Signals in Unix

Sistemas Operativos, 2004-2005

Sending signals using the keyboard

Ctrl-C

 Pressing this key causes the system to send an INT signal (SIGINT) to the running process. By default, this signal causes the process to immediately terminate.

■ <u>Ctrl-Z</u>

 Pressing this key causes the system to send a TSTP signal (SIGTSTP) to the running process. By default, this signal causes the process to suspend execution.

∎ <u>fg</u>

 On most shells, using the 'fg' command will resume execution of the process (that was suspended with Ctrl-Z), by sending it a SIGCONT signal.

2

Sending Signals using kill

- ∎ <u>kill</u>
 - kill -<signal> <PID>
 - For example, in order to send the INT signal to process with PID 5342, type:

kill -INT 5342

This has the same affect as pressing $\mbox{Ctrl-C}$ in the shell that runs that process.

 If no signal name or number is specified, the default is to send a SIGTERM signal to the process, which normally causes its termination, and hence the name of the kill command.

Sending Signals with System Calls

int kill(pid_t pid, int sig);

#include <unistd.h> #include <sys/types.h> #include <signal.h>

```
//.....
/*first, find my own process ID */
pid_t my_pid = getpid();
/* now that i got my PID, send myself the STOP
signal. */
kill(my_pid, SIGSTOP);
```

The signal() System Call

#include <stdio.h> #include <unistd.h> #include <sys/types.h> #include <signal.h>

/* first, here is the signal handler */ void catch_int(int sig_num) { /* re-set the signal handler again to catch_int, for next time */ signal(SIGINT, catch_int); /* and print the message */ print(T+P nao vale a pena fazer Ctrl-C..."); fflush(stdout);

Theorem ... */ // ... */ * and somewhere later in the code.... */ /* set the INT (Ctrl-Q) signal handler to 'catch_int' */ signal(SiGINT, catch_int); /* now, lets get into an infinite loop of doing nothing. */

for (;;) pause();

Another Example

6

#include <stdio.h>
#include <signal.h>
void handler_alarm();
int main()
{

{ signal(SIGALRM,handler_alarm); Signal(SIGALRW,nandler_alarm); printf("Vou dormir por algum tempo ...\n"); pause(); printf("...vou ficar na sorna mais 5 segundos\n"); sleep(5); exit(1);

exi(1); } void handler_alarm() { printf("ALARM !!!! \n"); fflush(stdout); }

Another Example

int signal_ocorreu=0; main(){

Signal(SIGUSR1, rotina_excepcao);

//...Quando é que posso saber que ocorreu um SIGUSR1?... if(signal_ocorreu)...

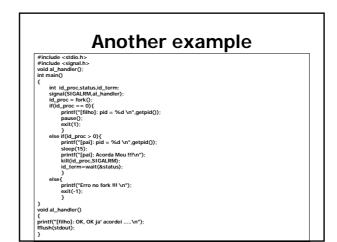
// vai para ali

else // vai para acoli

pause();

}

Void rotina_excepcao(){ Signal_ocorreu=1; }



SIGHUP	1	Hangup detected on controlling terminal or death of controlling process
SIGOUIT	2	Interrupt from keyboard Quit from keyboard
SIGUIT		Illegal Instruction
SIGABRT		Abort signal from abort(3)
SIGEPE	8	Floating point exception
SIGKILI		Kill signal
SIGSEGV		Invalid memory reference
SIGPIPE		Broken pipe: write to pipe with no readers
SIGALRM		Timer signal from alarm(2)
SIGTERM	15	Termination signal
SIGUSR1	30,10,16	User-defined signal 1
SIGUSR2	31,12,17	User-defined signal 2
SIGCHLD	20,17,18	
SIGCONT	19,18,25	Continue if stopped
	17,19,23	
	18,20,24	Stop typed at tty
	21,21,26	tty input for background process
SIGTTOU	22,22,27	D tty output for background process

Г



 SIG_DFL:
 Causes the system to set the default signal handler for the given signal (i.e. the same handler the system would have assigned for the signal when the process started running): signal(SIGTSTP, SIG_DFL);

10

- Saving/Restoring Signal Handlers:
 - old_routine = signal(SIGQUIT,new_routine);
 - signal(SIGQUIT,old_routine);

"Do" and "Don't" inside A **Signal Handler**

- Make it short the signal handler should be a short function that returns guickly. Instead of doing complex operations inside the signal handler, it is better that the function will raise a flag (e.g. a global variable) and have the main program check that flag occasionally.
 Proper Signal Masking don't be too lazy to define proper signal masking for a signal handler, preferably using the sigaction() system call.
 Careful with "fault" signals if you cach signals that indicate a program bug (SIGBUS, SIGSEGV, SIGFPE), don't try to be too smart and let the program continue just do the minimal required cleanup, and exit, preferably with a core dump (using the abort() function). Such signals usually indicate a bug in the program, that if ignored will most likely cause it to crush sooner or later, making you think the problem is somewhere else in the code. .
- •
- In the code. <u>Careful with timers</u> when you use timers, remember that you can only use one timer at a time. <u>Signals are NOT an event driven framework</u> it is easy to get carried away and try turning the signals system into an event-driven driver for a program, but signal handling functions were not meant for that. .

11

 <u>SIG_IGN</u>:
 Causes the process to ignore the specified signal. For example, in order to ignore Ctrl-C completely (useful for programs that must NOT be interrupted in the middle, or in critical sections), write this: signal(SIGINT, SIG_IGN);