CS 354 - Machine Organization & Programming
Tuesday, November 26, 2019

Project p5 (4.5%): DUE at 10 pm on Monday, December 2nd

Project p6 (4.5%): DUE at 10 pm on Saturday, December 14th

Homework hw7 (1.5%): DUE TOMORROW at 10 pm Wednesday, November 27th

Homework hw8 (1.5%): Assigned Tomorrow

Last Time

Kinds of Exceptions
Transferring Control via Exception Table
Exceptions in IA-32 & Linux
Processes and Context
User/Kernel Modes
Context Switch
Context Switch Example

Today

Meet Signals
Three Phases of Signaling
Processes IDs and Groups
Sending Signals
Receiving Signals

Next Time

Finish Signals
Multifile Coding
Linking and Symbols
Read: B&O 7.1 - 7.2

Meet Signals

What? A <u>*signal*</u> is a small message sent to a process via the kernel

Linux: 30 standard signal each given a unique positive int.

$kill -l (list of signals)

signal(7) $man 7 signal

Why?

- to notify user processes about
  1. low level hardware exceptions.
  2. high level events in kernel or user processes
- to enable user processes to communicate with each other
- to implement a highlevel software form of exceptional control flow.

Examples

1. divide by zero

exception #0 interrupts to kernel handler

- kernel signals user proc with #8 SIGFPE

2. illegal memory reference

exception #13 interrupts to kernel handler

- kernel signals user proc with #11 SIGSEGV

3. keyboard interrupt

- ctrl-c interrupts to kernel handler which
  signals #2 SIGINT signal, interrupts → for Program process terminates by default
- ctrl-z interrupts to kernel handler which
  signals #20 SIGTSTP terminal stop. suspend by default.

Three Phases of Signaling

Sending

- when the kernel exception handler runs in response to an exceptional event, or a signal from some process.
- is directed to a destinational process.

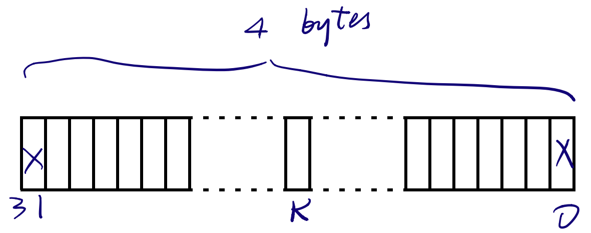
Delivering

when the kernel records a sent signal for its destination process.

<u>pending signal</u> is delieverd but not received

- each process has a bit vector for recording pending signals

<u>bit vectors</u> are kernel data structure where each bit has a distinct meaning



- bit k is set when signal k is delievered
  - - - - - cleaned - - - - received

Receiving

when the kernel causes the destination process to react to a pending signal.

- happens when the kernel transfers control back to a process
- multiple pending signals are done in order from low to high signal number

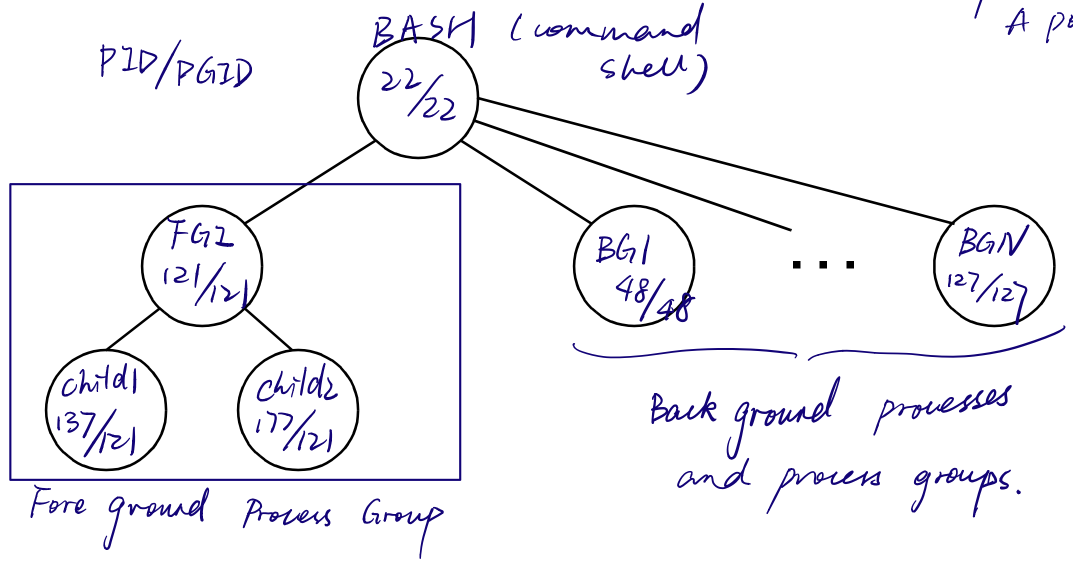
<u>blocking</u> prevent a signal from being received.

- enables a process to control which signal it pays attention to.
- each short process has a second bit vector to mark its blocked signals (1 == blocked)

Process IDs and Groups

What? Each process

- is identified by a pid, process ID. A positive int
- belongs to exait one group identified by PGID. (Process Group ID A positive int).



Why? numbers are far easier than names

How?

Recall:ps list processes
ps -U your processes ps -al all processes in long format

jobs lists command line initiated processes

getpid(2)get~~pgid~~pgrp(2)

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

`pid_t getpid(void)` returns calling processes PID

`pid_t getpgrp(void)` - — —— — - PGID.

Sending Signals

What? A signal is sent by the kernel or a user process via the kernel

From command line or in a program in a system using system calls

How? Linux Command

kill(1) sending a signal to specified destination process

kill -9 <pid>   9 is sigkill (kill process)

→ What happens if you kill your shell? logout

How? System Calls

kill(2) sends any signal from a calling process to a speifized destination process.

killpg(2) sends to entire program group.

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig)
```

↳ signal name or number

↳ destination process

↳ returns 0 on process, -1 on error

alarm(2) sets an alarm which deliever a sigalrm to calling process after a speificed number of seconds.

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds)
```

↳ delay till signal

↳ number of seconds remaining if preveious alarm still running.
otherwise, return 0.

Receiving Signals

What? A signal is received by its destination process by a default action or executing a pregramer specified signal handler.

How? Default Actions

- Terminate the process #2 sigint
- Terminate the process and dump core #11 sigsegv
- Stop the process #20 sigtstp
- Continue the process if it's currently stopped #18 sigcont.
- Ignore the signal #28 sigwinch

How? Signal Handler

1. code a signal handler
   - looks like a regular function but it's called by the kernel.
   - should not make unsafe system calls
     ~~E.g. printf okay in project p6~~.
2. Register the signal handler
   - casches one or more signals

~~signal(2)~~ dont use
sigaction(2) posix was for examining & changing signal actions.

Code Example

```
#include <signal.h>
#include ...
#include <string.h> // for memset function

void handler_SIGALRM() { ... }

int main(...) {
   struct sigaction sa;
   memset(&sa, 0, sizeof(sa)); //sa.sa_flags = 0;
   sa.sa_handler = handler_SIGALRM;
   if (sigaction(SIGALRM, &sa, NULL) != 0){
      printf("ERROR Binding SIGALRM HANDLER \n");
      exit(1);
   }
```