

بخش سوم

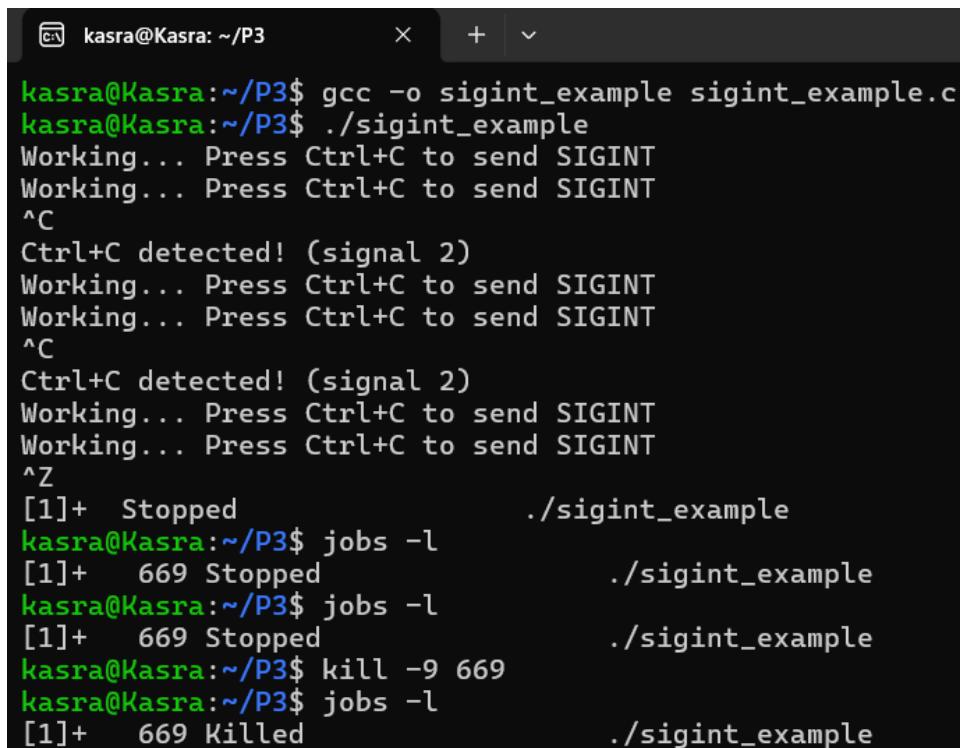
کسرا نوربخش 810100230

مقدمه

در این بخش به کنترل فرآیندها و بازپس گیری ترمینال در لینوکس پرداختیم.

پاسخ

ابتدا مساله برنامه در این بود که وقتی سیگنال SIGINT را دریافت می کرد یک پیام چاپ می کرد و هیچ گاه از حلقه بی پایان بیرون نمی آمد، در ابتدا با استفاده از Ctrl+z آن را به background بردیم تا شل را پس گیریم. سپس با استفاده از -l توانستیم آن را پیدا کنیم و با استفاده از kill -9 آن را terminate کنیم.



```
kasra@Kasra:~/P3$ gcc -o sigint_example sigint_example.c
kasra@Kasra:~/P3$ ./sigint_example
Working... Press Ctrl+C to send SIGINT
Working... Press Ctrl+C to send SIGINT
^C
Ctrl+C detected! (signal 2)
Working... Press Ctrl+C to send SIGINT
Working... Press Ctrl+C to send SIGINT
^C
Ctrl+C detected! (signal 2)
Working... Press Ctrl+C to send SIGINT
Working... Press Ctrl+C to send SIGINT
^Z
[1]+  Stopped                  ./sigint_example
kasra@Kasra:~/P3$ jobs -l
[1]+  669 Stopped                  ./sigint_example
kasra@Kasra:~/P3$ jobs -l
[1]+  669 Stopped                  ./sigint_example
kasra@Kasra:~/P3$ kill -9 669
kasra@Kasra:~/P3$ jobs -l
[1]+  669 Killed                 ./sigint_example
```

عکس 1: روند بخش 1

در بخش 2 کد را به نحوی تغییر دادیم که در صورت دریافت سیگنال های دیگر هم، سیگنال هندلر قبلی صدا زده شود:

```
void handle_signal(int sig) {
    printf("\nCaught signal %d — you can't exit with that!\n", sig);
}
```

```

int main() {

    // Catch Ctrl+C (SIGINT), Ctrl+Z (SIGTSTP), and Ctrl+\ (SIGQUIT)

    signal(SIGINT, handle_signal); // Ctrl+C
    signal(SIGTSTP, handle_signal); // Ctrl+Z
    signal(SIGQUIT, handle_signal); // Ctrl+\

    while (1) {

        printf("Working... You can't escape with Ctrl+C/Z/\\\\n");
        sleep(2);

    }

    return 0;
}

```

سیگنال SIGINT که در واقع با زدن Ctrl+C به پردازه فرستاده می شود، پردازه را خاتمه می یابد. سیگنال SIGTSTP به پردازه فرستاده می شود، پردازه را متوقف می کند و در آخر هم سیگنال SIGQUIT که با زدن Ctrl+\ به پردازه می رود، از کیبورد quit می دهد و باعث خاتمه بافتن با core dump می شود. در اینجا با استفاده از یک ترمینال دیگر، PID را یافتیم و به مانند قبل آن را خاتمه دادیم. (اگر آن را در یک سشن ssh داشتیم می توانستیم با زدن .~ و یا بستن ترمینال هم آن را kill کنیم)

The screenshot shows two terminal windows. The left window displays the output of the program 'sighamechi_example' which prints 'Working...' repeatedly and handles signals by printing messages like 'Caught signal 2 - you can't exit with that!' and 'Caught signal 20 - you can't exit with that!'. The right window shows the user running 'ps aux | grep sighamechi_example' to find the process ID (816), then killing it with 'kill -9 816'.

```

kasra@Kasra:~/P3$ ./sighamechi_example
Working... You can't escape with Ctrl+C/Z/\
Working... You can't escape with Ctrl+C/Z/\
^C
Caught signal 2 - you can't exit with that!
Working... You can't escape with Ctrl+C/Z/\
^Z
Caught signal 20 - you can't exit with that!
Working... You can't escape with Ctrl+C/Z/\
^\
Caught signal 3 - you can't exit with that!
Working... You can't escape with Ctrl+C/Z/\
Killed
kasra@Kasra:~/P3$ |

kasra@Kasra:~$ ps aux | grep sighamechi_example
kasra      816  0.0  0.0  2684  1032 pts/0    S+   21:39  0:00 ./sighamechi_example
kasra      820  0.0  0.0  4100  1932 pts/2    S+   21:39  0:00 grep --color=auto sighamechi_example
kasra@Kasra:~$ kill -9 816
kasra@Kasra:~$ |

```

عکس 2: روند بخش 2 (کلید ها را هم در ابتدا برای خارج شدن تست کردیم که به درستی کار نکردند!!!!)

تفاوت این دو در این است که **signal** ساده است در سرتاسر سیستما ها یکسان کار نمی کند و همانطور که دیدیم می توانیم آن ها را **override** کنیم و آن گونه که می خواهیم آن ها را هندل کنیم. اما **sigaction** رو باست تر می باشد و می توان برای آن فلگ هایی را تعریف نمود و مناسب برنامه هایی می باشد که سیگنال هندلینگ جدی و پیش بینی پذیر می خواهد.

کد را تغییر دادیم تا بجای استفاده از `signal` استفاده کند و خروجی به صورت زیر بود:

```

kasra@Kasra:~/P3$ ./sigaction_example
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^C
Caught signal 2 - you can't escape this way!
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^Z
Caught signal 20 - you can't escape this way!
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^\
Caught signal 3 - you can't escape this way!
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^\
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^\
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^\
Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\^\
Killed
kasra@Kasra:~/P3$ |
```

```

kasra@Kasra:~$ ps aux | grep sigaction_example
kasra    1037  0.0  0.0  2684  1056 pts/0   S+  21:50  0:00 ./sigaction_example
kasra    1039  0.0  0.0  4096  2020 pts/2   S+  21:50  0:00 grep --color=auto sigaction_examp
kasra@Kasra:~$ kill -9 1037
kasra@Kasra:~$ |
```

عکس ۳: روند بخش ۳ (کلید ها را هم در ابتدا برای خارج شدن تست کردیم که به درستی کار نکردند!!!!)

لينک های استفاده شده

برای پیش برد این بخش از ChatGPT کمک گرفته شد که لینک آن:

<https://chatgpt.com/share/68093038-6e98-800d-bafe-7b29fe530c23>

چت کامل:

How can I compile and run this code: #include <stdio.h> #include <signal.h> #include <unistd.h> void handle_sigint(int sig) { printf("\nCtrl+C detected! (signal %d)\n", sig); } int main() { signal(SIGINT, handle_sigint); // Handle Ctrl+C while (1) { printf("Working... Press Ctrl+C to send SIGINT\n"); sleep(2); } return 0; }

ChatGPT said:

To compile and run your C program on a Unix-like system (Linux/macOS), follow these steps:

1. Save the code to a file

Let's say you name it `siginint_example.c`.

2. Open a terminal and compile it

Use `gcc` (the GNU C compiler):

bash

CopyEdit

`gcc -o sigint_example sigint_example.c`

-o `sigint_example`: This tells the compiler to name the output file `sigint_example`. ●



3. Run the compiled program

bash

CopyEdit

`./sigint_example`

4. Test Ctrl+C handling

While it's running, press Ctrl+C in the terminal. •

You should see: •

SCSS

CopyEdit

Ctrl+C detected! (signal 2)

Instead of terminating the program, the signal handler catches SIGINT (signal number 2), and the program continues running.

Let me know if you're using Windows or want to try this inside a specific environment like WSL, Code::Blocks, or VSCode.

40

You said:

In the terminal that the program is running, I want to take back the shell and terminate the process that the program is run on using CLI commands.

ChatGPT said:

To **take back the shell and terminate** your program from the command line (without pressing Ctrl+C), you can do it by interacting with the process directly.

💡 Step-by-step using CLI:

1. Run the program in the background

Start it with & to run in the background:

bash

CopyEdit

`./sigint_example &`

You'll see something like:

csharp

CopyEdit

[1] 12345

Where 12345 is the **PID** (Process ID).

2. Check background jobs (optional)

bash

CopyEdit

jobs

This shows your current background jobs. For example:

bash



[CopyEdit](#)

[1]+ Running ./.sigint_example &

3. Terminate the process using its PID

Use kill with the PID you saw:

bash

[CopyEdit](#)

kill -SIGINT 12345

Or just:

bash

[CopyEdit](#)

kill -2 12345

Both send SIGINT to the process, which triggers your custom handler.

If you want to **forcefully terminate it**, use:

bash

[CopyEdit](#)

kill -9 12345

(SIGKILL, which cannot be caught or ignored)

4. Bring it back to foreground and terminate manually

If you didn't background it with &, or want to bring it back:

bash

[CopyEdit](#)

fg

Then hit Ctrl+C if desired.

Let me know if you're trying this in a script, remote session, or need to automate the kill behavior!

40

You said:

No, the problem with the code is that when I use Ctrl+c in the terminal that the code is running it won't end. I wanted to use commands in the terminal that the code is being run on to take back the shell and terminate the process. (without changing the code)

ChatGPT said:

Got it — your program **catches SIGINT (Ctrl+C)**, so it **won't terminate** with Ctrl+C anymore. Since you're not allowed to change the code, here's how to **reclaim your terminal and stop the program from that same terminal** it's running in.

Here's how to do it without editing the code:

◊ 1. Send it to background with Ctrl+Z

In the terminal where it's running:



bash

CopyEdit

Ctrl+Z

This suspends the program and gives you the prompt back. You'll see something like:

bash

CopyEdit

```
[1]+ Stopped      ./sigint_example
```

❖ **2. Kill it with kill**

Now get its process ID (PID):

bash

CopyEdit

jobs -l

Sample output:

bash

CopyEdit

```
[1]+ 12345 Stopped      ./sigint_example
```

Then kill it:

bash

CopyEdit

kill 12345

This sends SIGTERM (default), which your program hasn't overridden, so it **will terminate**.

❖ **3. Or send SIGKILL to guarantee it dies**

If the process ignores SIGTERM too:

bash

CopyEdit

kill -9 12345

Bonus: One-liner version

bash

CopyEdit

kill -9 \$(jobs -p)

This kills all background jobs.

Let me know if you want to automate this, or handle multiple programs at once.

40

You said:

now change the code, in a way that using other keys we cannot get out of it (like Ctrl+c, Ctrl+z, Ctrl+\,...) tell me about each signal and now tell how can I get back terminal now with the new code.



ChatGPT said:

You're asking to modify the code so that **common terminal escape sequences like Ctrl+C, Ctrl+Z, and Ctrl+\ no longer work** — essentially making it a stubborn, persistent program that **traps all termination/suspend signals**.

Let's break this into:

 Step 1: Modify the code to trap SIGINT, SIGTSTP, SIGQUIT

Here's the updated C code:

C

CopyEdit

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

```
void handle_signal(int sig) {
    printf("\nCaught signal %d — you can't exit with that!\n", sig);
}
```

```
int main() {
```

```
// Catch Ctrl+C (SIGINT), Ctrl+Z (SIGTSTP), and Ctrl+\ (SIGQUIT)
    signal(SIGINT, handle_signal); // Ctrl+C
    signal(SIGTSTP, handle_signal); // Ctrl+Z
    signal(SIGQUIT, handle_signal); // Ctrl+\
```

```
    while (1) {
```

```
        printf("Working... You can't escape with Ctrl+C/Z/\\\\n");
        sleep(2);
    }
```

```
    return 0;
}
```

 What Each Signal Does:

Signal	Description	Default Shortcut	Default Behavior	What We Did
SIGINT	Interrupt from keyboard	Ctrl+C	Terminate the process	Overridden with handler
SIGTSTP	Terminal stop (suspend)	Ctrl+Z	Suspend (pause) process	Overridden with handler
SIGQUIT	Quit from keyboard	Ctrl+\	Terminate with core dump	Overridden with handler

Now **none of those shortcuts** will terminate or suspend the process.

Step 2: So... how do you take back the terminal now?

Even with all escape keys blocked, you **can still take control using external tools**. Here are your options:

◊ 1. Find the PID and kill it from another terminal

If you have another terminal window or SSH session:

bash

CopyEdit

```
ps aux | grep sigint_example
```

```
kill -9 <PID>
```

Or more specifically:

bash

CopyEdit

```
pkill -9 sigint_example
```

◊ 2. Reclaim from the same terminal (if stuck in foreground)

If you're **in the same terminal** and can't even suspend with Ctrl+Z:

● Press Ctrl+\ → This won't work anymore.

So now you're **stuck**. Use:

Escape plan:

Hit the **~. escape sequence** (only works in SSH sessions), **or** just close the terminal tab or window. It won't gracefully shut it down, but it will kill the terminal and with it, the process.

◊ 3. Use tty and pkill

From another terminal:

bash

CopyEdit

```
tty      # Get terminal name (e.g., /dev/pts/1)
```

```
ps -t pts/1
```

Then kill the process by PID or with:

bash

CopyEdit

```
pkill -t pts/1
```

◊ 4. Use kill -9 with job control

If you're lucky and you did background it first:

bash

CopyEdit



Ctrl+Z

kill -9 %1

🔗 Extreme: Kernel level

If it forks or resists everything, you'd need to use kill -9 from root or htop to nuke it.

Want to take it further and trap even more signals (like SIGHUP, SIGTERM, etc.)? I can help you build an “unkillable” test program — just let me know 😊

40

You said:

what is the difference between sigaction and signal? now do the same with it.

ChatGPT said:

Great question! Let's start with a quick breakdown of the **difference between signal() and sigaction()**, then we'll rewrite your signal-blocking program using sigaction.

🔗 signal() vs sigaction()

signal()

- Simple and portable.
- Not consistent across all systems (behavior may vary).
- Can be overridden by system calls after being set.
- **Limited control** over how the signal is handled.

sigaction()

- More robust and **POSIX-compliant**.
- Provides fine-grained control:
- Can specify a **signal mask** (to block other signals while handling one).
- Can set flags (like SA_RESTART, SA_SIGINFO, etc.).
- Recommended for **serious applications** that need predictable signal handling.

🔗 Rewriting your code with sigaction()

Here's the version that blocks Ctrl+C, Ctrl+Z, Ctrl+\ using sigaction:

C

CopyEdit

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



```

void handle_signal(int sig) {
    printf("\nCaught signal %d — you can't escape this way!\n", sig);
}

int main() {
    struct sigaction sa;

    sa.sa_handler = handle_signal;
    sigemptyset(&sa.sa_mask); // No additional signals blocked during handler
    sa.sa_flags = SA_RESTART; // Restart interrupted system calls (e.g., sleep)

    // Set up handlers for common terminal escape signals
    sigaction(SIGINT, &sa, NULL); // Ctrl+C
    sigaction(SIGTSTP, &sa, NULL); // Ctrl+Z
    sigaction(SIGQUIT, &sa, NULL); // Ctrl+\

    while (1) {
        printf("Working... You can't escape with Ctrl+C, Ctrl+Z, or Ctrl+\\\n");
        sleep(2);
    }
}

return 0;
}

```

❖ What's New with sigaction()?

Feature	Description
sa_handler	Function to handle the signal (like signal() uses).
sa_mask	Which signals should be blocked while handling this one.
sa_flags	Behavior tweaks (SA_RESTART is commonly used to avoid interrupted syscalls).

☛ How to Escape Now?

Same as before — even sigaction() can't trap everything:

Use one of these:

- kill -9 <PID> from another terminal
- pkill -9 <name>
- ps aux | grep <name> → get PID → kill -9 <PID>



- ◊ Close the terminal window/tab (force kill) •
- ◊ tty → pkill -t pts/X from another terminal •

Want to go deeper? You could trap **all catchable signals** (SIGHUP, SIGTERM, SIGUSR1, etc.), or even use sigaction with SA_SIGINFO for advanced diagnostics. Let me know if you're curious!