CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Signals

Today's Topics

- Advanced Signals
- Process Interaction with Signals

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

fork() and exec() interaction with Signals

- Signal handling options are catch, default and ignore
- On `fork()`, child inherits parent's disposition
- Since the address of the handler is meaningful in the child, the catch dispositions are inherited as well
- On `exec()`, caught signals are restored to default
- Ignore dispositions carry over as well

pause()

pause() makes a process sleep until a signal is recieved and the handler returns

Let's design an example:

- Fork into a child:
  - set SIGINT to ignore
  - write pid to console
  - exec sleeper
    - install SIGTERM handler
    - pause()
- In the parent, wait for child
- exit

Example

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

main (int argc, char **argv) {
   int cpid;
   int status;

   cpid = fork();
   if (cpid == 0) {
      signal(SIGINT, SIG_IGN);
      printf("Child pid <%d>\n",getpid());
      fflush(stdout);
      execl("sleeper", "sleeper", (char *)0);
      _exit(1);
   } else {
      wait(&status);
   }
}
```

Example

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

void handler() {
   printf("Sleeper exits on SIGTERM\n");
   fflush(stdout);
}

main() {
   signal(SIGTERM, handler);
   pause();
}
```

Blocking Signals

- Process has option to block a signal
- When a blocked signal is generated
  - If action is default or to catch, signal remains pending until
    - Signal is unblocked
    - Associated action set to ignore
  - Most systems deliver blocked signal only once
- Each process has signal mask for blocked signals

Functions for Handling Signal Actions

- `sigaction()` - This system call is used to change the action taken by a process on receipt of a specific signal
- `sigprocmask()` - This system call is used to change the list of currently blocked signals
- `sigpending()` - This system call allows the examination of pending signals (ones which have been raised while blocked).
- `sigsuspend()` - This system call temporarily replaces the signal mask of the calling process and then suspends the process until delivery of a signal whose action is to invoke a signal handler or to terminate a process.
- Manual pages for details!

Setting up our Example

- Our handler function prints number for received signal then exits
- The main function:
  - Registering handler for SIGINT and SIGTERM
  - Block SIGINT, then sleep for a while
  - Check for pending INT signals
  - Block SIGTERM as well, sleep for a while
  - Unblock both signals, then go back to sleep

Example I

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

void handler(int receivedSig) {
   printf("Handler caught sig <%d>\n",receivedSig);
   fflush(stdout);
   _exit(1);
}

main (int argc, char **argv) {
   sigset_t termMask;
   sigset_t intMask;
   sigset_t oldMask;
   sigset_t pendingMask;

   signal(SIGTERM, handler);
   signal(SIGINT, handler);

   sigemptyset(&termMask);
   sigaddset(&termMask, SIGTERM);
   sigemptyset(&intMask);
```

Example II

```
    sigaddset(&intMask, SIGINT);

    sigprocmask(SIG_BLOCK, &intMask, &oldMask);
    printf("SIGINT Blocked. Process <%d> sleeping.\n", getpid());
    fflush(stdout);
    sleep(30);

    sigpending(&pendingMask);
    if (sigismember(&pendingMask, SIGINT)) {
       printf("SIGINT pending. Ready to block TERM.\n");
       fflush(stdout);
    }

    sigprocmask(SIG_BLOCK, &termMask, NULL);
    printf("SIGTERM blocked. Sleeping.\n"); fflush(stdout);
    sleep(30);

    printf("Unblocking.\n"); fflush(stdout);

    sigprocmask(SIG_UNBLOCK, &termMask, NULL);
    sigprocmask(SIG_SETMASK, &oldMask, NULL);
    sleep(30);
}
```

Examples

- TERM signal before it's blocked
- INT signal when INT is blocked but before TERM is blocked
- TERM signal after TERM is blocked
- TERM and rapidly INT after both of them are blocked
- TERM and rapidly INT after both of them are blocked after the exit statement is removed from the code