CS240: Programming in C

Lecture 16: Signals

Synchronous vs. Asynchronous

- Synchronous action:
  - A procedure call: the caller calls the callee and waits until the callee returns with a result
  - A phone call: both parties must be available for the call to go through.
- An asynchronous action:
  - A response from the environment: user types Ctrl-C while the program runs
  - Email or text messages can be sent without the other party ready to receive

Dealing with an Asynchronous Action

1. Poll: ask the kernel or the OS: did event E take place since the last time I checked?

2. Handle: inform the kernel that when event E takes place, do the following ...

Kinds of Signals

Interrupts. Environment-triggered (SIGINT, Ctrl-C)

Hardware. (SIGSEGV); divide by 0, invalid memory reference

Software. (SIGPIPE, SIGALRM). Timeout on network connection, a broken pipe, ...

Handling a Signal

- Ignore it.
  - Doesn't always work though (e.g., not a good idea to ignore a hardware exception...)
- *Catch the signal*
  - setup a signal handler that gets invoked whenever the signal occurs.
- All signals have a default action
  - Most of the time, the default is to kill the process.

Generating a signal

- Most common case: kill

```
#include <signal.h>
int kill(pid_t pid, int sig);
 /* send signal 'sig' to process 'pid' */
```

- raise(signal): generate a signal that is handled by the program that contains the call to raise.

signal.h

- Defines a number of common signals

SIGABRT
Abnormal termination, such as instigated by the abort function. (Abort.)
SIGFPE
Erroneous arithmetic operation, such as divide by 0 or overflow. (Floating point exception.)
SIGILL
An ‘invalid object program’ has been detected. This usually means that there is an illegal instruction in the program. (Illegal instruction.)
SIGINT
Interactive attention signal; on interactive systems this is usually generated by typing some ‘break-in’ key at the terminal. (Interrupt.)
SIGSEGV
Invalid storage access; most frequently caused by attempting to store some value in an object pointed to by a bad pointer. (Segment violation.)
SIGTERM
Termination request made to the program. (Terminate.)

Example

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

long  prev_fact, i;

void  SIGhandler(int);

void  SIGhandler(int sig) {
  printf("\nReceived a SIGUSR1.  The answer is %ld! = %ld\n",
                i-1, prev_fact);
  exit(0);
}

void  main(void)
{
     long  fact;

     printf("Factorial Computation:\n\n");
     signal(SIGUSR1, SIGhandler);
     for (prev_fact = i = 1; ; i++, prev_fact = fact) {
          fact = prev_fact * i;
          if (fact < 0)
               raise(SIGUSR1);
          else if (i % 3 == 0)
               printf("     %ld! = %ld\n", i, fact);
     }
}
```

Defining a signal handler

```
#include <signal.h>
void (*signal (int sig, void (*func)(int)))(int);
```

signal is a function pointer to a function that takes as arguments:

- a signal (represented as an int)
- a handler

and returns a function that takes an int and returns void

The handler is a function pointer to a function that takes an int and returns void.

Signal handler

- signal installs a new handler for the supplied signal
- It returns the previous value of the handler as its result
  - If no such value exists, it returns SIG_ERR and sets errno appropriately

Example

```
static void sig_usr(int); /* one handler for two signals */
int main (void) {
  if (signal(SIGUSR1, sig_usr) == SIG_ERR)
      perror("cannot catch signal SIGUSR1");
  if (signal(SIGUSR2, sig_usr) == SIG_ERR)
      perror("cannot catch signal SIGUSR2");
  for(;;) pause();
}

static void sig_usr(int signo) {
   /*argument is signal number*/
  if (signo == SIGUSR1)
    printf("received SIGUSR1\n");
  else if (signo == SIGUSR2)
    printf("received SIGUSR2\n");
  else error("received signal %d\n", signo);
return;
}
```

Example

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


FILE *temp_file;
void leave(int sig);

main() {
        (void) signal(SIGINT,leave);
        temp_file = fopen("tmp","w");
        for(;;) {
                /*
                 * Do things....
                 */
                printf("Ready...\n");
                (void)getchar();
        }
        /* can't get here ... */
        exit(EXIT_SUCCESS);
}
```

```
/*
 * on receipt of SIGINT,
 * close tmp file */

void leave(int sig) {
  fprintf (temp_file,"\nInterrupted
                          \n");
  fclose(temp_file);
  exit(sig);
}
```

Example using setjmp/longjmp

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>


int i, j;
long T0;
jmp_buf Env;

void alarm_handler(int dummy)
{
  long t1;

  t1 = time(0) - T0;
  printf("%ld second%s has passed:
             j = %d.  i = %d\n", t1,
     (t1 == 1) ? "" : "s", j, i);
  if (t1 >= 8) {
    printf("Giving up\n");
    longjmp(Env, 1);
  }
  alarm(1);
  signal(SIGALRM, alarm_handler);
}
```

```
int main()
{

  signal(SIGALRM, alarm_handler);
  alarm(1);

  if (setjmp(Env) != 0) {
    printf("Gave up:  j = %d, i = %d\n", j, i);
    exit(1);
  }

  T0 = time(0);

  for (j = 0; j < 10000; j++) {
    for (i = 0; i < 1000000; i++);
  }
}
```