

BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

Process and Thread Signals

Notice: This set of slides is based on the notes by Professor Perrone of Bucknell and the textbook authors Silberschatz, Galvin, and Gagne, as well as the [tutorial](#) by Blaise Barney from Lawrence Livermore National Lab

Signals

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred, another mechanism of process communication.
- A **signal** is a piece of data deposited by the operating system
 - in a particular part in the system, e.g., a special purpose register, or a special area of memory
 - at the request of a process (user or system) to notify something of another process

Types of Unix Signals

- There are over 20 different types of Unix signals, some commonly used ones:
 - SIGALRM: send an alarm to a process, e.g., timer is up
 - SIGINT: inform the process that the user wants an interrupt, e.g., through Ctrl-C
 - SIGKILL: stop the process right away (kill)
 - SIGSEGV: inform the process that there is a virtual memory access violation (segmentation fault!)

http://en.wikipedia.org/wiki/Unix_signal

Signal Handling

- A **signal handler** is a program to process signals
 1. default
 2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
 - **User-defined signal handler** can override default
 - For single-threaded, signal delivered to process

Signal Handling with Threads

- Where should a signal be delivered for multi-threaded?
 - Deliver the signal to the thread to which the signal applies
 - Deliver the signal to every thread in the process
 - Deliver the signal to certain threads in the process
 - Assign a specific thread to receive all signals for the process

A Simple Example (*main()*)

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
time_t curtime;
int main(int argc, char *args[]) {
    int seconds;
    void sigcatch();
    if (argc != 2) {
        fprintf(stderr, "usage: %s num-sec\n", args[0]);
        exit(1);
    }
    seconds = atoi(args[1]);
    time(&curtime);
    printf("Started the timer:\n%s", ctime(&curtime));
    alarm(seconds);
    signal(SIGALRM, sigcatch);
    while(1); // busy waiting
}
```

A Simple Example (*handler()*)

```
void sigcatch() {  
    time(&curtime);  
    printf("Caught alarm at:\n%s", ctime(&curtime));  
    exit(1);  
}
```

[xmeng@polaris code]\$./a.out 2

Started the timer:

Wed Sep 18 11:16:03 2013

Caught alarm at:

Wed Sep 18 11:16:05 2013

[xmeng@polaris code]\$

Program asks for a 2-second timer

Timer started

Timer expired