COMS3157: Advanced Programming
Signals

April 19, 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. Do the signal() and sigaction() methods pause the flow of code? (1 mark)

Ans: No. The `signal()` and `sigaction()` functions allow the user to asynchronously handle these signals when the program receives these signals in the future. These functions do not pause the flow of code and wait for the signal to come in, they allow custom handling for when the specified signals are recieved in the future.

3. How would each of these signals be triggered? (4 marks)
   (a) SIGFPE
   (b) SIGINT
   (c) SIGTSTP
   (d) SIGCONT

Ans:

(a) **SIGFPE** is triggered by a floating-point exception, such as division by zero or overflow in arithmetic operations.

(b) **SIGINT** is triggered when the user types `Ctrl+C` in the terminal. It is used to interrupt a running process.

(c) **SIGTSTP** is triggered when the user types `Ctrl+Z` in the terminal. It is used to stop (suspend) a process temporarily.

(d) **SIGCONT** is triggered when a stopped process (e.g., from `SIGTSTP`) is continued, typically by the `fg` or `bg` command.

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 how to handle the signal

6. What do each field for 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 1: sigaction struct

Ans:

(a) void (*sa_handler)(int): function pointer to custom function that handles signal

(b) void (*sa_sigaction)(int, siginfo_t *, void *): function pointer to a more advanced function that handles signal (more advanced to *sa_handler(int))

(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 e.g. SA_SIGINFO tells compiler to use sa_sigaction() function rather than sa_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 2: 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 3: 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 4: simple example

Ans: SIGTERM recieved.

11. What is the output of the code in these two scenarios? (2 marks)
    (a) Sending a SIGINT signal through Ctrl + C `immediately` after the code runs.
    (b) Sending a SIGINT signal after 6 seconds.

```c
void sig_handler_2(int signum) {
    sleep(signum); // sleep for duration equal to signal number
    printf("%d\n", signum);
}

void B(void) {
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));          // Zero out the struct
    sa.sa_handler = sig_handler_2;       // Set handler function
    sigemptyset(&sa.sa_mask);            // Initialize signal mask to
   empty
    sigaddset(&sa.sa_mask, SIGINT);      // Block SIGINT during
   handler

    sigaction(SIGQUIT, &sa, NULL);       // Set handler for SIGQUIT

    kill(getpid(), SIGQUIT);             // Send SIGQUIT to self
    sleep(2);                            // Sleep after sending signal
    printf("%d\n", 200);
}

int main(void) {
    B();
    return 0;
}
```

Listing 5: pending signals

Ans:

(a) 3. The SIGINT signal is blocked by the sigaction() but is executed immediately after the signal handler has completed. This means that the program terminates immediately after the signal handler, meaning that it will not print out 200.

(b) 3 200

12. What is the output of the code? If a process was terminated, write "TERM". (1 mark)

```c
void C(void) {
    struct sigaction sa = {0}; /* memset struct to 0 */
    sa.sa_handler = sig_handler_1;
    pid_t pid = fork();
    if (pid == 0) {
        sigaction(SIGTERM, &sa, NULL);
        sleep(2);
        kill(getppid(), SIGTERM);
        kill(getpid(), SIGKILL);
    }
}

int main(void) {
    C();
    return 0;
}
```

Listing 6: signals and fork

Ans: TERM TERM. The parent process does not have the signal SIGTERM handled, so the first kill function call kills the parent (note: `getppid()`) and the second kill function call kills the child.

13. 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

14. 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.

```c
    iRet = raise(SIGINT);
```

Listing 8: raise() example

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

15. 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).

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

Listing 9: kill() example

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

16. What is the output for each of these commands? The code is stored in a executable named "sleep". (2 marks)

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

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

```c
    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};
        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: sleep() example

Ans:
(a) Slept for 2 of the 2 seconds allowed.
(b) Slept for 4 of the 5 seconds allowed.

17. Answer the following questions about the alarm(int time) function. (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 after `time` seconds, which you can use to catch using the signal function.
(b) alarm(0) cancels any pending alarm that has not gone off from any previous alarm() calls.

18. If SIGALRM is not caught, what is the default behavior of SIGALRM? (1 mark)

Ans: It will terminate the process that called the alarm function.

19. What is the output of this code? (1 mark)

```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

Ans: 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. This continues indefinitely.

20. What is the output for each of these scenarios? (1 mark)

    (a) you enter the number '4' after 4 seconds

    (b) you enter the number '7' after 7 seconds

```c
static void myHandler(int iSig)
{
    printf("\nSorry. You took too long.\n");
    exit(EXIT_RETURN);
}

itn main(void)
{
    int i;
    signal(SIGALRM, myHandler);
    printf("Enter a number: ");
    alarm(5);
    scanf("%d", &i);
    alarm(0);
    printf("You entered the number %d.\n", i);
    return 0;
}
```

Listing 12: time bomb example

Ans:

(a) You entered the number: 4.

(b) Sorry. You took too long.