Семестр 4 (2019), занятие 4 (часть 1). Сигналы

Понятие сигнала

Приведенная ниже программа ожидает от пользователя ввода целого числа. После попытки считывания введенного числа программа завершает свое выполнение с кодом 0. При этом неважно, успешно или нет было прочитано введенное число.

```
int main(void) {
    int x;
    scanf("%d", &c);
    return 0;
}
```

Скомпилируем и запустим эту программу.

```
$ ./prog
```

Не вводя никаких чисел, нажмем комбинацию клавиш Ctrl+C. Программа тут же завершит свое выполнение, при этом вернет код отличный от 0.

```
$ echo $?
130
```

Попытаемся разобраться с тем, что же здесь произошло.

Выполнение пользовательских программ осуществляется под управлением операционной системы. Для этого используется механизм процессов. Для каждой запущенной на выполнение пользователем программы в ядре операционной системы создается специальный объект, называемый *процессом*. В первом приближении понятия «выполняющаяся пользовательская программа» и «процесс» могут быть отождествлены друг с другом. Каждый процесс имеет свой уникальный идентификатор – целое число, называемое *pid*.

Процессы взаимодействуют с операционной системой, а также могут взаимодействовать друг с другом. *Сигналы* представляют собой простейший способ подобного взаимодействия. Процесс или операционная система может послать другому процессу целое число (код сигнала) из предопределенного множества. Для этого необходимо знать pid процесса-адресата и код соответствующего сигнала.

Получить список сигналов можно с помощью программы kill (необходимо вызвать с ключом -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 ...
```

Сигнал SIGKILL сразу уничтожает получивший его процесс. Остальные сигналы процесс может проигнорировать или перехватить и обработать. По умолчанию обработчик сигнала уничтожает процесс. При этом процесс возвращает код равный 128 + <код сигнала>.

В рассотренном примере нажатие комбинации клавиш Ctrl+C приводит к отправке выполняемой программе сигнала SIGINT с кодом 2. Обработчик по умолчанию этого сигнала завершает выполнение программы с кодом 128 + 2.

Заметим, что ранее, мы уже неоднократно сталкивались с сигналами SIGFPE (исключение в операции с плавающей точкой) и SIGSEGV (ошибка сегментирования). Эти сигналы операционная система отправляла выполняющейся программе, когда ее действия приводили к возникновению соответствующего исключения.

Программа kill позволяет отпавить процессу сигнал. Для этого необходимо знать pid процесса-адресата.

```
$ pgrep prog
10457
$ kill -8 10457
```

В приведенном примере сначала с помощью утилиты pgrep был получен список идентификаторов всех процессов, исполняемый файл которых имеет имя prog. Далее, процессу с pid равным 10457 был отправлен сигнал SIGFPE (код 8).

Установка обработчика сигнала

Функция обработчик-сигнала должна иметь следующий вид

```
void handler(int signo) {
    ...
}
```

Она имеет один целочисленный входной параметр (код полученного сигнала) и тип возвращаемого значения void.

Обработчик сигнала устанавливается с помощью системного вызова sigaction. Этот системный вызов имеет три входных параметра: код сигнала, указатель на структуру типа struct sigaction (в поле sa_handler этой структуры задается указатель на функцию обработчика сигнала), число 0. В случае успеха системный вызов возвращает значение 0, а в случае ошибки возвращает значение -1.

Пример

В приведенной ниже программе устанавливаются обработчики для сигналов SIGINT и SIGPIPE (строки 12 – 21). В обоих случаях в качестве обработчика выступает функция handler (строки 7 – 9). Эта функция сохраняет переданный ей код полученного сигнала в глобальную переменную n.

В функции main сразу после установки обработчиков сигналов присутствует цикл (строки 23 – 24). На каждой итерации этого цикла программа «засыпает» на одну секунду. Цикл выполняется пока глобальная переменная n имеет нулевое значение.

Заметим, что при создании эта переменная инициализируется нулевым значением, которое может

быть изменено только в результате вызова функции `handler`. В свою очередь эта функция может быть вызвана, только если программа перехватила сигнал `SIGINT` или `SIGPIPE`.

После выхода из цикла (строка 26) программа печатает значение глобальной переменной `n` (код перехваченного сигнала).

Определение глобальной переменной `n` содержит ключевое слово `volatile`. Оно сообщает компилятору, что во время выполнения программы значение этой переменной может «неожиданно» измениться. Это предостерегает компилятор от выполнения оптимизации программного кода с участием этой переменной. В качестве типа переменной был выбран `sig_atomic_t`. Это гарантирует, что обращения к этой переменной осуществляются «атомарно». На практике этот тип скорее всего является псевдонимом для типа `int`, введенным с помощью `typedef`.

```
1  #include <signal.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  volatile sig_atomic_t n;
6
7  void handler(int signo) {
8      n = signo;
9  }
10
11 int main(void) {
12     struct sigaction a;
13     a.sa_handler = handler;
14     a.sa_flags   = 0;
15     sigemptyset(&a.sa_mask);
16
17     if(sigaction(SIGINT, &a, 0) == -1)
18         return -1;
19
20     if(sigaction(SIGPIPE, &a, 0) == -1)
21         return -1;
22
23     while(!n)
24         sleep(1);
25
26     printf("signo: %d\n", n);
27
28     return 0;
29 }
```