Advanced Programming
COMS 3157

April 18, 2025

Patrick Shen

pts2125@columbia.edu

---

1. Q1? (2 marks)
   (a) What is a signal?
   (b) What is a signal handler?

Ans:

(a) A small message that notifies a process that an event of some type has occured.

(b) A signal handler is a function that executes in response to the arrival and consumption of a signal. The signal handler *runs in the process that recieves the signal.*

2. Give the scenario where each signal would occur. (4 marks)
   (a) SIGFPE
   (b) SIGINT
   (c) SIGTSTP
   (d) SIGCONT

Ans:

(a) SIGFPE: Whenever a process commits an integer-divide-by-zero, the kernel signals a **SIGFPE** signal to the offending process.

(b) SIGINT: When you type ctrl-c, the kernel sends a **SIGINT** to the foreground process (and by default, that foreground is terminated).

(c) SIGTSTP: When you type ctrl-z, the kernel issues a **SIGTSTP** to the foreground process (and by default, the foreground process is halted unti la subsequent **SIGCONT** signal instructs it to continue).

(d) SIGCONT: When a process attempts to publish data to the write end of a pipe after the read end has been closed, the kernel sends a **SIGPIPE** to the offending process.

3. Give the following actions for the predefined signal function handlers in signal() (2 marks)
   (a) SIG_DFL
   (b) SIG_IGN

Ans:

(a) SIG_DFL: clears any custom function handler for signal.
clears ”somehandler” signal function

```c
int main(void)
{
...
signal(SIGINT, somehandler);
...
signal(SIGINT, SIG_DFL);
...
}

```

Listing 1: SIG_DFL example

(b) SIG_IGN: ignores signals

```c
int main(void)
{
...
signal(SIGINT, SIG_IGN);
...
}

```

Listing 2: SIG_IGN example

4. Which two signals cannot be handled? (2 marks)

Ans:

(a) SIGKILL (9)

(b) SIGSTOP (19)

5. Briefly explain each argument in for sigaction(int signum, const struct sigaction *act, struct sigaction *oldact); (3 marks)
   (a) int signum
   (b) struct *act

> **Ans:**
> (a) int signum: signal number to handle
>
> (b) struct *act: pointer to a struct sigaction describing the new signal handler

6. Briefly explain each field in the sigaction struct (4 marks)
   (a) void (*sa_handler)(int);
   (b) void (*sa_sigaction)(int, siginfo_t *, void *);
   (c) sigset_t sa_mask;
   (d) int sa_flags;

```c
    struct sigaction {
        void     (*sa_handler)(int);
        void     (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t   sa_mask;
        int        sa_flags;
        void     (*sa_restorer)(void); // obsolete, ignore
    };

```

Listing 3: sigaction struct

> **Ans:**
> (a) void (*sa_handler)(int): function pointer to custom function that handles signal
>
> (b) void (*sa_sigaction)(int, siginfo_t *, void *): a more advanced signal handler (alternative to sa_handler)
>
> (c) sigset_t sa_mask: A set of signals to block during the execution of the handler. Prevents specific signals from interrupting the current handler.
>
> (d) int sa_flags: Modifies behavior of the signal handler.

7. Briefly explain what each function does for sa_mask in the sigaction struct (3 marks)

(a) int sigemptyset(sigset_t *set)

(b) int sigaddset(sigset_t *set, int signum)

(c) int sigfillset(sigset_t *set)

Ans:

(a) int sigemptyset(sigset_t *set): clears all signals that are blocked by the signal handler

(b) int sigaddset(sigset_t *set, int signum): adds a signal specified by signum to the set of signals blocked by the signal handler

(c) int sigfillset(sigset_t *set): blocks all signals, except SIGKILL (9) and SIGSTOP (19)

8. What does the function call memset() do here? (1 mark)

```c
    struct sigaction act;

    memset (&act, '\0', sizeof(act));

```

Listing 4: memset()

Ans: Because the `sigaction` struct was declared as a stack variable, its fields may initially contain garbage values from previous computations. The `memset()` function is used to initialize all fields to `'\0'`, ensuring a clean state before setting specific fields.

9. What does act = {0} do here? (1 mark)

```c
    struct sigaction act;

    act = {0};

```

Listing 5: act

Ans: It does the same thing as memset() previously, but makes it more concise. This syntax only works after version C99.

10. Suppose a SIGTERM signal comes in. What is the output? (1 mark)

```c
    static void hd1 (int sig, siginfo_t *siginfo, void *context)
    {
        printf("SIGTERM receieved.");
    }

    ....

    struct sigaction act;

    memset (&act, '\0', sizeof(act));

    act.sa_sigaction = &hd1;
    act.sa_flags = SA_SIGINFO;

    if (sigaction(SIGTERM, &act, NULL) < 0)
    {
        perror("sigaction");
        return 1;
    }

```

Listing 6: simple example

Ans: SIGTERM recieved.

11. What do the following keywords in C do? (2 marks)

    (a) volatile

    (b) sig_atomic_t

```c
volatile sig_atomic_t signal_val = 0;
```

Listing 7: keywords

Ans:

(a) volatile: tells the compiler that this variable can be changed outside the current flow of code, so don't optimize or cache this value.

(b) sig_atomic_t: ensures reads/writes are atomic (happens in one instruction) and not interrupted by signals

12. What does the raise(int iSig) function do? (1 mark)

Ans: Commands OS to send a signal of type iSig to calling process. Returns 0 to indicate success, non-0 to indicate failure.

```
    iRet = raise(SIGINT);

```

Listing 8: raise() example

raise(SIGINT) sends a 2/SIGINT signal to calling process.

13. What does the kill(pid_t iPid, int iSig) function do? (1 mark)

Ans: Sends a iSig signal to the process iPid. Equivalent to raise(iSig) when iPid is the id of current process. You must own process pid (or have admin privileges).

```
    iRet = kill(1234, SIGINT);
```

Listing 9: kill() example

kill(1234, SIGINT) sends a 2/SIGINT signal to process 1234.

14. What is the output for each of these commands? The code is stored in a executable named "sleep".

    (a) ./sleep 2 (Ctrl + C is not sent)

    (b) ./sleep 5 (Ctrl + C is sent 4 seconds in)

```
    void catch_signal(int sig) {
        got_signal = 1;
    }

    int main(int argc, char *argv[]) {
        if (argc != 2) {
            fprintf(stderr, "Usage: %s <seconds>\n", argv[0]);
            return EXIT_FAILURE;
        }

        int max_snooze_secs = atoi(argv[1]);
        if (max_snooze_secs <= 0) {
            fprintf(stderr,
            "Error: Invalid number of seconds '%s' for max snooze time.\n",
            argv[1]);
            return EXIT_FAILURE;
        }

        struct sigaction action = {0};
```

```c
        action.sa_handler = catch_signal;
        action.sa_flags = SA_RESTART;
        if (sigaction(SIGINT, &action, NULL) == -1) {
            perror("sigaction");
            return EXIT_FAILURE;
        }

        int count = 0;
        while (!got_signal && count < max_snooze_secs) {
            sleep(1);
            count++;
        }
        printf("Slept for %d of the %d seconds allowed.\n",
        count, max_snooze_secs);

        return EXIT_SUCCESS;
    }


```

Listing 10: kill() example

Ans:

(a) Slept for 2 of the 2 seconds allowed.

(b) Slept for 4 of the 5 seconds allowed.

15. Q15. (2 marks)

 (a) What does the alarm(int time) function do?

 (b) What happens if the time argument is set to 0?

Ans:

(a) The alarm function sends the SIGALARM (14) signal, which you can use to catch using the signal function. Below is an example:

```c
static void myHandler(int iSig)
{
    printf("In myHandler with argument %d\n", iSig);
    alarm(2); /* Set another alarm */
}

int main(void)
{
    signal(SIGALARM, myHandler);
    alarm(2); /* Set an alarm */
    printf("Entering an infinite loop\n");
    for (;;)
        ;
    return 0;
}

```

Listing 11: alarm() example

In this code, this would cause an alarm to be set every two seconds, and then the print statement in the signal handler would be printed.

(b) alarm(0) cancels any pending alarm that has not gone off from any previous alarm() calls.