(http://www.yolinux.com/)

C/C++ signal handling

C and C++ signal handling and C++ signal classes and examples.

Tutorial Table of Contents:

Search

Related YoLinux Tutorials:

°Linux and C++ (LinuxTutorialC++.html)

°C++ Unions & Structures (LinuxTutorialC++Structures.html)

°C++ Templates (Cpp-Templates.html)

°C++ STL (LinuxTutorialC++STL.html)

°C++ STL Map (CppStlMultiMap.html)

°C++ String Class (LinuxTutorialC++StringClass.html)

°C++ Singleton (C++Singleton.html)

°C++ Coding Style (LinuxTutorialC++CodingStyle.html)

°C++ XML Beans (C++XmlBeansxx.html)

°C/C++ Dynamic Memory (Cpp-DynamicMemory.html)

°C++ Memory Corruption (C++MemoryCorruptionAndMemoryLeaks.html)

°C++ CGI (LinuxTutorialC++CGI.html)

°Google C++ Unit Test (Cpp-GoogleTest.html)

°Jenkins CI (Jenkins.html)

°Jenkins Plugins for C++ (Jenkins-Cpp-builds.html)

°YoLinux Tutorials Index (/TUTORIALS/)

Signals:

Description: Signals are software interrupts delivered to a process by the operating system. Signals can also be issued by the operating system based on system or error conditions. There is a default behavior for some (i.e. a process is terminated when it receives an inturrupt `SIGINT` signal by pressing keystrokes ctrl-C) but this tutorial shows how to handle the signal by defining callback functions to manage the signal. Where possible, this allows one to close files and perform operations and react in a manner defined by the programmer.

Note that not all signals can be handled.

Types of signals:

Signal	Value	Description
SIGHUP	1	Hangup (POSIX) Report that user's terminal is disconnected. Signal used to report the termination of the controlling process.
SIGINT	2	Interrupt (ANSI) Program interrupt. (ctrl-c)
SIGQUIT	3	Quit (POSIX) Terminate process and generate core dump.
SIGILL	4	Illegal Instruction (ANSI) Generally indicates that the executable file is corrupted or use of data where a pointer to a function was expected.
SIGTRAP	5	Trace trap (POSIX)
SIGABRT SIGIOT	6	Abort (ANSI) IOT trap (4.2 BSD) Process detects error and reports by calling abort
SIGBUS	7	BUS error (4.2 BSD) Indicates an access to an invalid address.
SIGFPE	8	Floating-Point arithmetic Exception (ANSI). This includes division by zero and overflow.The IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) defines various floating-point exceptions.
SIGKILL	9	Kill, unblockable (POSIX) Cause immediate program termination. Can not be handled, blocked or ignored.
SIGUSR1	10	User-defined signal 1
SIGSEGV	11	Segmentation Violation (ANSI) Occurs when a program tries to read or write outside the memory that is allocated for it by the operating system, dereferencing a bad or NULL pointer. Indicates an invalid access to valid memory.
SIGUSR2	12	User-defined signal 2
SIGPIPE	13	Broken pipe (POSIX) Error condition like trying to write to a socket which is not connected.

Signal	**Value**	**Description**
SIGALRM	14	Alarm clock (POSIX) Indicates expiration of a timer. Used by the `alarm()` function.
SIGTERM	15	Termination (ANSI) This signal can be blocked, handled, and ignored. Generated by "kill" command.
SIGSTKFLT	16	Stack fault
SIGCHLD SIGCLD	17	Child status has changed (POSIX) Signal sent to parent process whenever one of its child processes terminates or stops. See the YoLinux.com Fork, exec, wait, waitpid tutorial (ForkExecProcesses.html)
SIGCONT	18	Continue (POSIX) Signal sent to process to make it continue.
SIGSTOP	19	Stop, unblockable (POSIX) Stop a process. This signal cannot be handled, ignored, or blocked.
SIGTSTP	20	Keyboard stop (POSIX) Interactive stop signal. This signal can be handled and ignored. (ctrl-z)
SIGTTIN	21	Background read from tty (POSIX)
SIGTTOU	22	Background write to tty (POSIX)
SIGURG	23	Urgent condition on socket (4.2 BSD) Signal sent when "urgent" or out-of-band data arrives on a socket.
SIGXCPU	24	CPU limit exceeded (4.2 BSD)
SIGXFSZ	25	File size limit exceeded (4.2 BSD)
SIGVTALRM	26	Virtual Time Alarm (4.2 BSD) Indicates expiration of a timer.
SIGPROF	27	Profiling alarm clock (4.2 BSD) Indicates expiration of a timer. Use for code profiling facilities.
SIGWINCH	28	Window size change (4.3 BSD, Sun)
SIGIO SIGPOLL	29	I/O now possible (4.2 BSD) Pollable event occurred (System V) Signal sent when file descriptor is ready to perform I/O (generated by sockets)
SIGPWR	30	Power failure restart (System V)
SIGSYS	31	Bad system call

See: `/usr/include/bits/signum.h`

Signals which can be processed include: `SIGINT, SIGABRT, SIGFPE, SIGILL, SIGSEGV, SIGTERM, SIGHUP`

List all signals available to the system:

Use the command: `kill -l`

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1
36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4  39) SIGRTMIN+5
40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8  43) SIGRTMIN+9
44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13
52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9
56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6  59) SIGRTMAX-5
60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1
64) SIGRTMAX
```

Sending a process a signal:

A process can be sent a signal using the "`kill`" command: `kill -s` *`signal-number pid`*
Where the pid (process id) can be obtained using the "`ps`" command.

C Signal handler and Example:

Basic C signal callback function example:

File: `signalExample.cpp`

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

// Define the function to be called when ctrl-c (SIGINT) signal is sent to process
void
signal_callback_handler(int signum)
{
   printf("Caught signal %d\n",signum);
   // Cleanup and close up stuff here

   // Terminate program
   exit(signum);
}

int main()
{
   // Register signal and signal handler
   signal(SIGINT, signal_callback_handler);

   while(1)
   {
      printf("Program processing stuff here.\n");
      sleep(1);
   }
   return EXIT_SUCCESS;
}
```

Example to handle ctrl-c

Compile: `gcc signalExample.cpp`

Run: `a.out`

Results:

```
Program processing stuff here.
Program processing stuff here.
Caught signal 2
```

The function prototype: `void (*signal (int sig, void (*func)(int)))(int);`

C++ Signal Registration and Handling Class:

File: `signalHandler.hpp`

```
#ifndef __SIGNALHANDLER_H__
#define __SIGNALHANDLER_H_
#include <stdexcept>
using std::runtime_error

class SignalException : public runtime_error
{
public:
   SignalException(const std::string& _message)
      : std::runtime_error(_message)
   {}
};

class SignalHandler
{
protected:
    static bool mbGotExitSignal;

public:
    SignalHandler();
    ~SignalHandler();

    static bool gotExitSignal();
    static void setExitSignal(bool _bExitSignal);

    void    setupSignalHandlers();
    static void exitSignalHandler(int _ignored);

};
#endif
```

File: `signalHandler.cpp`

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

#include "signalHandler.hpp

bool SignalHandler::mbGotExitSignal = false;

/**
* Default Contructor.
*/
SignalHandler::SignalHandler()
{
}

/**
* Destructor.
*/
SignalHandler::~SignalHandler()
{
}

/**
* Returns the bool flag indicating whether we received an exit signal
* @return Flag indicating shutdown of program
*/
bool SignalHandler::gotExitSignal()
{
    return mbGotExitSignal;
}

/**
* Sets the bool flag indicating whether we received an exit signal
*/
void SignalHandler::setExitSignal(bool _bExitSignal)
{
    mbGotExitSignal = _bExitSignal;
}

/**
* Sets exit signal to true.
* @param[in] _ignored Not used but required by function prototype
*                     to match required handler.
*/
void SignalHandler::exitSignalHandler(int _ignored)
{
    mbGotExitSignal = true;
}

/**
* Set up the signal handlers for CTRL-C.
*/
void SignalHandler::setupSignalHandlers()
{
    if (signal((int) SIGINT, SignalHandler::exitSignalHandler) == SIG_ERR)
    {
        throw SignalException("!!!!! Error setting up signal handlers
!!!!!");
    }
}
```

File: test.cpp

```
#include <iostream>
#include <unistd>
#include <stdlib.h>
#include "signalHandle.hpp"
using namespace std;

main()
{
  int iret;

  try
  {
    SignalHandler signalHandler;

    // Register signal handler to handle kill signal
    signalHandler.setupSignalHandlers();

    // Infinite loop until signal ctrl-c (KILL) received
    while(!signalHandler.gotExitSignal())
    {
        sleep(1);
    }

    iret = EXIT_SUCCESS;
  }
  catch (SignalException& e)
  {
    std::cerr << "SignalException: " << e.what() << std::endl;
    iret = EXIT_FAILURE;
  }
  return(iret);
}
```

Compile: `g++ signalHandle.cpp test.cpp`

C Signal Man Pages:

C functions:

- signal (http://man.yolinux.com/cgi-bin/man2html?cgi_command=signal) - ANSI C signal handling
- raise (http://man.yolinux.com/cgi-bin/man2html?cgi_command=raise) - send a signal to the current process
- strsignal (http://man.yolinux.com/cgi-bin/man2html?cgi_command=strsignal) - return string describing signal (GNU extension)
- psignal (http://man.yolinux.com/cgi-bin/man2html?cgi_command=psignal) - print signal message
- sigaction (http://man.yolinux.com/cgi-bin/man2html?cgi_command=sigaction) - POSIX signal handling functions
- sigsetops (http://man.yolinux.com/cgi-bin/man2html?cgi_command=sigsetops) - POSIX signal set operations
- sigvec (http://man.yolinux.com/cgi-bin/man2html?cgi_command=sigvec) - BSD software signal facilities
- alarm (http://man.yolinux.com/cgi-bin/man2html?cgi_command=alarm) - set an alarm clock for delivery of a signal

Commands:

- kill (http://man.yolinux.com/cgi-bin/man2html?cgi_command=kill) - terminate a process
- ps (http://man.yolinux.com/cgi-bin/man2html?cgi_command=ps) - report a snapshot of the current processes.

Books:

C++ How to Program by Harvey M. Deitel, Paul J. Deitel ISBN #0131857576, Prentice Hall Fifth edition. The first edition of this book (and Professor Sheely at UTA) taught me to program C++. It is complete and covers all the nuances of the C++ language. It also has good code examples. Good for both learning and reference.	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/0131857576/&tag=yolinux-20)
"Advanced UNIX Programming" **Second Edition** by Marc J. Rochkind ISBN # 0131411543, Addison-Wesley Professional Computing Series	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/0131411543/&tag=yolinux-20)

"Advanced Programming in the UNIX Environment" **First Edition** by W. Richard Stevens ISBN # 0201563177, Addison-Wesley Professional Computing Series It is the C programmers guide to programming on the UNIX platform. This book is a must for any serious UNIX/Linux programmer. It covers all of the essential UNIX/Linux API's and techniques. This book starts where the basic C programming book leaves off. Great example code. This book travels with me to every job I go to.	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/0201563177/&tag=yolinux-20)
"UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI" **Second Edition** by W. Richard Stevens ISBN # 013490012X, Prentice Hall PTR This book covers network APIs, sockets + XTI, multicast, UDP, TCP, ICMP, raw sockets, SNMP, MBONE. In depth coverage of topics.	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/013490012X/&tag=yolinux-20)

"UNIX Network Programming Volume 2: Interprocess Communications" by W. Richard Stevens ISBN # 0130810819, Prentice Hall PTR This book covers semaphores, threads, record locking, memory mapped I/O, message queues, RPC's, etc.	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/0130810819/&tag=yolinux-20)
"Advanced Unix Programming" by Warren W. Gay ISBN # 067231990X, Sams White Book Series This book covers all topics in general: files, directories, date/time, libraries, pipes, IPC, semaphores, shared memory, forked processes and I/O scheduling. The coverage is not as in depth as the previous two books (Stevens Vol 1 and 2)	(http://www.amazon.com/gp/redirect.html?ie=UTF8&location=http://www.amazon.com/exec/obidos/ASIN/067231990X/&tag=yolinux-20)

(http://www.addthis.com/bookmark.php?v=250&pub=yolinux)

Advertisements

to top of page