2020-11-30

Занятие #12.

Задача 12.1.

В лекции вам предлагалось написать полный рабочий код программы, псевдокод которой показан в таблице:

Process A	Process B
Do work	Set up signal handler for `SIGUSR1`
Work on `foo()`	`volatile sig_atomic_t gFlag=0;` Do work
`foo()` done; send `SIGUSR1` to process B	
	`signal_handler()` function entered asynchronously
	`gFlag = 1;`
[...]	[...]

Речь шла о том, что следует переключиться с использования глобального буфера на глобальную целочисленную переменную с типом данных **`sig_atomic_t`** и, что важно, отметьте ее как **`volatile`**, чтобы компилятор отключил оптимизацию вокруг нее.

Задача 12.2.

В операционной системе Solaris использовалась пара функций - для отображения номеров сигналов в их имена и обратно:

```
#include <signal.h>
int sig2str(int signo, char *str);
int str2sig(const char *str, int *signop);
```

Обе функции возвращают 0 в случае успеха, -1 — в случае ошибки. Эти функции удобны при разработке интерактивных программ, которые должны принимать и выводить номера сигналов и их имена. Функция **`sig2str`** преобразует номер сигнала в строку и сохраняет результат в памяти по адресу, переданному в аргументе *str*. Вызывающий процесс должен был предоставить буфер достаточного размера для хранения строки максимально возможной длины, с учетом завершающего нулевого символа. Для этих целей Solaris предусматривал в заголовочном файле <signal.h> константу **`SIG2STR_MAX`**, которая представляет максимальный размер строки, возвращаемой функцией **`sig2str`**. Возвращаемая строка содержит имя сигнала без префикса SIG. Например, если функции передать номер сигнала SIGKILL, она вернет строку «KILL» в буфере, на который указывает аргумент *str*. Реализуйте функцию **`sig2str`**.

Задача 12.3.

Напишите программу, с помощью которой можно было бы проверить функции синхронизации родительского и дочернего процессов из следующей программы:

```
static volatile sig_atomic_t sigflag; /* устанавливается обработчиком */
/* в ненулевое значение */
static sigset_t newmask, oldmask, zeromask;
static void sig_usr(int signo) { /* единый обработчик для сигналов SIGUSR1 и SIGUSR2 */
    sigflag = 1;
}

void TELL_WAIT(void) {
    if (signal(SIGUSR1, sig_usr) == SIG_ERR)
        perror("ошибка вызова функции signal(SIGUSR1)");
    if (signal(SIGUSR2, sig_usr) == SIG_ERR)
        perror("ошибка вызова функции signal(SIGUSR2)");
    sigemptyset(&zeromask);
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGUSR1);
    sigaddset(&newmask, SIGUSR2);
    /*
    * Заблокировать сигналы SIGUSR1 и SIGUSR2, и сохранить текущую маску сигналов.
    */
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
        perror("ошибка выполнения операции SIG_BLOCK");
}

void TELL_PARENT(pid_t pid) {
    kill(pid, SIGUSR2); /* сообщить родительскому процессу, что мы готовы */
}

void WAIT_PARENT(void) {
    while (sigflag == 0)
        sigsuspend(&zeromask); /* ждать ответа от родительского процесса */
    sigflag = 0;
    /* Восстановить маску сигналов в начальное состояние.*/
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        perror("ошибка выполнения операции SIG_SETMASK");
}

void TELL_CHILD(pid_t pid) {
    kill(pid, SIGUSR1); /* сообщить дочернему процессу, что мы готовы */
}

void WAIT_CHILD(void) {
    while (sigflag == 0)
        sigsuspend(&zeromask); /* дождаться ответа от дочернего процесса */
    sigflag = 0;
    /* Восстановить маску сигналов в начальное состояние. */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        perror("ошибка выполнения операции SIG_SETMASK");
}
```

Процесс должен создавать файл и записывать в него число 0. Затем вызывать функцию `fork`, после чего родительский и дочерний процессы должны по очереди увеличивать число, прочитанное из файла. При каждом увеличении счетчика процесс должен выводить информацию о том, кто произвел увеличение - родитель или потомок. В этом примере использованы сигналы, определяемые пользователем: сигнал `SIGUSR1` передается от родительского процесса дочернему, а `SIGUSR2` - от дочернего родительскому.