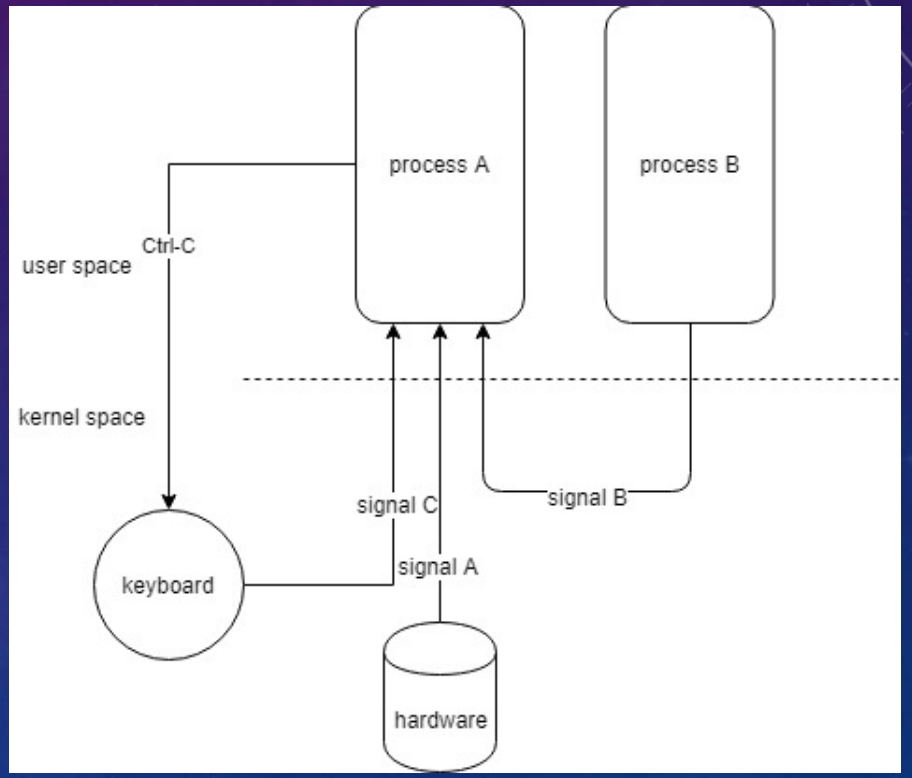
SIGNAL IN LINUX

CONTENT

- Fundamental concept
- Signal categories
- Signal type and action
- Signal handler
- Signal set and mask
- Abort and alarm
- Coredump file
- Realtime signal

FUNDAMENTAL CONCEPT

- Overview
  - A notification to process that an event has occurred
  - software interrupt
  - In most case, it is not possible to predict exactly when a signal arrive
  - A process can send a signal to other process or itself – a form of IPC
  - Normal source of various signal is the kernel. And type of events that cause the kernel to generate a signal for a process
    - Hardware exception occur – Malformed machine language instruction, dividing 0, access to part of memory that inaccessible
    - The user typed one of the terminal special character as interrupt character (Ctrl-C) or suspend character (Ctrl-D)
    - Software event – Input become available of file descriptor, terminal window is resize, timer went off, CPU time limit was exceeded or child of process is terminated



SIGNAL CATAGORIES

- Each s
  from 1
  -
  -
- A sign
  to a p
  is pen
- A pen
  it is ne
  is runn
- Some
  is not
  add a
  will bl

```
thientran@ubuntu:~$ kill -l
 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      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+
3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+
8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+
13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-
12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-
7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-
2
63) SIGRTMAX-1  64) SIGRTMAX
thientran@ubuntu:~$
```

SIGNAL ACTION

- Upon delivery of signal, a process carries out one of the following default action
  - ignored – it is discarded by kernel and has no effect on process
  - abnormal process termination
    - Cored dump file and process termination
  - Execution process is stopped
  - Execution process is resumed after stopped

- A program can set the signal disposition
  - Default action
  - Ignored
  - Signal handler

SIGNAL TYPES AND DEFAULT ACTION

Signal name	Description	Default action
SIGARBT	A process receive this signal when it calls abort function	terminate process; generate core-dump (use for debugging)
SIGALRM	Kernel generate this signal upon the expiration of a realtime timer that setup by alarm or setitimer function	terminate process
SIGBUS	Generation to indicate certain kind of memory error. One of such error is access to an address beyond underlying memory-mapped file	terminate process; generate core dump
SIGCHILD	Kernel generate and sent to parent process when one of child process terminate, kill by signal, stopped	Irgnored
SIGCONT	Kernel generate and sent to parent process when one of child process resume from stopped state	Resume process
SIGFPE	Generation to indicate certain type of arithmetic, such as devide-by-zero	terminate process; generate core dump

SIGNAL TYPES AND DEFAULT ACTION (2)

Signal name	Description	Default action
SIGHUP	When terminal disconnect this signal will send to controlling process of terminal Deamon use this signal to reinitialize itself and update the configuration	terminate process
SIGILL	Process tries illegal machine-languge instruction	terminate process; generate core dump
SIGINT	User type interrupt character Ctrl-C	terminate process;
SIGIO	Generate a IO event occur on certain type of file descriptor such terminal of socket	terminate process;
SIGKILL	Sure kill signal. Terminate process. Cannot blocked, irgnored, caught by a handler	terminate process
SIGPIPE	Process tries to write to PIPE , FIFO, or socket that closed	irgnored
SIGTRAP	Use for debugging tool as gdb for breakpoint trace	terminate process; generate core dump

SIGNAL TYPES AND DEFAULT ACTION (2)

Signal name	Description	Default action
SIGQUIT	User type quit character Ctrl \. This signal useful with program stuck in an infinite loop or otherwise not responding	terminate process; generate core dump
SIGSEGV	Very popular, is generated when program make an invalid memory reference as page is not exist, write to read only area, access the kernel memory, dereferencing a pointer containing a bad address …	terminate process; generate core dump
SIGSTOP	To surely give a process to stopped state. It cannot blocked, ignored, caught by handler	stop process;
SIGTERM	To terminate a process gracefully. Free all resource explicitly.	terminate process;
SIGKILL	Sure kill signal. Terminate process. Cannot blocked, irgnored, caught by a handler	terminate process; not generate core dump
SIGUSR1/SI	Programmer define purpose	terminate process;

CHANGING SIGNAL DISPOSITION -SIGNAL

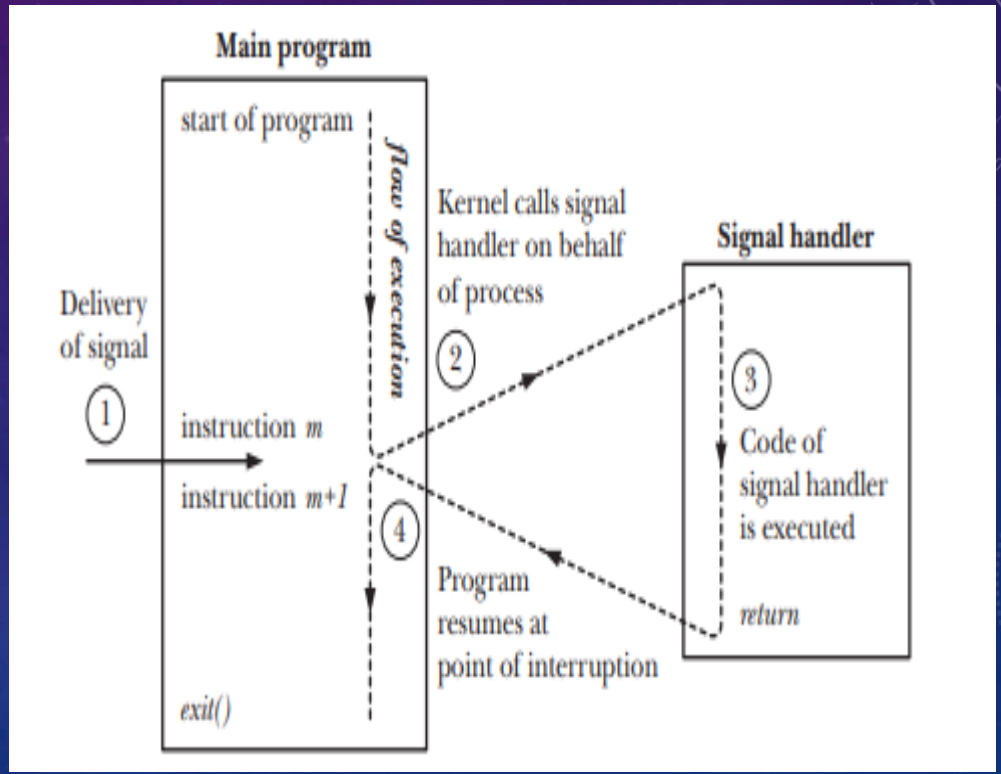
- signal function
  - Have arguments as signal number and signal handler
  - Return previous disposition
  - SIG_DFL
    - Reset the disposition to its default
  - SIG_IGN
    - Ignore the signal

```
void ( *signal(int sig, void (*handler)(int)) ) (int);
```

```
void (*oldHandler)(int);

oldHandler = signal(SIGINT, newHandler);
if (oldHandler == SIG_ERR)
    errExit("signal");

/* Do something else here. During this time, if SIGINT is
   delivered, newHandler will be used to handle the signal. */

if (signal(SIGINT, oldHandler) == SIG_ERR)
    errExit("signal");
```

SIGNAL HANDLER

- Signal handler
  - can interrupt flow of main program
  - After signal handler return, execution of the program resumes at the point where the hander interrupt it
  - Signal handler can do anything, but it should be design as simple as possible
  - A program can setup the same handler for multiple signal



SENDING SIGNAL

- kill
  - A process send signal to other process
  - Broadcast signal when pid is -1
  - Checking for existence of process with sig = 0, no signal send to process
    - ESRCH – receiving process doesn't exist
    - EPERM – sending process doesn't permission
    - Success – sending process have permission and receiving process exist
- raise
  - sending signal to itself
  - equivalent with kill(getpid(), sig)

```
int kill(pid_t pid, int sig);
```

```
int raise(int sig);
```

```
char *strsignal(int sig);
```

SIGNAL SETS

- signal sets framework
  - Allow a group of signal stand together into a group
  - A bitmask method

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
```

```
int sigaddset(sigset_t *set, int sig);
int sigdelset(sigset_t *set, int sig);
```

```
int sigismember(const sigset_t *set, int sig);
```

```
int sigandset(sigset_t *set, sigset_t *left, sigset_t *right);
int sigorset(sigset_t *dest, sigset_t *left, sigset_t *right);
```

```
void                    /* Print list of signals within a signal set */
printSigset(FILE *of, const char *prefix, const sigset_t *sigset)
{
    int sig, cnt;

    cnt = 0;
    for (sig = 1; sig < NSIG; sig++) {
        if (sigismember(sigset, sig)) {
            cnt++;
            fprintf(of, "%s%d (%s)\n", prefix, sig, strsignal(sig));
        }
    }

    if (cnt == 0)
        fprintf(of, "%s<empty signal set>\n", prefix);
}
```

SIGNAL MASK

- Process signal mask
  - A set of signal whose delivery to process is currently block
  - Sigprocmask use to explicitly add signal to or remove signal from the signal mask
  - Signal are not queued

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

- how
  - SIG_BLOCK
  - SIG_UNBLOCK
  - SIG_SETMASK

CHANGING SIGNAL DISPOSITION - SIGACTION

- sigaction
  - More complex but more flexibility
  - Get current disposition without changing it
  - Setup various control what happen when signal handler is invoked

```
int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);
```

```
struct sigaction {
    void    (*sa_handler)(int);    /* Address of handler */
    sigset_t sa_mask;              /* Signals blocked during handler
                                      invocation */
    int      sa_flags;             /* Flags controlling handler invocation */
    void    (*sa_restorer)(void);  /* Not for application use */
};
```

- sa_flags
  - Bitmask specifying how signal handler, use OR for multiple options
    - SA_NOCLDSTOP – If signal is SIGCHLD, don't generate this signal when a child process stopped/resume
    - SA_NOCLDWAIT – if signal is SIGCHLD, don't transform children into zoobies when they terminate
    - SA_NODEFER – when signal is caught, don't add it to the process signal mask while the handler executing
    - SA_ONSTACK - Invoke the handler for this signal using an alternate stack installed by sigaltstack() func
    - SA_RESETHAND – When this signal caught, reset its disposition to the default before invoking the handler
    - SA_RESTART – Automatically restart system calls interrupted by this signal
    - SA_SIGINFO – Invoke the signal with addition argument

WAITING FOR A SIGNAL

```
int pause(void);
```

- pause
  - Suspend execution of process until the call is interrupted by a signal handler
  - Pause always return -1 when signal arrive

TERMINATE A PROCESS WITH ABORT

```
void abort(void);
```

- abort function
  - Terminate calling process by raising a SIGABRT signal
  - Default action is generated a core dump file and terminate process
  - The core file use to debug the program at the time of the abort()

RAISE A TIMER WITH ALARM

SA_SIGINFO FLAG

- SA_SIGINFO
  - Allow the handler obtain additional information about signal when it is delivered
  - The handler need to declare full arguments
  - siginfo_t will provice full information of signal

```
struct sigaction {
    union {
        void (*sa_handler)(int);
        void (*sa_sigaction)(int, siginfo_t *, void *);
    } __sigaction_handler;
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

```
void handler(int sig, siginfo_t *siginfo, void *ucontext);
```

```
typedef struct {
    int     si_signo;        /* Signal number */
    int     si_code;         /* Signal code */
    int     si_trapno;       /* Trap number for hardware-generated signal
                                (unused on most architectures) */
    union sigval si_value;   /* Accompanying data from sigqueue() */
    pid_t   si_pid;          /* Process ID of sending process */
    uid_t   si_uid;          /* Real user ID of sender */
    int     si_errno;        /* Error number (generally unused) */
    void   *si_addr;         /* Address that generated signal
                                (hardware-generated signals only) */
    int     si_overrun;      /* Overrun count (Linux 2.6, POSIX timers) */
    int     si_timerid;      /* (Kernel-internal) Timer ID
                                (Linux 2.6, POSIX timers) */
    long    si_band;         /* Band event (SIGPOLL/SIGIO) */
    int     si_fd;           /* File descriptor (SIGPOLL/SIGIO) */
    int     si_status;       /* Exit status or signal (SIGCHLD) */
    clock_t si_utime;        /* User CPU time (SIGCHLD) */
    clock_t si_stime;        /* System CPU time (SIGCHLD) */
} siginfo_t;
```

SI_CODE OF SIGINFO_T

Signal	*si_code* value	Origin of signal
Any	SI_ASYNCIO	Completion of an asynchronous I/O (AIO) operation
	SI_KERNEL	Sent by the kernel (e.g., a signal from terminal driver)
	SI_MESGQ	Message arrival on POSIX message queue (since Linux 2.6.6)
	SI_QUEUE	A realtime signal from a user process via *sigqueue()*
	SI_SIGIO	SIGIO signal (Linux 2.2 only)
	SI_TIMER	Expiration of a POSIX (realtime) timer
	SI_TKILL	A user process via *tkill()* or *tgkill()* (since Linux 2.4.19)
	SI_USER	A user process via *kill()* or *raise()*
SIGBUS	BUS_ADRALN	Invalid address alignment
	BUS_ADRERR	Nonexistent physical address
	BUS_MCEERR_AO	Hardware memory error; action optional (since Linux 2.6.32)
	BUS_MCEERR_AR	Hardware memory error; action required (since Linux 2.6.32)
	BUS_OBJERR	Object-specific hardware error
SIGCHLD	CLD_CONTINUED	Child continued by SIGCONT (since Linux 2.6.9)
	CLD_DUMPED	Child terminated abnormally, with core dump
	CLD_EXITED	Child exited
	CLD_KILLED	Child terminated abnormally, without core dump
	CLD_STOPPED	Child stopped
	CLD_TRAPPED	Traced child has stopped
SIGFPE	FPE_FLTDIV	Floating-point divide-by-zero
	FPE_FLTINV	Invalid floating-point operation
	FPE_FLTOVF	Floating-point overflow
	FPE_FLTRES	Floating-point inexact result
	FPE_FLTUND	Floating-point underflow
	FPE_INTDIV	Integer divide-by-zero
	FPE_INTOVF	Integer overflow
	FPE_SUB	Subscript out of range
SIGILL	ILL_BADSTK	Internal stack error
	ILL_COPROC	Coprocessor error
	ILL_ILLADR	Illegal addressing mode
	ILL_ILLOPC	Illegal opcode
	ILL_ILLOPN	Illegal operand
	ILL_ILLTRP	Illegal trap
	ILL_PRVOPC	Privileged opcode
	ILL_PRVREG	Privileged register
SIGPOLL/ SIGIO	POLL_ERR	I/O error
	POLL_HUP	Device disconnected
	POLL_IN	Input data available
	POLL_MSG	Input message available
	POLL_OUT	Output buffers available
	POLL_PRI	High-priority input available
SIGSEGV	SEGV_ACCERR	Invalid permissions for mapped object
	SEGV_MAPERR	Address not mapped to object
SIGTRAP	TRAP_BRANCH	Process branch trap
	TRAP_BRKPT	Process breakpoint
	TRAP_HWBKPT	Hardware breakpoint/watchpoint
	TRAP_TRACE	Process trace trap

INTERRUPT AND RESTART SYSTEM CALL

- SA_RESTART
  - Scenario
    - We establish a handler for some signal
    - We call blocking system call like read on terminal
    - When the system call is blocked, signal delivered and signal handler is invoked
    - System call break and return error with EINTR and we prefer to continue the execution of interrupted system call
  - Don’t need to check errno EINTR case
  - System call restart automatically

```
while ((cnt = read(fd, buf, BUF_SIZE)) == -1 && errno == EINTR)
    continue;                   /* Do nothing loop body */

if (cnt == -1)                  /* read() failed with other than EINTR */
    errExit("read");
```

THE CORE DUMP FILE

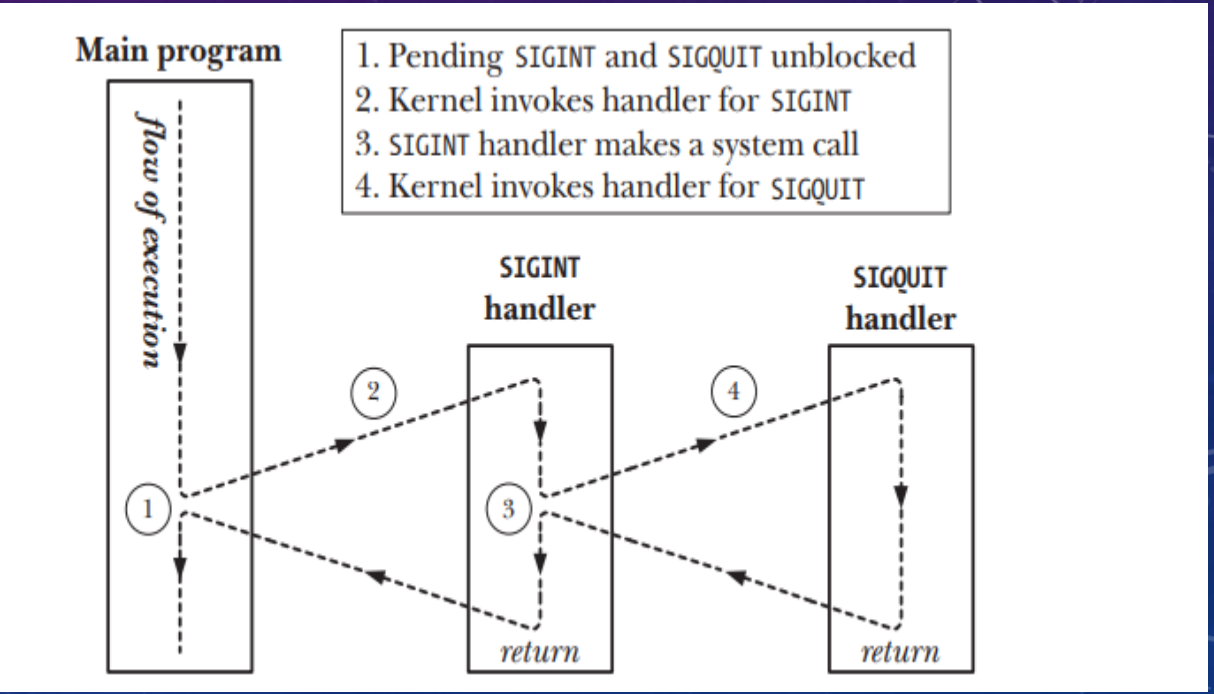
- Debug issue
  - Certain signal cause a process core dump and terminate
  - A core dump is file cont of process at the time it
  - This file loaded into the examine the state of a p the time signal arrive
  - Ctrl \ cause SIGQUIT sig
  - Core is generated at wo
  - /proc/sys/kernel/core_pattern

Specifier	Replaced by
%c	Core file size soft resource limit (bytes; since Linux 2.6.24)
%e	Executable filename (without path prefix)
%g	Real group ID of dumped process
%h	Name of host system
%p	Process ID of dumped process
%s	Number of signal that terminated process
%t	Time of dump, in seconds since the Epoch
%u	Real user ID of dumped process
%%	A single % character

- Some situation core dump is not generated
  - cess don't have permission to write
  - r file has same name exist
  - king directory is not exist
  - resource limit on the core dump size is 0
  - resource limit of the file process created
  - ut of space

TIMING AND ORDER OF SIGNAL DELIVERY

- Signal delivered
  - Hardware exception triggers a signal or a process raise signal to itself, signal delivered immediately
  - When signal send from process to process, it delay a little time because kernel only delivered signal to process when it is scheduled



SOME SPECIAL CASE

- SIGKILL and SIGSTOP
  - Cannot change default action for SIGKILL and SIGSTOP, SIGKILL always terminate process, SIGSTOP always stop the process
- SIGCONT
  - Always continue a stopped process event it blocked or ignored
- Hardware-generated signals (SIGBUS, SIGFPE, SIGILL, SIGSEGV)
  - The behavior of process is undefined if it returned from a handler for signal or if it ignored or blocked
  - It can be cause a loop of signal to process when return from signal handler
  - Linux force a delivery and terminate process event if process tries to block or ignore these signals
- It is highly recommend process terminate and produce a core file to find the reasons that cause these signals

REALTIME SIGNAL

```
lim = sysconf(_SC_SIGQUEUE_MAX);
```

- Realtime signal
  - Defined by POSIX, use for application defined purposes
  - Realtime signal is queue. If multiple realtime signal is sent to process, it is delivered multiple-time. In contrast, we send multiple-signal that pending already for process, it sent only once
  - When sending a realtime signal, it is possible to specific data that accompanies with signal
  - The order of delivery of realtime signal is guarrentee

- Limits on the number of queued realtime signals
  - Queueing realtime signal with associated data requires that the kernel maintain data structures listing the signal queued for each process
  - This will consume kernel memory and kernel places limit on the number of realtime signal that queued
  - Default is 32

SENDING AND HANDLING REALTIME SIGNAL

- Sending realtime signal with sigqueue
  - Same meaning with kill even when sig = 0
  - Value bring data accompany the signal
  - sigqueue error when number of signal in queue reach max
- Handling realtime signal
  - Sigaction with SA_SIGINFO

```
int sigqueue(pid_t pid, int sig, const union sigval value);
```

```
union sigval {
    int   sival_int;      /* Integer value for accompanying data */
    void *sival_ptr;      /* Pointer value for accompanying data */
};
```

FETCHING SIGNALS VIA A FILE DESCRIPTOR

- Signal handler as driven I/O
  - Signal represent by file descriptor
  - Can use with select
  - Read more information with signalfd_siginfo

```
struct signalfd_siginfo {
    uint32_t  ssi_signo;    /* Signal number */
    int32_t   ssi_errno;    /* Error number (generally unused) */
    int32_t   ssi_code;     /* Signal code */
    uint32_t  ssi_pid;      /* Process ID of sending process */
    uint32_t  ssi_uid;      /* Real user ID of sender */
    int32_t   ssi_fd;       /* File descriptor (SIGPOLL/SIGIO) */
    uint32_t  ssi_tid;      /* Kernel timer ID (POSIX timers) */
    uint32_t  ssi_band;     /* Band event (SIGPOLL/SIGIO) */
    uint32_t  ssi_tid;      /* (Kernel-internal) timer ID (POSIX timers) */
    uint32_t  ssi_overrun;  /* Overrun count (POSIX timers) */
    uint32_t  ssi_trapno;   /* Trap number */
    int32_t   ssi_status;   /* Exit status or signal (SIGCHLD) */
    int32_t   ssi_int;      /* Integer sent by sigqueue() */
    uint64_t  ssi_ptr;      /* Pointer sent by sigqueue() */
    uint64_t  ssi_utime;    /* User CPU time (SIGCHLD) */
    uint64_t  ssi_stime;    /* System CPU time (SIGCHLD) */
    uint64_t  ssi_addr;     /* Address that generated signal
                               (hardware-generated signals only) */
};
```

FUNCTION REVIEW

- signal
- Kill
- Raise
- Abort
- Pause
- sigqueue
- Sigprocmask
- Signalfd