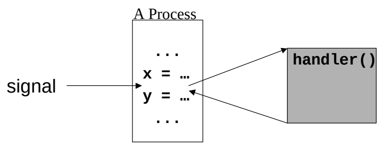
Signals

CS 217

Signals

- Outside world interrupts (signals) the process
  - user types ^C, ^Z, …
  - phone or terminal hangs up
  - illegal instruction, bus error, …
- Process responds to the signal



Signals (cont)

- Register a signal handler (system call)

```
void (* signal(int sig,
        void (* handler)(int)) (int);
```

  function **`handler`** will be invoked on signal **`sig`**

  return the old handler on success; **`-1`** on error

- Example

```
#include <signal.h>
...
signal(SIGINT, SIG_IGN);
signal(SIGSTOP, SIG_DFL);
signal(SIGALRM, handler);
```

Signals (cont)

- Predefined signals

SIGKILL	**terminate (can’t catch)**
SIGILL	**illegal instruction**
SIGTERM	**software termination**
SIGSTOP	**suspend (^Z)**
SIGALRM	**alarm clock**
SIGINT	**interrupt (^C)**

- Send a signal (command and system call)

```
kill –9 %2
int kill(pid_t pid, int sig);
```

Example

```
#include <signal.h>
char *tmpfile = “temp.xxx”;
void cleanup(int sig); {
   unlink(tmpfile);
   exit(1);
}
void main(void) {
   int fd;

   signal(SIGINT, cleanup);
   fd = open(tmpfile, O_CREAT, 0666);
      ...
   close(fd);
}
```

Example (cont)

- Same handler as previous slide, but…

```
void main(void) {
   int fd;

   if (signal(SIGINT, SIG_IGN) != SIG_IGN)
      signal(SIGINT, cleanup);
   fd = open(tmpfile, O_CREAT, 0666);
      ...
   close(fd);
}
```

PL Support

- Some programming languages (not C) provide explicit support for exception handling
- Throw an exception

```
throw("out of memory");
```

- Try and catch

```
try {
   ... nested calls that can raise exceptions ...
}
catch("out of memory") {
... exception handler ...
}
```

Signals with Fork

- Signals are sent to all your processes
- Parent may want to ignore signals but let child process them

```
#include <signal.h>
...
if (fork() == 0)
   execlp(...)
h = signal(SIGINT, SIG_IGN);
wait(&status);
signal(SIGINT, h);
```

Blocking Signals

- Keep signals from being delivered (sys call)

```
int sigblock(int mask);
int sigmask(int sig);
```

these are BSD-style; alternative POSIX calls

- Used to protect critical sections of code

```
mask = sigmask(SIGTERM);
oldmask = sigblock(mask);
...critical section...
sigblock(oldmask);
```

Setjmp/Longjmp

- Interrupt a long printout and go back to the main processing loop

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

void handler(int sig) {
   signal(SIGINT, handler);
   fprintf(stderr, "Interrupted\n");
   longjmp(jmpbuf, 0);
}
```

Setjmp/Longjmp (cont)

```
void main(void)
{
   if (signal(SIGINT, SIG_IGN) != SIG_IGN)
      signal(SIGINT, handler);
   setjmp(jmpbuf);
   for ( ; ; ) {
      ...
   }
}
```

Alarms

- Create a child process and kill it in 20 secs

```
#include <signal.h>
int pid;

void OnAlarm(int sig) {
   kill(pid, SIGKILL);
}
...
if ((pid = fork()) == 0)
   execlp(...)
signal(SIGALRM, OnAlarm);
alarm(20);
...
```

Alarms (cont)

- Used to implement timeouts
  - TCP: retransmit if don't receive an ACK
  - NFS: decide that server is not responding
- What if you want smaller granularity?

```
    unsigned ualarm(unsigned value,
                    unsigned interval);
    int setitimer(int which,
                  struct itimerval *value,
                  struct itimerval *ovalue);
  which = ITIMER_REAL, ITIMER_VIRTUAL,
          ITIMER_PROF
```

Fall 2001 13

Reading the Clock

- System call

```
gettimeofday(struct timeval *tv,
             struct timezone *tz);
```

- Example

```
#include <sys/time.h>
...
gettimeofday(start, NULL);
... activity you want to measure ...
gettimeofday(stop, NULL);
print_elapsed_time(start, stop)
```

Fall 2001 14

Precision

- Time structures

```
struct timeval {
   long tv_sec;
   long tv_usec;
}
```

structure `itimerval` is similar

- Limitations
  - system clock (~10ms)
  - scheduling delays
- Solution: read/write the CPU cycle counter

Fall 2001 15