specified signal (default SIGTERM) to each process.
    - For more information about various options man pkill
  - skill
    - Report process status
    - See man skill for more information
  - killall
    - Send a signal to any of the processes running any of the specified commands.
    - If no signal is specified SIGTERM is sent
    - never kill itself (may kill other killall processes)



# Signal

- `int pause(void);`
  - suspends program execution until a signal arrives whose action is either to execute a handler function, or to terminate the process
  - Only returns when a signal was caught and the signal catching function returned.
  - simplicity of `pause` can conceal serious timing errors that can make a program hang mysteriously
  - can't safely use `pause` to wait until one more signal arrives, and then resume real work
  - Can lead to infinite waiting because of signal miss





# Signal

- `int raise(int sig);`
  - Sends a signal to the current process
  - Equivalent to `kill(getpid(),sig);`



# Tracing

- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets



# BCC

- Debugging interactions between user-level and kernel code nearly impossible without toolset that understands both and an instrument their actions
- BCC (BPF Compiler Collection) is a rich toolkit providing tracing features for Linux
  - See also the original DTrace
- For example, disksnoop.py traces disk I/O activity

TIME(s)	T	BYTES	LAT(ms)
1946.29186700	R	8	0.27
1946.33965000	R	8	0.26
1948.34585000	W	8192	0.96
1950.43251000	R	4096	0.56
1951.74121000	R	4096	0.35

- Many other tools (next slide)

# Linux BCC / BPF Tracing Tools

## Linux bcc/BPF Tracing Tools

