

# Signals

Jaye-Jyh Tsai

Inst. Of CIS, NCTU

March 11, 2002



# Outline

---

- What are signals?
- Data Structures
- ***sigset\_t*** Functions
- Set Functions
- Sending Signals
- Other Signal-Related Functions

# Signals – What are signals? (1)

- A signal is a notification to a process that an event has occurred
- Signals can be sent
  - by one process to another process (or to itself)
  - by the kernel to a process
- Signals are a form of interprocess communication (IPC)

# Signals – What are signals? (2)

- Signals are practically useless for bidirectional communication
  - No elaborate message can be sent
  - No identity of the sender
- Within limits, the receiver is not obligated to respond in any way and can even entirely ignore most signals

# Signals – What are signals? (3)

- Five conditions that generate signals
  - The *kill* system call
  - The *kill* command
  - Certain terminal characters
    - Ctrl-C generates a SIGINT signal
  - Certain hardware conditions
    - Floating point arithmetic error generates a SIGFPE signal
  - Certain software conditions
    - The SIGURG signal is generated when out-of-band data arrive on a socket

# Signals – What are signals? (4)

include/asm-i386/signal.h

```
#define SIGHUP          1          #define SIGSTOP        19
#define SIGINT         2          #define SIGTSTP        20
#define SIGQUIT        3          #define SIGTTIN        21
#define SIGILL         4          #define SIGTTOU        22
#define SIGTRAP        5          #define SIGURG         23
#define SIGABRT        6          #define SIGXCPU        24
#define SIGIOT         6          #define SIGXFSZ        25
#define SIGBUS         7          #define SIGVTALRM      26
#define SIGFPE         8          #define SIGPROF        27
#define SIGKILL        9          #define SIGWINCH       28
#define SIGUSR1        10         #define SIGIO          29
#define SIGSEGV        11         #define SIGPOLL        SIGIO
#define SIGUSR2        12         /*
#define SIGPIPE       13         #define SIGLOST        29
#define SIGALRM       14         */
#define SIGTERM       15         #define SIGPWR         30
#define SIGSTKFLT     16         #define SIGSYS         31
#define SIGCHLD       17         #define SIGUNUSED      31
#define SIGCONT       18
```

# Signals – How to use?

user space application

```
#include <signal.h>
```

```
...
```

```
void sig_alarm(int sig)
```

```
/* signal handler */
```

```
{
```

```
    // do something to handle the signal
```

```
}
```

```
...
```

```
{
```

```
    struct sigaction sa;
```

```
    ...
```

```
    sa.sa_flags = SA_NOCLDSTOP;
```

```
/* turn off SIGCHLD */
```

```
    sa.sa_handler = sig_alarm;
```

```
/* set signal handler of this sigaction */
```

```
    sigaction(SIGALRM, &sa, NULL);
```

```
/* register this sigaction to handle SIGALRM */
```

```
    alarm(5);
```

```
/* send SIGALRM after 5 seconds */
```

```
    ...
```

```
}
```

# Signals – Data Structures (1)

```
include/asm-i386/signal.h
```

```
#define _NSIG                64  
#define _N_SIG_BPW          32  
#define _NSIG_WORDS         (_NSIG / _NSIG_BPW)
```

```
typedef struct {  
    unsigned long sig[_NSIG_WORDS];  
} sigset_t;
```

# Signals – Data Structures (1)

```
include/asm-i386/signal.h
```

```
typedef void (*__sig_handler_t)(int);
```

```
#define SIG_DFL ((__sig_handler_t)0) /* default signal handling */
```

```
#define SIG_IGN ((__sig_handler_t)1) /* ignore signal */
```

```
#define SIG_ERR ((__sig_handler_t)-1) /* error return value */
```

```
struct sigaction {  
    __sig_handler_t sa_handler;  
    unsigned long sa_flags;  
    void (*sa_restorer)(void);  
    sigset_t sa_mask;  
};
```

```
struct k_sigaction {  
    struct sigaction sa;  
};
```

```
include/asm-i386/signal.h
```

```
#define SA_NOCLDSTOP          0x00000001  
#define SA_NOCLDWAIT        0x00000002 /* not  
supported yet */
```

```
#define SA_SIGINFO           0x00000004
```

```
#define SA_ONSTACK           0x08000000
```

```
#define SA_RESTART           0x10000000
```

```
#define SA_NODEFER           0x40000000
```

```
#define SA_RESETHAND         0x80000000
```

```
#define SA_NOMASK            SA_NODEFER
```

```
#define SA_ONESHOT           SA_RESETHAND
```

```
#define SA_INTERRUPT         0x20000000 /*
```

```
dummy -- ignored */
```

```
#define SA_RESTORER          0x04000000
```

# Signals – Data Structures (2)

```
include/asm-i386/signinfo.h
```

```
typedef struct siginfo {  
    int si_signo;           /* signal number */  
    int si_errno;          /* sender's error value */  
    int si_code;           /* signal's source */  
  
    union {  
        int _pad[SI_PAD_SIZE];  
  
        /* kill() */  
        struct {  
            pid_t _pid;    /* sender's pid */  
            uid_t _uid;    /* sender's uid */  
        } _kill;  
        ...  
    } _sifields;  
} siginfo_t;
```

# Signals – Data Structures (2)

```
include/asm-i386/siginfo.h
```

```
typedef struct siginfo {
```

```
    int si_signo;           /* signal number */
    int si_errno;          /* sender's error value */
    int si_code;           /* signal's source */
```

```
union {
```

```
    #define SI_USER          0           /* sent by kill, sigsend, raise */
    #define SI_KERNEL       0x80        /* sent by the kernel from somewhere */
    #define SI_QUEUE        -1          /* sent by sigqueue */
    #define SI_TIMER        __SI_CODE(__SI_TIMER,-2) /* sent by timer expiration */
    #define SI_MESGQ        -3          /* sent by real time mesq state change */
    #define SI_ASYNCIO      -4          /* sent by AIO completion */
    #define SI_SIGIO        -5          /* sent by queued SIGIO */
```

```
    }
} siginfo_t;

#define SI_FROMUSER(siptr) ((siptr)->si_code <= 0)
#define SI_FROMKERNEL(siptr) ((siptr)->si_code > 0)
```

# Signals – Data Structures (2)

```
include/asm-i386/siginfo.h
```

```
typedef struct siginfo {
```

```
    int si_signo;
```

```
    int si_errno;
```

```
    int si_code;
```

```
    union {
```

```
        int _pad[SI_PAD_SIZE];
```

```
        /* kill() */
```

```
        struct {
```

```
            pid_t _pid;
```

```
            /* sender's pid */
```

```
            uid_t _uid;
```

```
            /* sender's uid */
```

```
        } _kill;
```

```
        ...
```

```
    } _sifields;
```

```
} siginfo_t;
```

```
include/asm-i386/siginfo.h
```

```
#define SI_MAX_SIZE
```

```
128
```

```
#define SI_PAD_SIZE
```

```
((SI_MAX_SIZE/sizeof(int)) - 3)
```

# Signals – Data Structures (3)

```
include/linux/signal.h
```

```
struct sigqueue
```

```
{
```

```
    struct sigqueue *next;
```

```
    siginfo_t info;
```

```
};
```

```
struct sigpending {
```

```
    struct sigqueue *head, **tail;
```

```
    sigset_t signal;
```

```
};
```

# Signals – Platform-Independent *sigset\_t* Functions (1)

include/linux/signal.h

```
static inline void sigaddset(sigset_t *set, int _sig)
```

```
{
```

```
    unsigned long sig = _sig - 1;
```

```
    if (_NSIG_WORDS == 1)
```

```
        set->sig[0] |= 1UL << sig;
```

```
    else
```

```
        set->sig[sig / _NSIG_BPW] |= 1UL << (sig % _NSIG_BPW);
```

```
}
```

```
static inline void sigdelset(sigset_t *set, int _sig)
```

```
{
```

```
    unsigned long sig = _sig - 1;
```

```
    if (_NSIG_WORDS == 1)
```

```
        set->sig[0] &= ~(1UL << sig);
```

```
    else
```

```
        set->sig[sig / _NSIG_BPW] &= ~(1UL << (sig % _NSIG_BPW));
```

```
}
```

```
    #if (_NSIG_WORDS == 1)
```

```
        set->sig[0] |= 1UL << sig;
```

```
    #else
```

```
        set->sig[sig / _NSIG_BPW] |= 1UL << (sig % _NSIG_BPW);
```

```
    #endif
```

# Signals – Platform-Independent *sigset\_t* Functions (2)

include/linux/signal.h

```
static inline int sigismember(sigset_t *set, int _sig)
{
    unsigned long sig = _sig - 1;
    if (_NSIG_WORDS == 1)
        return 1 & (set->sig[0] >> sig);
    else
        return 1 & (set->sig[sig / _NSIG_BPW] >> (sig % _NSIG_BPW));
}
```

```
static inline int sigfindinword(unsigned long word)
{
    return ffz(~word);
}
```

```
#define sigmask(sig)          (1UL << ((sig) - 1))
```

# Signals –

## Platform-Dependent *sigset\_t* Functions (1)

```
include/asm-i386/signal.h
```

```
static __inline__ void sigaddset(sigset_t *set, int _sig)
{
    __asm__("btsl %1,%0" : "=m"(*set) : "Ir"(_sig - 1) : "cc");
}
```

```
static __inline__ void sigdelset(sigset_t *set, int _sig)
{
    __asm__("btrl %1,%0" : "=m"(*set) : "Ir"(_sig - 1) : "cc");
}
```

```
#define sigmask(sig) (1UL << ((sig) - 1))
static __inline__ int sigfindinword(unsigned long word)
{
    __asm__("bsfl %1,%0" : "=r"(word) : "rm"(word) : "cc");
    return word;
}
```

# Signals – Platform-Dependent *sigset\_t* Functions (2)

```
include/asm-i386/signal.h
```

```
static __inline__ int __const_sigismember(sigset_t *set, int _sig)
{
    unsigned long sig = _sig - 1;
    return 1 & (set->sig[sig / _NSIG_BPW] >> (sig % _NSIG_BPW));
}
```

```
static __inline__ int __gen_sigismember(sigset_t *set, int _sig)
{
    int ret;
    __asm__ ("btl %2,%1\n\t;sbbl %0,%0"
            : "=r"(ret) : "m"(*set), "l"(_sig-1) : "cc");
    return ret;
}
```

```
#define sigismember(set,sig) \
    (__builtin_constant_p(sig) ? \
     __const_sigismember((set),(sig)) : \
     __gen_sigismember((set),(sig)))
```

compile-time operator that reports whether the value of its argument can be computed at compile time

# Signals – Set B

```
include/linux/signal.h
```

```
#define _SIG_SET_BINOP(name, op)
```

```
#define _sig_or(x,y) ((x) | (y))  
_SIG_SET_BINOP(sigorsets, _sig_or)
```

```
#define _sig_and(x,y) ((x) & (y))  
_SIG_SET_BINOP(sigandsets, _sig_and)
```

```
#define _sig_nand(x,y) ((x) & ~(y))  
_SIG_SET_BINOP(signanandsets, _sig_nand)
```

```
static inline void name(sigset_t *r, const sigset_t *a, const sigset_t *b) \  
{  
    unsigned long a0, a1, a2, a3, b0, b1, b2, b3;  
    unsigned long i;  
  
    for (i = 0; i < _NSIG_WORDS/4; ++i) {  
        a0 = a->sig[4*i+0]; a1 = a->sig[4*i+1];  
        a2 = a->sig[4*i+2]; a3 = a->sig[4*i+3];  
        b0 = b->sig[4*i+0]; b1 = b->sig[4*i+1];  
        b2 = b->sig[4*i+2]; b3 = b->sig[4*i+3];  
        r->sig[4*i+0] = op(a0, b0);  
        r->sig[4*i+1] = op(a1, b1);  
        r->sig[4*i+2] = op(a2, b2);  
        r->sig[4*i+3] = op(a3, b3);  
    }  
    switch (_NSIG_WORDS % 4) {  
        case 3:  
            a0 = a->sig[4*i+0]; a1 = a->sig[4*i+1]; a2 = a->sig[4*i+2];  
            b0 = b->sig[4*i+0]; b1 = b->sig[4*i+1]; b2 = b->sig[4*i+2];  
            r->sig[4*i+0] = op(a0, b0);  
            r->sig[4*i+1] = op(a1, b1);  
            r->sig[4*i+2] = op(a2, b2);  
            break;  
        case 2:  
            a0 = a->sig[4*i+0]; a1 = a->sig[4*i+1];  
            b0 = b->sig[4*i+0]; b1 = b->sig[4*i+1];  
            r->sig[4*i+0] = op(a0, b0);  
            r->sig[4*i+1] = op(a1, b1);  
            break;  
        case 1:  
            a0 = a->sig[4*i+0]; b0 = b->sig[4*i+0];  
            r->sig[4*i+0] = op(a0, b0);  
            break;  
    }  
}
```

# Signals – Set

```
include/linux/signal.h
```

```
#define _SIG_SET_BINOP(name, op)
```

```
#define _sig_or(x,y) ((x) | (y))  
_SIG_SET_BINOP(sigorsets, _sig_or)
```

```
#define _sig_and(x,y) ((x) & (y))  
_SIG_SET_BINOP(sigandsets, _sig_and)
```

```
#define _sig_nand(x,y) ((x) & ~(y))  
_SIG_SET_BINOP(signandsets, _sig_nand)
```

```
static inline void name(sigset_t *r, const sigset_t *a, const sigset_t *b) \  
{  
    unsigned long a0, a1, a2, a3, b0, b1, b2, b3;  
    unsigned long i;  
  
    for (i = 0; i < _NSIG_WORDS/4; ++i) {  
        a0 = a->sig[4*i+0]; a1 = a->sig[4*i+1];  
        a2 = a->sig[4*i+2]; a3 = a->sig[4*i+3];  
        b0 = b->sig[4*i+0]; b1 = b->sig[4*i+1];  
        b2 = b->sig[4*i+2]; b3 = b->sig[4*i+3];  
        r->sig[4*i+0] = op(a0, b0);  
        r->sig[4*i+1] = op(a1, b1);  
        r->sig[4*i+2] = op(a2, b2);  
        r->sig[4*i+3] = op(a3, b3);  
    }  
    switch (_NSIG_WORDS % 4) {  
        case 3:  
            a2 = a->sig[4*i+2];  
            b2 = b->sig[4*i+2];  
            r->sig[4*i+2] = op(a2, b2);  
        case 2:  
            a1 = a->sig[4*i+1];  
            b1 = b->sig[4*i+1];  
            r->sig[4*i+1] = op(a1, b1);  
        case 1:  
            a0 = a->sig[4*i+0]; b0 = b->sig[4*i+0];  
            r->sig[4*i+0] = op(a0, b0);  
    }  
}
```

# Signals – Set Functions (2)

include/linux/signal.h

```
#define _SIG_SET_OP(name, op) \
static inline void name(sigset_t *set) \
{ \
    unsigned long i; \
 \
    for (i = 0; i < _NSIG_WORDS/4; ++i) { \
        set->sig[4*i+0] = op(set->sig[4*i+0]); \
        set->sig[4*i+1] = op(set->sig[4*i+1]); \
        set->sig[4*i+2] = op(set->sig[4*i+2]); \
        set->sig[4*i+3] = op(set->sig[4*i+3]); \
    } \
    switch (_NSIG_WORDS % 4) { \
        case 3: set->sig[4*i+2] = op(set->sig[4*i+2]); \
        case 2: set->sig[4*i+1] = op(set->sig[4*i+1]); \
        case 1: set->sig[4*i+0] = op(set->sig[4*i+0]); \
    } \
} \

#define _sig_not(x) (~(x))
_SIG_SET_OP(signotset, _sig_not)
```

# Signals – Set Functions (3)

include/linux/signal.h

```
static inline void sigemptyset(sigset_t *set)
{
    switch (_NSIG_WORDS) {
    default:
        memset(set, 0, sizeof(sigset_t));
        break;
    case 2: set->sig[1] = 0;
    case 1: set->sig[0] = 0;
        break;
    }
}
```

```
static inline void sigfillset(sigset_t *set)
{
    switch (_NSIG_WORDS) {
    default:
        memset(set, -1, sizeof(sigset_t));
        break;
    case 2: set->sig[1] = -1;
    case 1: set->sig[0] = -1;
        break;
    }
}
```

# Signals – Set Functions (4)

include/linux/signal.h

```
static inline void sigaddsetmask(sigset_t *set, unsigned long mask)
{
    set->sig[0] |= mask;
}
```

```
static inline void sigdelsetmask(sigset_t *set, unsigned long mask)
{
    set->sig[0] &= ~mask;
}
```

```
static inline int sigtestsetmask(sigset_t *set, unsigned long mask)
{
    return (set->sig[0] & mask) != 0;
}
```

# Signals – Set Functions (5)

include/linux/signal.h

```
static inline void siginitset(sigset_t *set, unsigned long mask)
{
    set->sig[0] = mask;
    switch (_NSIG_WORDS) {
    default:
        memset(&set->sig[1], 0, sizeof(long)*(_NSIG_WORDS-1));
        break;
    case 2: set->sig[1] = 0;
    case 1: ;
    }
}

static inline void siginitsetinv(sigset_t *set, unsigned long mask)
{
    set->sig[0] = ~mask;
    switch (_NSIG_WORDS) {
    default:
        memset(&set->sig[1], -1, sizeof(long)*(_NSIG_WORDS-1));
        break;
    case 2: set->sig[1] = -1;
    case 1: ;
    }
}
```

# Signals – Sending Signals (1)

kernel/signal.c

```
asmlinkage long sys_kill(int pid, int sig)
{
    struct siginfo info;

    info.si_signo = sig;
    info.si_errno = 0;
    info.si_code = SI_USER;
    info.si_pid = current->pid;
    info.si_uid = current->uid;

    return kill_something_info(sig, &info, pid);
}
```

```
sys_kill() → kill_something_info()
```

## Signals – Sending Signals (2)

kernel/signal.c

```
static int kill_something_info(int sig, struct siginfo *info, int pid)
{
    if (!pid) {
        return kill_pg_info(sig, info, current->pgrp);
    } else if (pid == -1) {
        int retval = 0, count = 0;
        struct task_struct * p;

        read_lock(&tasklist_lock);
        for_each_task(p) {
            if (p->pid > 1 && p != current) {
                int err = send_sig_info(sig, info, p);
                ++count;
            }
            ...
        }
    }
}
```

*/\* If pid is 0, the current process wants \*/  
/\* to send the signal to its entire process \*/  
/\* group. \*/*

```
sys_kill() → kill_something_info()
```

## Signals – Sending Signals (2)

kernel/signal.c

```
static int kill_something_info(int sig, struct siginfo *info, int pid)
```

```
{
```

```
...
```

```
} else if (pid == -1) {
```

```
    int retval = 0, count = 0;
```

```
    struct task_struct * p;
```

```
    read_lock(&tasklist_lock);
```

```
    for_each_task(p) {
```

```
        if (p->pid > 1 && p != current) {
```

```
            int err = send_sig_info(sig, info, p);
```

```
            ++count;
```

```
...
```

```
} else if (pid < 0) {
```

```
...
```

```
}
```

```
include/linux/sched.h
```

```
#define for_each_task(p) \
```

```
    for (p = &init_task ; (p = p->next_task) != &init_task ; )
```

```
/* Each task with pid larger than 1 and is */  
/* not the current process will be signaled.*/
```

```
sys_kill() → kill_something_info()
```

## Signals – Sending Signals (2)

kernel/signal.c

```
static int kill_something_info(int sig, struct siginfo *info, int pid)
{
    ...
        if (err != -EPERM)
            retval = err;
    }
}
read_unlock(&tasklist_lock);
return count ? retval : -ESRCH;
} else if (pid < 0) {
    return kill_pg_info(sig, info, -pid);
} else {
    return kill_proc_info(sig, info, pid);
}
}
```

```
/* If pid is negative but not -1, a process */
/* group with the absolute value of pid as*/
/* its process group number will be      */
/* signaled.                               */
```

```
sys_kill() → kill_something_info()
```

## Signals – Sending Signals (2)

kernel/signal.c

```
static int kill_something_info(int sig, struct siginfo *info, int pid)
{
    ...
        if (err != -EPERM)
            retval = err;
    }
}
read_unlock(&tasklist_lock);
return count ? retval : -ESRCH;
} else if (pid < 0) {
    return kill_pg_info(sig, info, -pid);
} else {
    return kill_proc_info(sig, info, pid);
}
}
```

/\* If pid is positive, a single process with \*/  
/\* pid as its process ID will be singled. \*/

sys\_kill() → kill\_something\_info() → kill\_pg\_info()

## Signals – Sending Signals (3)

kernel/signal.c

```
int kill_pg_info(int sig, struct siginfo *info, pid_t pgrp)
{
    int retval = -EINVAL;
    if (pgrp > 0) {
        struct task_struct *p;

        retval = -ESRCH;
        read_lock(&tasklist_lock);
        for_each_task(p) {
            if (p->pgrp == pgrp) {
                int err = send_sig_info(sig, info, p);
                if (retval)
                    retval = err;
            }
        }
        read_unlock(&tasklist_lock);
    }
    return retval;
}
```

```
sys_kill() → kill_something_info() ↗ kill_pg_info()  
→ kill_proc_info()
```

## Signals – Sending Signals (4)

kernel/signal.c

```
inline int kill_proc_info(int sig, struct siginfo *info, pid_t pid)
{
    int error;
    struct task_struct *p;

    read_lock(&tasklist_lock);
    p = find_task_by_pid(pid);
    error = -ESRCH;
    if (p)
        error = send_sig_info(sig, info, p);
    read_unlock(&tasklist_lock);
    return error;
}
```

include/linux/sched.h

```
static inline struct task_struct *find_task_by_pid(int pid)
{
    struct task_struct *p, **htable = &pidhash[pid_hashfn(pid)];

    for(p = *htable; p && p->pid != pid; p = p->pidhash_next)
        ;

    return p;
}
```

```
sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info()
```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
{
    unsigned long flags;
    int ret;

    ret = -EINVAL;
    if (sig < 0 || sig > _NSIG)
        goto out_noblock;

    ret = -EPERM;
    if (bad_signal(sig, info, t))
        goto out_noblock;

    ...
}
```

kernel/signal.c

```
int bad_signal(int sig, struct siginfo *info, struct task_struct *t)
{
    return (!info || ((unsigned long)info != 1 && SI_FROMUSER(info)))
        && ((sig != SIGCONT) || (current->session != t->session))
        && (current->euid ^ t->suid) && (current->euid ^ t->uid)
        && (current->uid ^ t->suid) && (current->uid ^ t->uid)
        && !capable(CAP_KILL);
}
```

```
sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
{
    ...
    ret = 0;
    if (!sig || !t->sig)
        goto out_nolock;

    spin_lock_irqsave(&t->sigmask_lock, flags);
    handle_stop_signal(sig, t);

    if (ignored_signal(sig, t))
        goto out;

    if (sig < SIGRTMIN && sigismember(&t->sigmask, sig))
        goto out;

    ret = deliver_signal(sig, info, t);
    ...
}
```

kernel/signal.c

```
static void handle_stop_signal(int sig, struct task_struct *t)
{
    switch (sig) {
    case SIGKILL: case SIGCONT:
        /* Wake up the process if stopped. */
        if (t->state == TASK_STOPPED)
            wake_up_process(t);
        t->exit_code = 0;
        rm_sig_from_queue(SIGSTOP, t);
        rm_sig_from_queue(SIGTSTP, t);
        rm_sig_from_queue(SIGTTOU, t);
        rm_sig_from_queue(SIGTTIN, t);
        break;

    case SIGSTOP: case SIGTSTP:
    case SIGTTIN: case SIGTTOU:
        /* If we're stopping again, cancel SIGCONT */
        rm_sig_from_queue(SIGCONT, t);
        break;
    }
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal()

```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
...
```

```
ret = 0;
```

```
if (!sig || !t->sig)
```

```
    goto out_nolock;
```

```
spin_lock_irqsave(&t->sigmask_lock, &fl);
```

```
handle_stop_signal(sig, t);
```

```
if (ignored_signal(sig, t))
```

```
    goto out;
```

```
if (sig < SIGRTMIN && sigismember(&t->pending.signal, sig))
```

```
    goto out;
```

```
ret = deliver_signal(sig, info, t);
```

```
...
```

```
}
```

kernel/signal.c

```
static int ignored_signal(int sig, struct task_struct *t)
```

```
{
```

```
    /* Don't ignore traced or blocked signals */
```

```
    if ((t->ptrace & PT_PTRACED) || sigismember(&t->blocked, sig))
```

```
        return 0;
```

```
    return signal_type(sig, t->sig) == 0;
```

```
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()

```

# Signals – S

kernel/signal.c

```

int send_sig_info(int sig, struct tinfo *t)
{
    ...
    ret = 0;
    if (!sig || !t->sig)
        goto out_nolock;

    spin_lock_irqsave(&t->siglock, &flags);
    handle_stop_signal(sig, t);

    if (ignored_signal(sig, t))
        goto out;

    if (sig < SIGRTMIN && sig > SIGRTMAX)
        goto out;

    ret = deliver_signal(sig, info);
    ...
}

```

```

static int signal_type(int sig, struct signal_struct *signals)
{
    unsigned long handler;

    if (!signals)
        return 0;

    handler = (unsigned long) signals->action[sig-1].sa.sa_handler;
    if (handler > 1)
        return 1;

    /* "Ignore" handler.. Illogical, but that has an implicit handler for SIGCHLD */
    if (handler == 1)
        return sig == SIGCHLD;

    /* Default handler. Normally lethal, but.. */
    switch (sig) {

        /* Ignored */
        case SIGCONT: case SIGWINCH:
        case SIGCHLD: case SIGURG:
            return 0;

        /* Implicit behaviour */
        case SIGTSTP: case SIGTTIN: case SIGTTOU:
            return 1;

        /* Implicit actions (kill or do special stuff) */
        default:
            return -1;
    }
}

```

```
sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()
```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
{
    ...
    ret = 0;
    if (!sig || !t->sig)
        goto out_nolock;

    spin_lock_irqsave(&t->sigmask_lock, flags);
    handle_stop_signal(sig, t);

    if (ignored_signal(sig, t))
        goto out;

    if (sig < SIGRTMIN && sigismember(&t->pending.signal, sig))
        goto out;

    ret = deliver_signal(sig, info, t);
    ...
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()
                                                         → deliver_signal()

```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
...
```

```
ret = 0;
```

```
if (!sig || !t->sig)
```

```
    goto out_nolock;
```

```
spin_lock_irqsave(&t->siglock,
```

```
handle_stop_signal(sig,
```

```
if (ignored_signal(sig,
```

```
    goto out;
```

```
if (sig < SIGRTMIN &&
```

```
    goto out;
```

```
ret = deliver_signal(sig, info, t);
```

```
...
```

```
}
```

kernel/signal.c

```
static int deliver_signal(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
    int retval = send_signal(sig, info, &t->pending);
```

```
    if (!retval && !sigismember(&t->blocked, sig))
```

```
        signal_wake_up(t);
```

```
    return retval;
```

```
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()
                                                         → deliver_signal() → send_signal()

```

# Signal

kernel/signal.c

```
static int send_signal(int sig, struct siginfo *info, struct sigpending *signals)
```

```

{
    struct sigqueue *q = NULL;
    if (atomic_read(&nr_queued_signals) < max_queued_signals) {
        q = kmem_cache_alloc(sigqueue_cachep, GFP_ATOMIC);
    }
    if (q) {
        ...
        switch ((unsigned long) info) {
            case 0:
                q->info.si_signo = sig; q->info.si_errno = 0; q->info.si_code = SI_USER;
                q->info.si_pid = current->pid; q->info.si_uid = current->uid;
                break;
            case 1:
                q->info.si_signo = sig; q->info.si_errno = 0; q->info.si_code = SI_KERNEL;
                q->info.si_pid = 0; q->info.si_uid = 0;
                break;
            default:
                copy_siginfo(&q->info, info);
                break;
        }
    } else if (sig >= SIGRTMIN && info && (unsigned long)info != 1
               && info->si_code != SI_USER) {
        return -EAGAIN;
    }
    sigaddset(&signals->signal, sig);
    return 0;
}

```

kernel/signal.c

```
int send_sig_
```

```
{
```

```
...
```

```
ret = 0;
```

```
if (!sig ||
```

```
got
```

```
spin_loc
```

```
handle_s
```

```
if (ignore
```

```
got
```

```
if (sig <
```

```
got
```

```
ret = deliver_
```

```
...
```

```
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()
                                                         → deliver_signal() → send_signal()
                                                         → signal_wake_up()

```

## Signals – Sending Signals (5)

kernel/signal.c

```
int send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
...
```

```
ret = 0;
```

```
if (!sig || !t->sig)
```

```
    goto out_nolock;
```

```
spin_lock_irqsave(&t->siglock,
```

```
handle_stop_signal(sig,
```

```
if (ignored_signal(sig,
```

```
    goto out;
```

```
if (sig < SIGRTMIN &&
```

```
    goto out;
```

```
ret = deliver_signal(sig, info, t);
```

```
...
```

```
}
```

kernel/signal.c

```
static int deliver_signal(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
    int retval = send_signal(sig, info, &t->pending);
```

```
    if (!retval && !sigismember(&t->blocked, sig))
```

```
        signal_wake_up(t);
```

```
    return retval;
```

```
}
```

```

sys_kill() → kill_something_info() → kill_pg_info() → send_sig_info() → bad_signal()
                                     → kill_proc_info() → handle_stop_signal()
                                                         → ignored_signal() → signal_type()
                                                         → deliver_signal() → send_signal()
                                                         → signal_wake_up()

```

# Signals – Ser

kernel/signal.c

```

int send_sig_info(int sig, struct task_struct *t)
{
    ...
    ret = 0;
    if (!sig || !t->sig)
        goto out_nolock;

    spin_lock_irqsave(&t->siglock, flags);
    handle_stop_signal(sig, t);

    if (ignored_signal(sig, t))
        goto out;

    if (sig < SIGRTMIN && sig > SIGRTMAX)
        goto out;

    ret = deliver_signal(sig, info, t);
    ...
}

```

kernel/signal.c

```

static inline void signal_wake_up(struct task_struct *t)
{
    t->sigpending = 1;

#ifdef CONFIG_SMP
    /*
     * If the task is running on a different CPU
     * force a reschedule on the other CPU to make
     * it notice the new signal quickly.
     *
     * The code below is a tad loose and might occasionally
     * kick the wrong CPU if we catch the process in the
     * process of changing - but no harm is done by that
     * other than doing an extra (lightweight) IPI interrupt.
     */
    spin_lock(&runqueue_lock);
    if (t->has_cpu && t->processor != smp_processor_id())
        smp_send_reschedule(t->processor);
    spin_unlock(&runqueue_lock);
#endif /* CONFIG_SMP */

    if (t->state & TASK_INTERRUPTIBLE) {
        wake_up_process(t);
        return;
    }
}

```

# Signals – Sending Signals (6)

kernel/signal.c

```
int force_sig_info(int sig, struct siginfo *info, struct task_struct *t)
```

```
{
```

```
    unsigned long int flags;
```

```
    spin_lock_irqsave(&t->sigmask_lock, flags);
```

```
    if (t->sig == NULL) {
```

```
        spin_unlock_irqrestore(&t->sigmask_lock, flags);
```

```
        return -ESRCH;
```

```
    }
```

```
    if (t->sig->action[sig-1].sa_handler) {
```

```
        t->sig->action[sig-1].sa_handler(t, info, sig);
```

```
        sigdelset(&t->blocked, sig);
```

```
        recalc_sigpending(t);
```

```
        spin_unlock_irqrestore(&t->sigmask_lock, flags);
```

```
        return send_sig_info(sig, info, t);
```

```
    }
```

```
include/linux/sched.h
```

```
static inline void recalc_sigpending(struct task_struct *t)
```

```
{
```

```
    t->sigpending = has_pending_signals(&t->pending.signal, &t->blocked);
```

```
}
```

# Signals – Se

kernel/signal.c

```
int force_sig_info(int sig, struct k_sigaction *act, struct task_struct *t)
{
    unsigned long int flags;

    spin_lock_irqsave(&t->siglock, flags);
    if (t->sig == NULL) {
        spin_unlock_irqrestore(&t->siglock, flags);
        return -ESRCH;
    }

    if (t->sig->action[sig-1].sa_handler == NULL)
        t->sig->action[sig-1].sa_handler = SIG_DFL;
    sigdelset(&t->blocked, sig);
    recalc_sigpending(t);
    spin_unlock_irqrestore(&t->siglock, flags);

    return send_sig_info(sig, act, t);
}
```

include/linux/sched.h

```
static inline int has_pending_signals(sigset_t *signal, sigset_t *blocked)
{
    unsigned long ready;
    long i;

    switch (_NSIG_WORDS) {
    default:
        for (i = _NSIG_WORDS, ready = 0; --i >= 0 ; )
            ready |= signal->sig[i] &~ blocked->sig[i];
        break;

    case 4: ready = signal->sig[3] &~ blocked->sig[3];
            ready |= signal->sig[2] &~ blocked->sig[2];
            ready |= signal->sig[1] &~ blocked->sig[1];
            ready |= signal->sig[0] &~ blocked->sig[0];
            break;

    case 2: ready = signal->sig[1] &~ blocked->sig[1];
            ready |= signal->sig[0] &~ blocked->sig[0];
            break;

    case 1: ready = signal->sig[0] &~ blocked->sig[0];
            }
    return ready != 0;
}
```

# Signals – Sending Signals (7)

arch/i386/kernel/entry.S

```
...
ENTRY(ret_from_sys_call)
    cli                                     # need_resched and signals atomic test
    cmpl $0,need_resched(%ebx)
    jne reschedule
    cmpl $0,sigpending(%ebx)
    jne signal_return
restore_all:
    RESTORE_ALL

signal_return:
    sti                                     # we can get here from an interrupt handler
    testl $(VM_MASK),EFLAGS(%esp)
    movl %esp,%eax
    jne v86_signal_return
    xorl %edx,%edx
    call SYMBOL_NAME(do_signal)
    jmp restore_all
...
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t *oldset)
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmask_lock);
        signr = dequeue_signal(&current->blocked, &info);
        spin_unlock_irq(&current->sigmask_lock);

        if (!signr)
            break;
        ...
    }
    ...
}
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmask);
        signr = dequeue_signal(&current->sigmask, &info);
        spin_unlock_irq(&current->sigmask);

        if (!signr)
            break;
        ...
    }
    ...
}
```

kernel/signal.c

```
int dequeue_signal(sigset_t *mask, siginfo_t *info)
{
    int sig = 0;

    sig = next_signal(current, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    current->sigpending = 0;
                    return 0;
                }
            }
        }

        if (!collect_signal(sig, &current->pending, info))
            sig = 0;
    }
    recalc_sigpending(current);

    return sig;
}
```

# Signals – Send

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t *mask)
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmask);
        signr = dequeue_signal(&current, mask);
        spin_unlock_irq(&current->sigmask);

        if (!signr)
            break;

        ...
    }
    ...
}
```

kernel/signal.c

```
static int next_signal(struct task_struct *tsk, sigset_t *mask)
{
    unsigned long i, *s, *m, x;
    int sig = 0;

    s = tsk->pending.signal.sig;
    m = mask->sig;
    switch (_NSIG_WORDS) {
    default:
        for (i = 0; i < _NSIG_WORDS; ++i, ++s, ++m)
            if ((x = *s & ~ *m) != 0) {
                sig = ffz(~x) + i * _NSIG_BPW + 1;
                break;
            }
        break;

    case 2: if ((x = s[0] & ~ m[0]) != 0)
            sig = 1;
            else if ((x = s[1] & ~ m[1]) != 0)
                sig = _NSIG_BPW + 1;
            else
                break;
            sig += ffz(~x);
            break;

    case 1: if ((x = *s & ~ *m) != 0)
            sig = ffz(~x) + 1;
            break;
    }
}
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmask);
        signr = dequeue_signal(&current->sigmask, &info);
        spin_unlock_irq(&current->sigmask);

        if (!signr)
            break;
        ...
    }
    ...
}
```

kernel/signal.c

```
int dequeue_signal(sigset_t *mask, siginfo_t *info)
{
    int sig = 0;

    sig = next_signal(current, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    current->sigpending = 0;
                    return 0;
                }
            }
        }

        if (!collect_signal(sig, &current->pending, info))
            sig = 0;
    }

    recalc_sigpending(current);

    return sig;
}
```

# Signals – Sending

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t *sigset)
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmask);
        signr = dequeue_signal(&current->sigmask, sigset);
        spin_unlock_irq(&current->sigmask);

        if (!signr)
            break;
        ...
    }
    ...
}
```

kernel/signal.c

```
static int collect_signal(int sig, struct sigpending *list, siginfo_t *info)
{
    if (sigismember(&list->signal, sig)) {
        /* Collect the siginfo appropriate to this signal. */
        struct sigqueue *q, **pp;
        pp = &list->head;
        while ((q = *pp) != NULL) {
            if (q->info.si_signo == sig)
                goto found_it;
            pp = &q->next;
        }

        /* Ok, it wasn't in the queue. We must have
           been out of queue space. So zero out the
           info. */
        sigdelset(&list->signal, sig);
        info->si_signo = sig;
        info->si_errno = 0;
        info->si_code = 0;
        info->si_pid = 0;
        info->si_uid = 0;
        return 1;
    }

found_it:
    ...
}
```

# Signals – Sending

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t
{
    siginfo_t info;
    struct k_sigaction *ka;
    ...
    for (;;) {
        unsigned long signr;

        spin_lock_irq(&current->sigmas
        signr = dequeue_signal(&current
        spin_unlock_irq(&current->sigm

        if (!signr)
            break;
        ...
    }
    ...
}
```

kernel/signal.c

```
static int collect_signal(int sig, struct sigpending *list, siginfo_t *info)
{
    if (sigismember(&list->signal, sig)) {
        ...
    found_it:
        if ((*pp = q->next) == NULL)
            list->tail = pp;

        /* Copy the sigqueue information and free the queue entry */
        copy_siginfo(info, &q->info);
        kmem_cache_free(sigqueue_cache,q);
        atomic_dec(&nr_queued_signals);

        /* Non-RT signals can exist multiple times.. */
        if (sig >= SIGRTMIN) {
            while ((q = *pp) != NULL) {
                if (q->info.si_signo == sig)
                    goto found_another;
                pp = &q->next;
            }
        }

        sigdelset(&list->signal, sig);
    found_another:
        return 1;
    }
    return 0;
}
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t *oldset)
```

```
{
```

```
...
```

```
for (;;) {
```

```
...
```

```
if ((current->ptrace & PT_PTRACED) &&
```

```
    /* Let the debugger run. */
```

```
    current->exit_code = signr;
```

```
    current->state = TASK_STOPPED;
```

```
    notify_parent(current, SIGCHLD);
```

```
    schedule();
```

```
...
```

```
/* If the (new) signal is now blocked, requeue it. */
```

```
if (sigismember(&current->blocked, signr)) {
```

```
    send_sig_info(signr, &info, current);
```

```
    continue;
```

```
}
```

```
...
```

```
}
```

kernel/signal.c

```
void
```

```
notify_parent(struct task_struct *tsk, int sig)
```

```
{
```

```
    read_lock(&tasklist_lock);
```

```
    do_notify_parent(tsk, sig);
```

```
    read_unlock(&tasklist_lock);
```

```
}
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs, sigset_t *oldset)
{
    ...
    for (;;) {
        ...
        ka = &current->sig->action[signr-1];
        if (ka->sa.sa_handler == SIG_IGN) {
            ...
        }
        if (ka->sa.sa_handler == SIG_DFL) {
            ...
        }
        ...
        handle_signal(signr, ka, &info, oldset, regs);
        return 1;
    }
    ...
}
```

# Signals – Sending Signals (8)

arch/i386/kernel/signal.c

```
int do_signal(struct pt_regs *regs)
{
    ...
    for (;;) {
        ...
        ka = &current->sig->action[0];
        if (ka->sa.sa_handler)
            ...
        }
        if (ka->sa.sa_handler)
            ...
        }
        ...
        handle_signal(sig);
        return 1;
    }
    ...
}
```

arch/i386/kernel/signal.c

```
static void handle_signal(unsigned long sig, struct k_sigaction *ka,
                          siginfo_t *info, sigset_t *oldset, struct pt_regs *regs)
{
    ...
    /* Set up the stack frame */
    if (ka->sa.sa_flags & SA_SIGINFO)
        setup_rt_frame(sig, ka, info, oldset, regs);
    else
        setup_frame(sig, ka, oldset, regs);

    if (ka->sa.sa_flags & SA_ONESHOT)
        ka->sa.sa_handler = SIG_DFL;

    if (!(ka->sa.sa_flags & SA_NODEFER)) {
        spin_lock_irq(&current->sigmask_lock);
        sigorsets(&current->blocked, &current->blocked, &ka->sa.sa_mask);
        sigaddset(&current->blocked, sig);
        recalc_sigpending(current);
        spin_unlock_irq(&current->sigmask_lock);
    }
}
```

# Signals – Sending Signals (9)

arch/i386/kernel/entry.S

...

ENTRY(ret\_from\_sys\_call)

cli

# need\_resched and signals atomic test

cmpl \$0,need\_resched(%ebx)

jne reschedule

cmpl \$0,sigpending(%ebx)

jne signal\_return

restore\_all:

RESTORE\_ALL

signal\_return:

sti

# we can get here from an interrupt handler

testl \$(VM\_MASK),EFLAGS(%esp)

movl %esp,%eax

jne v86\_signal\_return

xorl %edx,%edx

call SYMBOL\_NAME(do\_signal)

jmp restore\_all

...

# Signals – Other Signal-Related Functions

- `sys_sigpending`
  - The process ask whether any nonrealtime signals arrived while they were blocked by this system call
- `sys_sigaction`
  - This system call associates an action with a signal, so that the action is performed when the process receives the signal
- `sys_rt_sigtimedwait`
  - This system call waits for a signal to arrive, optionally timing out after a specified interval