CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Signals

Today's Topics

- Signals
- Process Interaction with Signals

Signals

- Unix supports a signal facility which looks like a software version of the interrupt subsystem on a conventional CPU
- Process can *send a signal* to another
- Kernel can send signal to a process (like an interrupt or a trap)
- Process can arrange to *ignore or handle* a given signal
- Processes handle signals by binding a function to the arrival of the designated signal (like embedding an interrupt handler in the interrupt vector)
- Section 1 stuff: `kill(1)`
- Section 2&3 stuff: `kill(2) and signal(3)`
- Section 7: `signal(7)`

Signals Defined in Linux

- Different signal types in Linux, coded by small integers
- Analogous to different interrupt sources in hardware
- The section 7 signal page has details about available signals on the system

Signal	Value	Action
SIGHUP	1	Term
SIGINT	2	Term
SIGQUIT	3	Core
SIGILL	4	Core
SIGABRT	6	Core
SIGFPE	8	Core
SIGKILL	9	Term
SIGSEGV	11	Core
SIGPIPE	13	Term
SIGALRM	14	Term
SIGTERM	15	Term
SIGUSR1	30,10,16	Term
SIGUSR2	31,12,17	Term
SIGCHLD	20,17,18	Ign
SIGCONT	19,18,25	Cont
SIGSTOP	17,19,23	Stop

Default Actions

- Term - Terminate the process
- Core - Terminate the process and create a core dump
- Ign - Ignore the Signal
- Cont - Continue the process if it is stopped
- Stop - Stop the process

Sending a Signal: The kill() System Call

- Section 2 manual page for kill!
- Different from what the name implies: used to send any signal, not just SIGTERM
- If pid is positive, then signal 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 except for the first one
- If pid is less than -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 performed anyway

Handling Signals: The signal() System Call

- Section 3 manual page for signal!
- The signal system call installs a new signal handler for a signal
- Alternatively, the default action could be chosen for a signal, or it could be set to ignore

Code Examples

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

main() {
  int kidpid;

  if ((kidpid=fork()) == 0) {
    execl("catchsig", "catchsig", (char *)0);
  }
  else {
    sleep(5);
    kill(kidpid,SIGUSR1);
    wait(NULL);
    fprintf(stderr, "Sendsig detects death of catchsig\n");
    exit(0);
  }
  exit(1);
}
```

Code Examples

```c
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

void usr1handler() {
  write(1, "\nOuttahere\n", 11);
  exit(0);
}

main() {
  /* Embed the handler */
  signal(SIGUSR1, usr1handler);
  while(1) {
    sleep(1);
    write(1,"A",1);
  }
}
```

You can also send signals from the command line!

Code Examples

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

main() {
  int kidpid, status;

  if ((kidpid=fork()) == 0) {
    execl("newcatchsig", "newcatchsig", (char *)0);
  }
  else {
    sleep(5);
    kill(kidpid,SIGUSR1);
    sleep(5);
    kill(kidpid,SIGUSR1);
    wait(&status);
    fprintf(stderr, "Sendsig detects death of catchsig\n");
    if (WIFSIGNALED(status))
      fprintf(stderr, "Died due to uncaught sig %d\n"
        ,WTERMSIG(status));
    exit(0);
  }
  exit(1);
}
```

Code Examples

```c
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

int flag = 1;

void usr1handler() {
  flag = 0;
}

main() {
  signal(SIGUSR1, usr1handler);
  while(flag) { sleep(1); write(1,"A",1);}
  write(1, "\nEscaped Loop\n", 14);
  flag = 1;
  while(flag) { sleep(1); write(1,"B",1);}
  write(1, "\nEscaped Loop\n", 14);
  exit(0);
}
```

Compatibility Issues

On some older systems you may get the following error:

```
AAAAA
Escaped Loop
BBBBB
Sendsig detects death of catchsig
Died due to uncaught sig 10
```

See the Portability section of the manual!

Code Examples

```
#include <signal.h>
#include <stdio.h>
int x;
void handler(int sig) {
  x++;
}
main() {
  int cpid;
  x = 0;
  signal(SIGUSR1,handler);
  cpid=fork();
  if (cpid==0) {
    while (x == 0);
    write(1, "Child second.\n",14);
  } else {
    write(1, "Parent first.\n",14);
    /* x++; Not here!!! */
    kill(cpid,SIGUSR1);
    wait();
    write(1,"Parent third.\n",14);
  }
}
```

Signal Handlers and Reentrant Functions

- Signal handler may be called from within itself
- This can lead to inconsistent results
- Consider:

Broken Example

- Let's have an array to store signal numbers we receive in the order we get them
- Let our handler search for an empty slot to place the new signal in, then write it in that slot
- Let our program register handlers and simply wait.

Not Reentrant Example

```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
int list[10] = {0,0,0,0,0,0,0,0,0,0};
void handler(int sig) {
   int i = 0;
   while (list[i] != 0) {i++;}
   list[i] = sig;
   write(1,"Outta here\n",11);
}
void dump(int sig) {
   int i;
   for (i=0;i<10;i++) {
      printf("list[%d]=%d\n",i,list[i]);
   }
   _exit(0);
}
main() {
   signal(SIGUSR1,handler);
   signal(SIGUSR2,handler);
   signal(SIGTERM,dump);
   printf("Handlers installed\n");
   while(1);
}
```

Visible Not Reentrant Example

```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
int list[10] = {0,0,0,0,0,0,0,0,0,0};
void handler(int sig) {
  int i = 0;
  while (list[i] != 0) {i++;}
  sleep(10);
  list[i] = sig;
  write(1,"Outta here\n",11);
}
void dump(int sig) {
  int i;
  for (i=0;i<10;i++) {
    printf("list[%d]=%d\n",i,list[i]);
  }
  _exit(0);
}
main() {
  signal(SIGUSR1,handler);
  signal(SIGUSR2,handler);
  signal(SIGTERM,dump);
  printf("Handlers installed\n");
  while(1);
}
```

Reentrant Functions

- A reentrant function can begin responding to one call, be interrupted by other calls and complete them all with the same results as if the function had received and executed each call serially.
- POSIX.1 standard specifies functions that are guaranteed to be reentrant
- Most notably, malloc() and friends are NOT reentrant
- The list is in manual section 7 signal