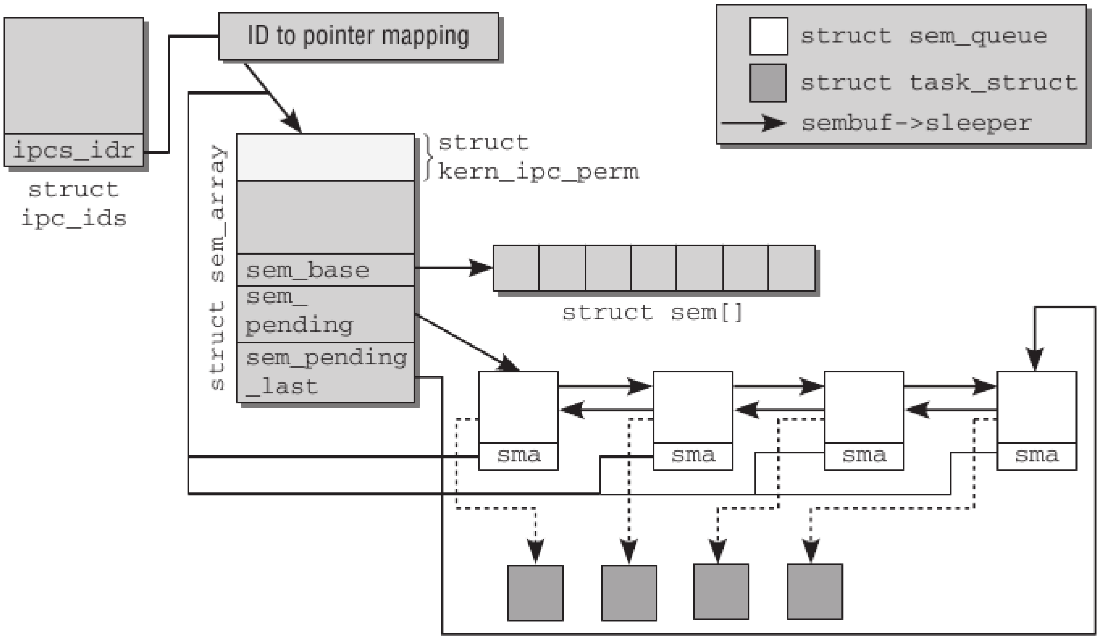
Interprocess communication and signals

semaphores(SYS V)



Data-structures involved with semaphores

<linux/sem.h>

```
struct sem_array {
        struct kern_ipc_perm sem_perm;      /* permissions .. see ipc.h */
        time_t               sem_otime;     /* last semop time */
        time_t               sem_ctime;     /* last change time */
        struct sem           *sem_base;     /* ptr to first semaphore in array */
        struct sem_queue     *sem_pending;  /* pending operations to be processed */
        struct sem_queue     **sem_pending_last; /* last pending operation */
        struct sem_undo      *undo;         /* undo requests on this array */
        unsigned long        sem_nsems;     /* no. of semaphores in array */
};
```

```
<linux/sem.h>
struct sem {
      int  semval; /* current value */
      int  sempid; /* pid of last operation */
};
```

```
<linux/sem.h>
struct sem_queue {
      struct sem_queue *  next;    /* next entry in the queue */
      struct sem_queue ** prev;    /* previous entry in the queue, *(q->prev) == q */
      struct task_struct* sleeper; /* this process */
      struct sem_undo *   undo;    /* undo structure */
      int                 pid;     /* process id of requesting process */
      int                 status;  /* completion status of operation */
      struct sem_array *  sma;     /* semaphore array for operations */
      int                 id;      /* internal sem id */
      struct sembuf *     sops;    /* array of pending operations */
      int                 nsops;   /* number of operations */
      int                 alter;   /* does the operation alter the array? */
};
```

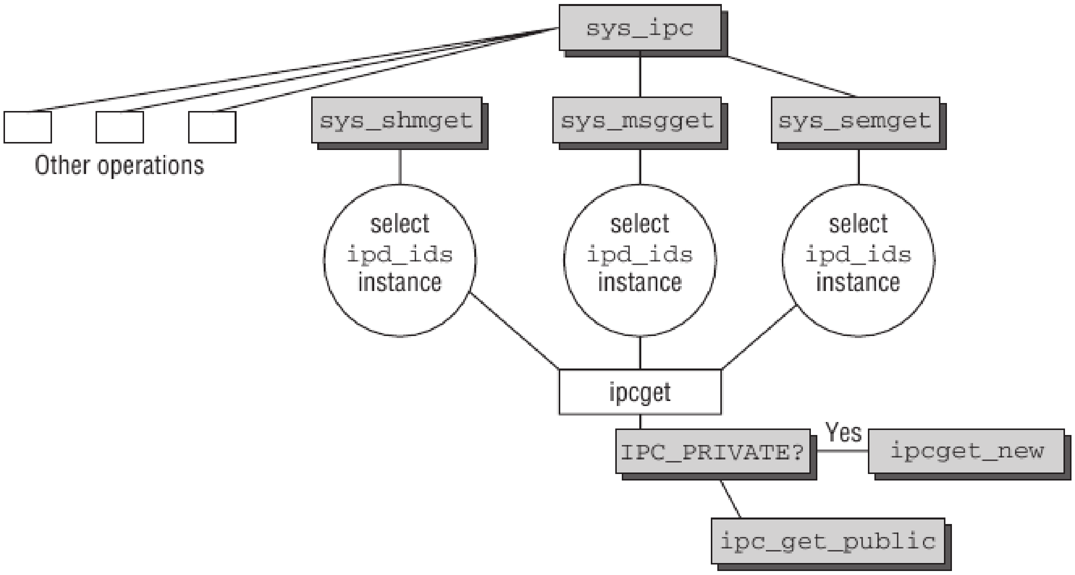
```
<linux/ipc.h>
struct kern_ipc_perm
{
      int          id;
      key_t        key;
      uid_t        uid;
      gid_t        gid;
      uid_t        cuid;
      gid_t        cgid;
      mode_t       mode;
      unsigned long seq;
};
```

struct ipc_ids is defined as follows:

```
   ipc/util.h
   struct ipc_ids {
           int in_use;
           unsigned short seq;
           unsigned short seq_max;
           struct rw_semaphore rw_mutex;
           struct idr ipcs_idr;
   };
```

slide3

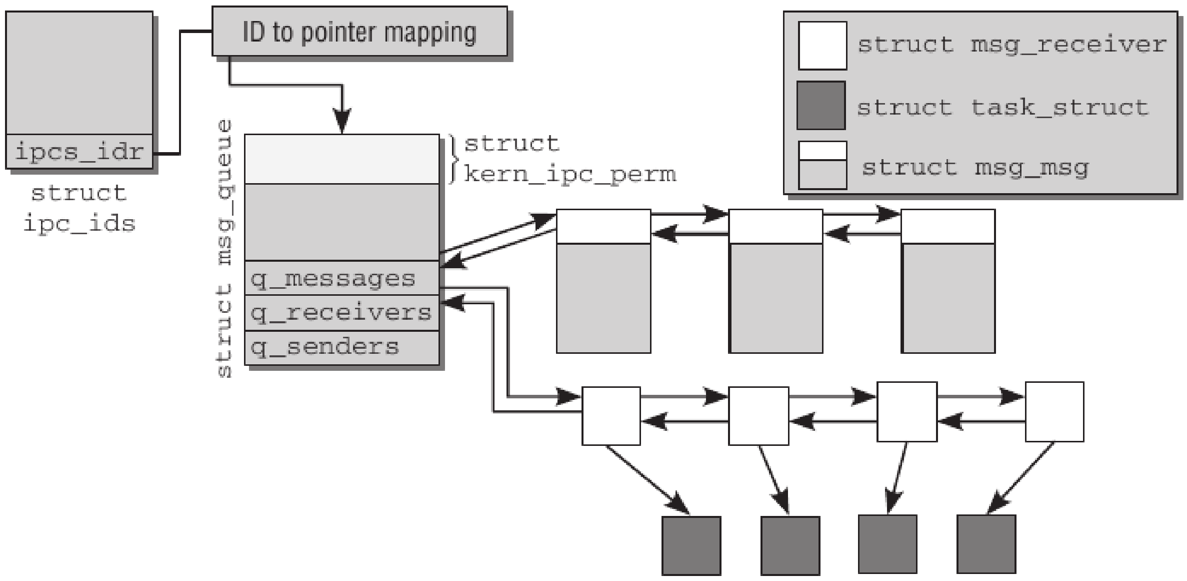
Illustration of system calls involved with SYS V IPCs



----------------------Intentionally left blank-------------------

message queues(SYS V)

Illustration of SYS V message queue architecture



Data-structures involved with message-queues(SYS V)

```
<linux/msg.h>
struct msg_queue {
        struct kern_ipc_perm q_perm;
        time_t q_stime;              /* last msgsnd time */
        time_t q_rtime;              /* last msgrcv time */
        time_t q_ctime;              /* last change time */
        unsigned long q_cbytes;      /* current number of bytes on queue */
        unsigned long q_qnum;        /* number of messages in queue */
        unsigned long q_qbytes;      /* max number of bytes on queue */
        pid_t q_lspid;               /* pid of last msgsnd */
        pid_t q_lrpid;               /* last receive pid */
        struct list_head q_messages;
        struct list_head q_receivers;
        struct list_head q_senders;
};
```

ipc/msg.c

```
struct msg_msg {
        struct list_head m_list;
        long m_type;
        int m_ts;          /* message text size */
        struct msg_msgseg* next;
        /* the actual message follows immediately */
};
```

ipc/msgutils.c

```
struct msg_msgseg {
        struct msg_msgseg* next;
        /* the next part of the message follows immediately */
};
```

ipc/msg.c

```
struct msg_sender {
        struct list_head list;
        struct task_struct* tsk;
};
```

ipc/msg.c

```
struct msg_receiver {
        struct list_head        r_list;
        struct task_struct      *r_tsk;
        int                     r_mode;
        long                    r_msgtype;
        long                    r_maxsize;
        struct msg_msg *volatile r_msg;
};
```

Unix signal ipc mechanism

- signals serve two main purposes:
  - to make a process aware that a specific event has occurred
  - to cause a process to execute a signal handler function included in its code
- the two purposes are not mutually exclusive, because often a process must react to some event by executing a specific routine.

- first 31 signals are regular signals handled by Linux 2.6
- besides the regular signals described in this table, the POSIX standard has introduced a new class of signals denoted as real-time signals ; their signal numbers range from 32 to 64 on Linux.
- they mainly differ from regular signals because they are always queued so that multiple signals sent will be received.
- on the other hand, regular signals of the same kind are not queued: if a regular signal is sent many times in a row, just one of them is delivered to the receiving process.
- although the Linux kernel does not use real-time signals, it fully supports the POSIX standard by means of several specific system calls.

- all signal-related data are managed with the help of a linked data structure consisting of several C structures. Its entry point is the task_struct task structure, which includes various signal-relevant fields.

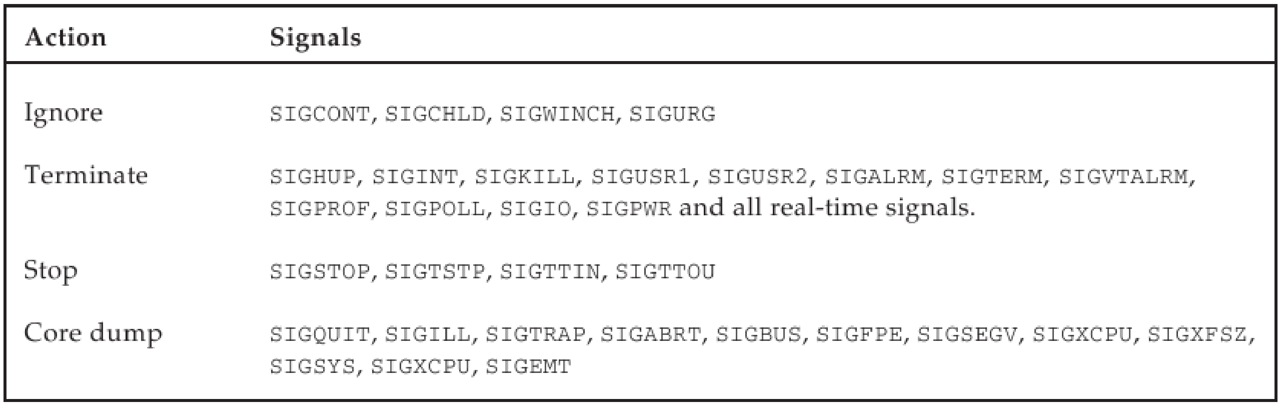
```
<linux/sched.h>
struct task_struct {
...
/* signal handlers */
        struct signal_struct *signal;
        struct sighand_struct *sighand;
        sigset_t blocked;
        struct sigpending pending;
        unsigned long sas_ss_sp;
        size_t sas_ss_size;
...
};
```

```
<linux/sched.h>
struct sighand_struct {
        atomic_t        count;
        struct k_sigaction action[_NSIG];
};
```

```
<asm-arch/signal.h>

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

Default actions for various signals



Action	Signals
Ignore	SIGCONT, SIGCHLD, SIGWINCH, SIGURG
Terminate	SIGHUP, SIGINT, SIGKILL, SIGUSR1, SIGUSR2, SIGALRM, SIGTERM, SIGVTALRM, SIGPROF, SIGPOLL, SIGIO, SIGPWR and all real-time signals.
Stop	SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU
Core dump	SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGBUS, SIGFPE, SIGSEGV, SIGXCPU, SIGXFSZ, SIGSYS, SIGXCPU, SIGEMT

```
<asm-arch/signal.h>
  #define _NSIG              64
  #define _NSIG_BPW           32
  #define _NSIG_WORDS          (_NSIG / _NSIG_BPW)


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

pending is the final task structure element of relevance for signal handling. It creates a linked list of all signals raised and still to be handled by the kernel. The following data structure is used:

```
<linux/signal.h>
  struct sigpending {
          struct list_head list;
          sigset_t signal;
  };
```

list manages all pending signals in a doubly linked list, while signal, with the bitmask described above, specifies the numbers of all signals still to be handled. The list elements are instances of type sigqueue, which is essentially defined as follows:

```
<signal.h>
struct sigqueue {
        struct list_head list;
        siginfo_t info;
};
```

The individual entries are linked by means of list. The siginfo_t data structure contains more detailed information on the pending signals.

```
<asm-generic/siginfo.h>
typedef struct siginfo {
      int si_signo;
      int si_errno;
      int si_code;
      union {
        /* Signal-specific information */
        struct { ... } _kill;
        struct { ... } _timer;   /* POSIX.1b timers */
        struct { ... } _rt;      /* POSIX.1b signals */
        struct { ... } _sigchld;
        struct { ... } _sigfault; /* SIGILL, SIGFPE, SIGSEGV, SIGBUS */
        struct { ... } _sigpoll;
      } _sifields;
} siginfo_t;
```

Illustration of the overall signal management architecture

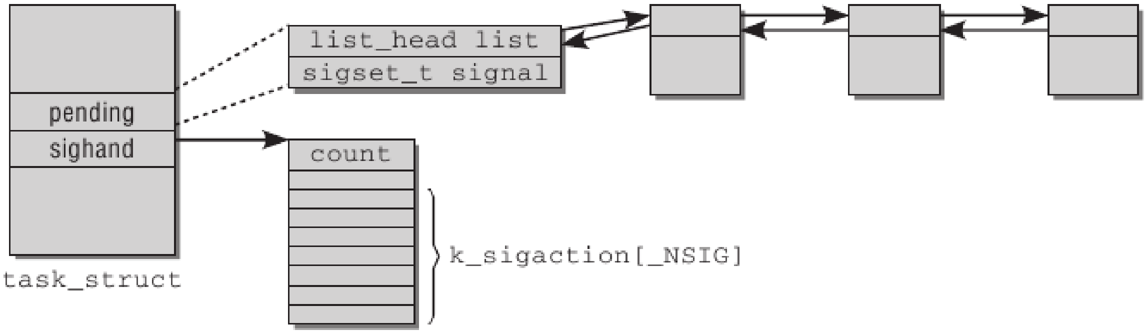
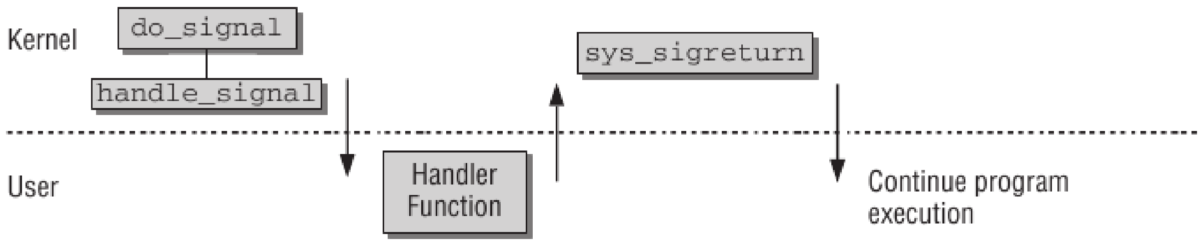
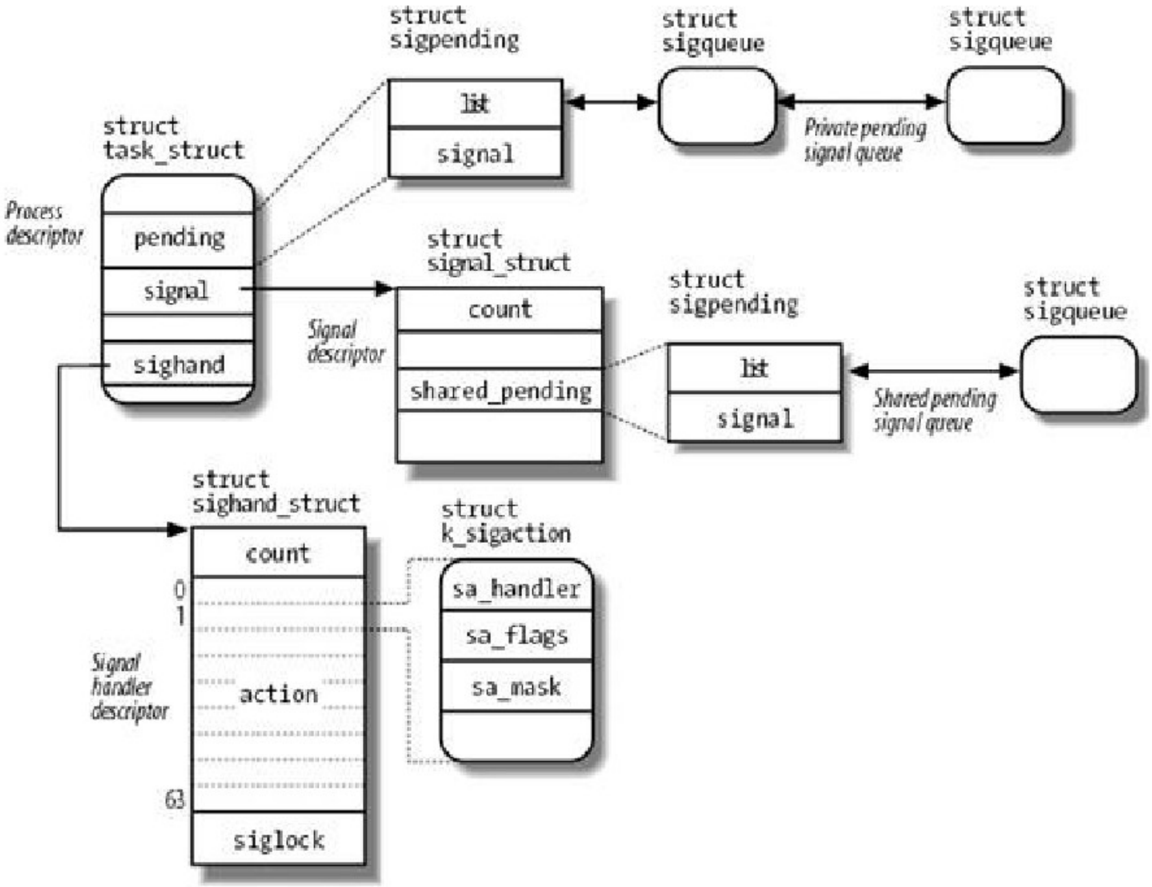


Illustration of signal handling/delivery



signals in a multi-threaded process


struct sigpending
list
signal
struct sigqueue
struct sigqueue
Private pending signal queue
struct task_struct
Process descriptor
pending
signal
sighand
struct signal_struct
Signal descriptor
count
shared_pending
struct sigpending
list
signal
Shared pending signal queue
struct sigqueue
struct sighand_struct
count
0
1
action
63
siglock
Signal handler descriptor
struct k_sigaction
sa_handler
sa_flags
sa_mask