# Network Programming


BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani Campus

# Signals (R1: Ch20)

# What is a Signal?

innovate achieve lead

- A signal is a notification to a process that an event has occurred.
  - A signal is an *asynchronous* event which is delivered to a process.
  - Asynchronous means that the event can occur at any time
    - e.g. user types `ctrl-C`
- Source of signals:
  - Hardware exceptions
    - Dividing by 0, accessing inaccessible memory
  - User typing special characters at the terminal
    - Control-c, Control-\, Control-Z
  - Software events
    - Data available on a descriptor
    - A timer went off
    - Child is terminated etc.
  - kill function allows a process to send a signal to another process.

# Signals

- Each signal is defined by a unique integer, starting from 1 to 31. 0 is a NULL signal.
  - Mapping varies across architectures. Better go by symbols.

| Name | Signal number | Description | SUSv3 | Default |
|---|---|---|---|---|
| SIGABRT | 6 | Abort process | • | core |
| SIGALRM | 14 | Real-time timer expired | • | term |
| SIGBUS | 7 (SAMP=10) | Memory access error | • | core |
| SIGCHLD | 17 (SA=20, MP=18) | Child terminated or stopped | • | ignore |
| SIGCONT | 18 (SA=19, M=25, P=26) | Continue if stopped | • | cont |
| SIGEMT | undef (SAMP=7) | Hardware fault | | term |
| SIGFPE | 8 | Arithmetic exception | • | core |
| SIGHUP | 1 | Hangup | • | term |
| SIGILL | 4 | Illegal instruction | • | core |
| SIGINT | 2 | Terminal interrupt | • | term |
| SIGIO / SIGPOLL | 29 (SA=23, MP=22) | I/O possible | • | term |
| SIGKILL | 9 | Sure kill | • | term |
| SIGPIPE | 13 | Broken pipe | • | term |

SAMP: Sun SPARC and SPARC64 (S), HP/Compaq/Digital Alpha (A), MIPS (M), and HP PA-RISC (P)

# Important Signals



- SIGABRT
  - When a process calls *abort()*, this signal is delivered to the process. Terminates with a core dump.
- SIGALRM
  - Kernel generates this signal upon the expiration of a timer set by alarm() or settimer().
- SIGCHLD
  - Kernel generates this signal upon the termination of a child process.
- SIGHUP
  - When a terminal is disconnected, the controlling process of the terminal is sent this signal.
  - Daemons process repond to this signal by reinitializing from config file.

# Important Signals



- SIGINT
  - Generated when a user presses Ctrl-c on a terminal to interrupt a process.
- SIGIO
  - Useful in signal driven I/O on sockets.
- SIGKILL
  - Can’t be blocked, ignored or caught by a handler. Always terminates a process. Used by admins.
- SIGPIPE
  - Kernel generates this when a pipe has no readers but a process writes into it.
- SIGQUIT
  - Generated when user presses Ctrl-\ on the terminal. It terminates the process with core dump.

# Important Signals



- SIGSEGV
  - Generated when
    - Referencing an unmapped address
    - Updating a read-only page.
    - Accessing kernel memory.
- SIGSTOP
  - Used by admin to stop a process. Can't be ignored, blocked or handled. Process can be started again by SIGCONT signal.
- SIGTERM
  - Standard signal used for terminating a process. *init* process sends this signal during shutdown.
- SIGUSR1 & SIGUSR2
  - Kernel never generates these signals.
  - Available for programmer defined purposes.

# Signals



- A signal is said to be *generated* by some event. Once generated signal is *delivered* to the process. Between the time it is generated and delivered, signal is said to be *pending*.
  - A pending signal is delivered as soon as the process scheduled to run or immediately if the process is running.
- Sometimes we do not want signals to be delivered to the process when executing a critical code segment.
  - Signals can be added to *signal mask* in the kernel. These signals are blocked. If a such is signal is generated, it will be kept pending.
  - When the mask is cleared, the pending signals are delivered to the process.

# Signal Disposition



- A process can inform kernel how it wants to deal with a signal:
  - ignore/discard the signal (not possible with `SIGKILL` or `SIGSTOP`)
    - Kernel will not deliver such a signal to the process.
  - Catch the signal and execute a signal handler function, and then possibly resume execution.
    - Process installs a handler function for a signal with the Kernel. When the signal is received, kernel executes the function on the process behalf in user space.
  - Let the default action apply. Every signal has a default action.
    - Default action can be ignore, terminate the process. Depends on the signal.
- This choice is called the *signal disposition.*

# Setting Signal Disposition



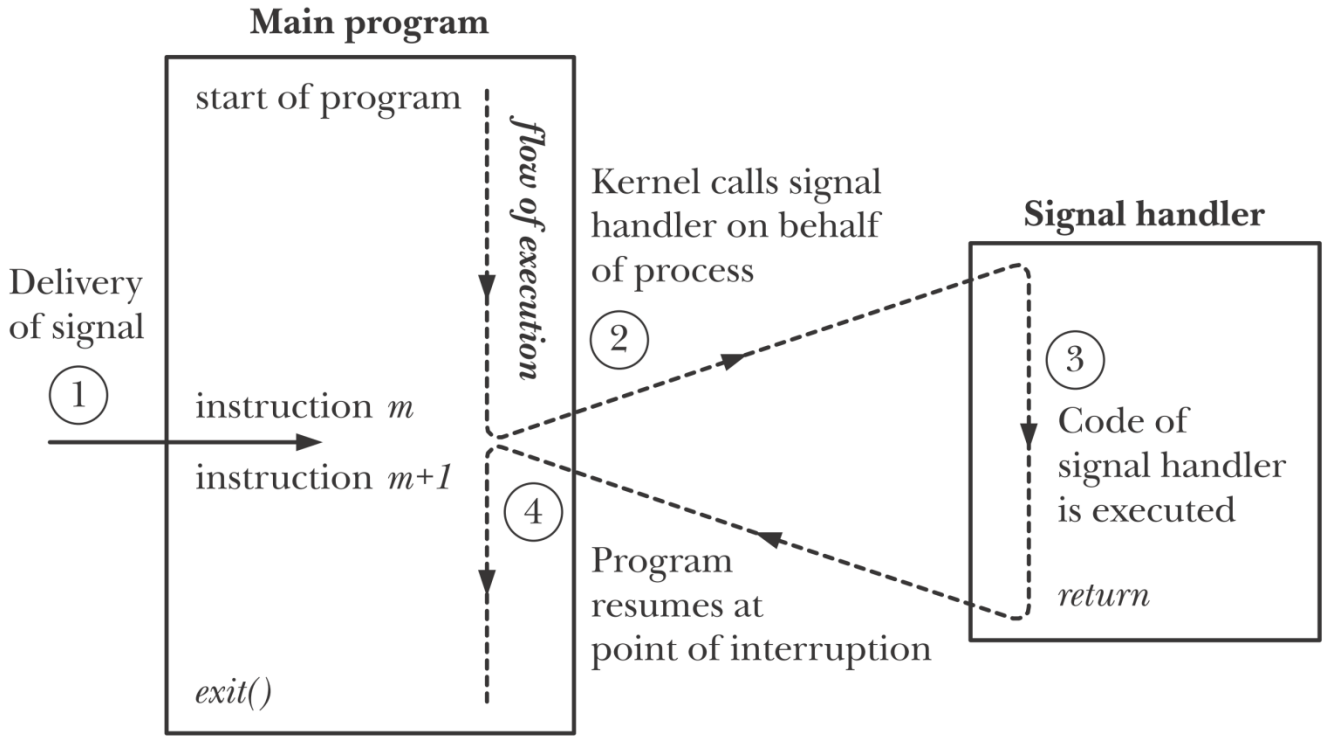
- *signal()* system call takes a function pointer *handler* and registers against the signal *signo*.

```
2  #include <signal.h>
3  typedef void Sigfunc(int);  /* my defn */
4  Sigfunc *signal( int signo, Sigfunc *handler );
5  /*Returns previous signal disposition if ok, SIG_ERR on error.*/
```

  - For other dispositions, the Sigfunc values are
    - SIG_IGN  Ignore / discard the signal.
    - SIG_DFL  Use default action to handle signal.
  - In case of error
    - SIG_ERR  Returned by signal() as an error.
- We can't know the current disposition for a signal with this sys call.
  - *sigaction()* sys call can do that.

# Signal Handlers

- Kernel calls the handlers on the process's behalf.
- Handler may be called at any time.
  - After the execution of the handler, program resumes from the point where it got interrupted.



# Signal Handler: example

innovate achieve lead

```
 2  /*signals/ouch.c*/
 3  #include <signal.h>
 4  #include "tlpi_hdr.h"
 5  static void
 6  sigHandler(int sig)
 7  {
 8      printf("Ouch!\n");                  /* UNSAFE (see Section R1: 21.1.2) */
 9  }
10  int
11  main(int argc, char *argv[])
```

```
1  $ ./ouch
2  0                          Main program loops, displaying successive integers
3  Type Control-C
4  Ouch!                      Signal handler is executed, and returns
5  1                          Control has returned to main program
6  2
7  Type Control-C again
8  Ouch!
9  3
10 Type Control-\ (the terminal quit character)
11 Quit (core dumped)
```

# kill() and raise()function



- Send a signal to a process (or group of processes).

```
2  #include <signal.h>
3  int kill( pid_t pid, int signo );
4  int raise(int signo);
5  /*Return 0 if successful, -1 on error.*/
```

  - pid > 0  send signal to process pid
  - pid== 0  send signal to all processes whose process group ID equals the sender's pgid.
    - e.g. parent kills all children
- Using raise(), a process can send a signal to itself.
- To know whether a PID is in use
  - Send a null signal to that PID
    - kill(PID, 0)
- kill command
  - kill -INT 9400

# Signal Sets



- Many signal related system calls need a set of signals as input.
- Signal set is a data structure represented by *sigset_t* data type.
- Following are the library functions on *sigset_t* data type.

```
2  #include <signal.h>
3  int sigemptyset(sigset_t *set);
4  int sigfillset(sigset_t *set);
5  int sigaddset(sigset_t *set, int signo);
6  int sigdelset(sigset_t *set, int signo);
7  /*All four return: 0 on success, -1 on error */
8  int sigismember(const sigset_t *set, int signo);
9  /*Returns: 1 if true, 0 if false, -1 on error*/
```

# Signal Mask



- For each process kernel maintains a *signal mask* – set of signals the process has blocked.
  - When a signal is to be delivered but it is blocked, then that signal is kept pending until it is unblocked by the process.
  - When a signal handler is invoked, the signal that caused invocation is automatically added to the mask.
- Using *sigprocmask()* system call, signals can be added or deleted from the mask or retrieve the mask.

```
2  #include <signal.h>
3  int sigprocmask(int  how , const sigset_t * set , sigset_t * oldset );
4  /*Returns 0 on success, or -1 on error*/
```

  - how is interpreted as
    - SIG_BLOCK: add *set* to the signal mask
    - SIG_UNBLOCK: delete *set* from the mask.
    - SIG_SETMASK: reset signal mask to *set.*

# sigprocmask()

innovate achieve lead

```
2   sigset_t blockSet, prevMask;
3   /* Initialize a signal set to contain SIGINT */
4   sigemptyset(&blockSet);
5   sigaddset(&blockSet, SIGINT);
6   /* Block SIGINT, save previous signal mask */
7   if (sigprocmask(SIG_BLOCK, &blockSet, &prevMask) == -1)
8       errExit("sigprocmask1");
9   /* ... Code that should not be interrupted by SIGINT ... */
10  /* Restore previous signal mask, unblocking SIGINT */
11  if (sigprocmask(SIG_SETMASK, &prevMask, NULL) == -1)
12      errExit("sigprocmask2");
```

# Pending Signals



- To know pending signals we use *sigpending()* system call.

```
2  #include <signal.h>
3  int sigpending(sigset_t * set );
4  /*Returns 0 on success, or -1 on error*/
```

  - Pending signals are returned through *set.*
  - They are examined using *sigismember()* function.
- If a blocked signal is generated more than once then in most systems the signal is delivered only once. That is the signal is not queued.
- If many signals of *different* types are ready to be delivered (e.g. a `SIGINT`, `SIGSEGV`, `SIGUSR1`), they are not delivered in any fixed order.

# Setting Signal Disposition: *sigaction()*

innovate achieve lead

```
2  #include <signal.h>
3  int sigaction(int  sig , const struct sigaction * act ,
4  struct sigaction * oldact );
5  /*Returns 0 on success, or -1 on error*/
6
7  struct sigaction {
8      void   (*sa_handler)(int);/* Address of handler */
9      sigset_t sa_mask;         /* Signals blocked during handler
10                                   invocation */
11     int      sa_flags;        /* Flags controlling handler invocation */
12     void   (*sa_restorer)(void);/* Not for application use */
13 };
```

- An alternative to *signal()*
  - Allows us to retrieve the disposition of a signal without changing it.
  - To atomically block signals while executing in a handler.
  - To retrieve attribute of a signal such pid, uid etc using SA_SIGINFO.
  - To automatically restart a system call using SA_RESTART.

# Waiting for a Signal



- Calling *pause()* suspends execution of the process until the call is interrupted by a signal handler.

```
2  #include <unistd.h>
3  int pause(void);
4  /*Always returns -1 with errno set to  EINTR*/
```

  - When a signal is handled, *pause()* is interrupted and always returns -1.

- *sigsuspend():* atomically unblocks signals and suspends execution of the process until a signal is caught and its handler returns.

```
2  #include <signal.h>
3  int sigsuspend(const sigset_t * mask );
4  /*(Normally) returns -1 with errno set to  EINTR*/
```

  - Replaces the signal mask in the kernel with *mask.*
  - Suspends execution until signal handler returns.

# *pause()* example

innovate achieve lead

```
 2  #include <signal.h>
 3  long n;
 4  void sigalrm(int signo){
 5          alarm(1);
 6          n=n+1;
 7          printf("%d seconds elapsed\n",n);
 8          }
 9  main(){
10          signal(SIGALRM, sigalrm);
11          alarm(1);
12
13          while(1) pause();
14          }
```

# Synchronously Waiting for a Signal



- *pause()* and *sigsusepnd()* wait until a signal handler is executed. But we can do away with signal handlers.

```
2  #define _POSIX_C_SOURCE 199309
3  #include <signal.h>
4  int sigwaitinfo(const sigset_t * set , siginfo_t * info );
5  /*Returns number of delivered signal on success, or -1 on error*/
```

  - Suspends execution until one of the signals in *set* becomes pending and returns that signal number.
  - Useful only if signal is *set* are blocked using *sigprocmask().*
  - Faster than *sigsuspend()* because there is no signal handler.

# *sigwaitinfo()* example

innovate achieve lead

```
 /*signals/t_sigwaitinfo.c*/
/* Block all signals (except SIGKILL and SIGSTOP) */
    sigfillset(&allSigs);
    if (sigprocmask(SIG_SETMASK, &allSigs, NULL) == -1)
        errExit("sigprocmask");
    printf("%s: signals blocked\n", argv[0]);

     for (;;) {/* Fetch signals until SIGINT (^C) or SIGTERM */
        sig = sigwaitinfo(&allSigs, &si);
        if (sig == -1)
            errExit("sigwaitinfo");
        if (sig == SIGINT || sig == SIGTERM)
            exit(EXIT_SUCCESS);
     }
```

# Non-local goto in Signal Handler



- POSIX does not specify whether *longjmp()* will restore the signal context. If you want to save and restore signal masks, use *siglongjmp()*.
  - In a signal handler the signal that caused signal handler invocation is added to the signal mask. It will not be removed, if *longjmp()* is called.
- POSIX does not specify whether *setjmp()* will save the signal context. If you want to save signal masks, use *sigsetjmp()*.

```
2  #include <setjmp.h>
3  int sigsetjmp(sigjmp_buf env, int savemask);
4  /*Returns: 0 if called directly, nonzero if returning from a call to siglongjmp*/
5  void siglongjmp(sigjmp_buf env, int val);
```

innovate achieve lead

BITS Pilani

Pilani Campus

# Realtime Signals (R1: Ch22.8)

# Realtime Signals Advantages



- Real time signals provide an increased range of signals that can be used for application-defined purposes.
- Realtime signals are queued.
- When sending a realtime signal, it is possible to specify data
- The order of delivery of different realtime signals is guaranteed.
  - If multiple different realtime signals are pending, then the lowest-numbered signal is delivered first.
- Realtime signals are not individually identified by different constants in the manner of standard signals.
  - a realtime signal number can be referred to by adding a value to SIGRTMIN ;
    - the expression (SIGRTMIN + 1) refers to the second realtime signal

# Send realtime signal

innovate achieve lead

- The sending process sends the signal plus its accompanying data using the sigqueue() system call.

```
#define _POSIX_C_SOURCE 199309
#include <signal.h>
int sigqueue(pid_t  pid , int  sig , const union sigval  value );
    /*Returns 0 on success, or -1 on error*/
```

```
6   union sigval {
7       int   sival_int;      /* Integer value for accompanying data */
8       void *sival_ptr;      /* Pointer value for accompanying data */
9   };
```

# Receiving realtime signal



- The receiving process establishes a handler for the signal using a call to sigaction() that specifies the SA_SIGINFO flag.

```
1   struct sigaction act;
2   sigemptyset(&act.sa_mask);
3   act.sa_sigaction = handler;
4   act.sa_flags = SA_RESTART | SA_SIGINFO;
5   if (sigaction(SIGRTMIN + 5, &act, NULL) == -1)
6       errExit("sigaction");
```

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

# siginfo structure

innovate achieve lead

```
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;
```

- For real time signals, si_signo, si_code (source), si_value, si_pid, si_uid are set.

innovate achieve lead

BITS Pilani

Pilani Campus

# Thank You