Signal

Signal Concepts

- Signals
  - ✓ are a sort of software interrupts for application processes
  - ✓ have names for each
  - ✓ provide a way of handling asynchronous events

- Conditions that generate a signal
  - ✓ Signals from terminal
  - ✓ Signals from hardware exceptions
  - ✓ Signals from software conditions
  - ✓ Signals from processes using *kill()* function
  - ✓ Signals from shells using *kill* command

Signal Concepts

- Actions associated with a signal
  - ✓ Catch the signal
    - ➢ Register a signal handler
  - ✓ Let the default action apply if no signal handler is registered
    - ➢ Default action for most signals: termination of the process
  - ✓ When a process calls fork(), the child inherits the parent's signal actions

- Major signals

✓ SIGALRM	SIGIO	SIGUSR1
✓ SIGCHLD	SIGKILL	SIGUSR2
✓ SIGCONT	SIGQUIT	
✓ SIGHUP	SIGSTOP	
✓ SIGINT	SIGTERM	

Register a signal handler

- Signal
  - ✓ #include <signal.h>
  - ✓ void (*signal(int signo, void (*func)(int))) (int);
  - ✓ return: previous signal handler if OK, SIG_ERR (-1) on error

```
#include <stdio.h>
#include <signal.h>
void SigIntHandler(int signo) {
    printf("Received a SIGINT signal\n");
    /* process the signal */
    exit(0);
}
void main() {
    signal(SIGINT, SigIntHandler);
    /* do something */
}
```

Exercise

- Simple example for signal

  $ arm-linux-gnueabihf-gcc -o sig1 sig1.c (or make sig1)
  $ ./sig1
  ^C

- Another example

  $ arm-linux-gnueabihf-gcc -o sig2 sig2.c (or make sig2)
  $ ./sig2&

  $ pid=`pgrep sig2`
  $ kill –USR1 $pid
  $ kill –USR2 $pid
  $ kill –TERM $pid

Send a signal

- Send a signal to a process or a group of processes
  - ✓ #include <sys/types.h>
  - ✓ #include <signal.h>
  - ✓ int kill(pid_t pid, int signo);
  - ✓ return: 0 if OK, -1 on error

  - ✓ The first argument, pid
    - ➢ If pid > 0, send to the process of which the process ID is pid
    - ➢ If pid == 0, send to all processes of which the group ID equals the sender's
    - ➢ If pid < 0, send to all processes of which the group ID is the absolute value of pid

- Send a signal to itself
  - ✓ int raise(int signo);
  - ✓ return:0 if OK, -1 on error

Other system calls related with signals

- Sleep for the specified number of seconds
  - ✓ #include <unist.h>
  - ✓ unsigned int sleep(unsigned int seconds);
  - ✓ return:0 or number of seconds until previously set alarm
    (i.e., unslept time)

- Set an alarm clock for delivery of a signal
  - ✓ #include <unistd.h>
  - ✓ unsigned int alarm(unsigned int seconds);
  - ✓ return: 0 or number of seconds until previously set alarm
    (i.e., unslept time)

- Wait for signal
  - ✓ #include <unistd.h>
  - ✓ int pause(void);
  - ✓ return: -1 with errno set to EINTR

Exercise

- Make my own sleep system call using signal & pause
  - $ arm-linux-gnueabihf-gcc –o mysleep mysleep.c (or make mysleep)
  - $ ./mysleep

- An example for periodic alarms
  - $ arm-linux-gnueabihf-gcc –o alarm alarm.c (or make alarm)
  - $ ./alarm

Interrupted system calls

- If a process catch a signal while it has been blocked in a system call,
  - ✓ The system call returns an error and errno is set to EINTR
  - ✓ It's a good chance to wake up the blocked system call
  - ✓ So, you have to handle it as follows:

```
……
while (1) {
   if ( (n = read(fd, buf, MAX_BUF)) < 0 ) {
      if (errno == EINTR)
         continue;
       /* handle other errors */
      }
      …….
}
```

Signal handling in threads

- Which thread should handle signals?
  - ✓ To the thread to which the signal applies
  - ✓ To every thread in the process
  - ✓ To certain threads in the process
  - ✓ Assign a specific thread to receive all signals for the process
    - ➢ Solaris' implementation: main thread receives signals

Exercise

- Signals in threads
  $ arm-linux-gnueabihf-gcc -o sig_thread sig_thread.c –lpthread (or make sig_thread)
  $ ./sig_thread

Kill another thread (Thread cancellation)

- Send a cancellation request
  - ✓ #include <pthread.h>
  - ✓ int pthread_cancel(pthread_t tid);
  - ✓ return: 0 if OK, non-zero on error

- Set the type and state of cancellation request
  - ✓ #include <pthread.h>
  - ✓ int pthread_setcancelstate(int state, int *oldstate);
  - ✓ int pthread_setcanceltype(int type, int *oldtype);
  - ✓ return: 0 if OK, non-zero on error
  - ✓ The first argument in *pthread_setcancelstate()*, state
    - ➢ PTHREAD_CANCEL_DISALE, PTHREAD_CANCEL_ENABLE(by default)
  - ✓ The first argument in *pthread_setcanceltype()*, type
    - ➢ PTHREAD_CANCEL_ASYNCHRONOUS, PTHREAD_CANCEL_DIFERRED(by default)

Exercise

- Cancel thread executions
  $ arm-linux-gnueabihf-gcc -o cancel cancel.c –lpthread (or make cancel)
  $ ./cancel

Summary

- Signals
  - ✓ are a sort of software interrupts for application processes
  - ✓ provide a way of handling asynchronous events
  - ✓ SIGINT, SIGALRM, SIGCHLD, ….

- System calls for signals
  - ✓ signal
  - ✓ kill, raise
  - ✓ sleep, alarm, pause
  - ✓ pthread_cancel
  - ✓ pthread_setcancelstate, pthread_setcanceltype

E N D