**PIPE, SIGNAL**

int flag = 0;

void handler(int sig\_numb)

{

    flag = 1;

    printf(" Signal handler %d installed.\n", sig\_numb);

}

int main(void)

{

    pid\_t chpids[2];

    int wstatus;

    pid\_t w;

    char \*texts[2] = {"xx", "y-y-y-y-y"};

    int pipefd[2];

    if (pipe(pipefd) == -1) {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

    if (signal(SIGINT, handler) == SIG\_ERR) {

        perror("Ошибка signal");

        exit(EXIT\_FAILURE);

    }

    printf("Push Ctrl+C, to send message from childs.\n");

    sleep(2);

    for (size\_t i = 0; i < 2; ++i) {

        if ((chpids[i] = fork()) == -1) {

            perror("Can not fork.\n");

            exit(EXIT\_FAILURE);

        }

        else if (chpids[i] == 0) {

            if (flag) {

                printf("PID = %d write: '%s'\n", getpid(), texts[i]);

                close(pipefd[0]);

                write(pipefd[1], texts[i], strlen(texts[i]));

            } else

                printf("PID = %d no signal \n", getpid());

            exit(EXIT\_SUCCESS);

        }

    }

    for (size\_t i = 0; i < 2; ++i) {

        w = wait(&wstatus);

        if (w == -1) {

            perror("waitpid");

            exit(EXIT\_FAILURE);

        }

        printf("terminated pid = %d\t", w);

        if (WIFEXITED(wstatus)) {

            printf("exited, status = %d\n", WEXITSTATUS(wstatus));

        } else if (WIFSIGNALED(wstatus)) {

            printf("killed by signal %d\n", WTERMSIG(wstatus));

        }

    }

    char buf[50];

    for (size\_t i = 0; i < 2; ++i) {

        close(pipefd[1]);

        read(pipefd[0], &buf, strlen(texts[i]));

        printf("Received message from PID = %d: %s\n", chpids[i], buf);

        buf[0] = 0;

    }

    close(pipefd[1]);

    read(pipefd[0], buf, 1);

    printf("Received message: %s\n", buf);

    exit(EXIT\_SUCCESS);

}

**DAEMON**

#include "apue.h"

#include <syslog.h>

#include <fcntl.h>

#include <sys/resource.h>

#include <errno.h>

#include <pthread.h>

#define LOCKFILE "/var/run/daemon.pid"

#define CONFFILE "/etc/daemon.conf"

#define LOCKMODE (S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)

// Инициализация процесса-демона

void daemonize(const char \*cmd)

{

    int                 i, fd0, fd1, fd2;

    pid\_t               pid;

    struct rlimit       rl;

    struct sigaction    sa;

    // Сброс маску создания файлов в значение 0

    umask(0);

    // Получить максимально возможный номер дескриптора файла.

    if (getrlimit(RLIMIT\_NOFILE, &rl) == -1)

        err\_quit("%s: невозможно получить максимальный номер дескриптора ", cmd);

    // Стать лидером нового сеанса, чтобы утратить управляющий терминал

    if ((pid = fork()) == -1)

        err\_quit("%s: ошибка вызова функции fork", cmd);

    else if (pid != 0)  // родительский процесс

        exit(0);

    // Создаем новый сеанс

    if (setsid() == -1) {

        perror("Ошибка setsid");

        exit(EXIT\_FAILURE);

    }

    // Обеспечить невозможность обретения управляющего терминала в будущем.

    sa.sa\_handler = SIG\_IGN;

    sigemptyset(&sa.sa\_mask);   // количество различных сигналов может превышать количество бит в целочисленном типе, поэтому в большинстве случаев нельзя использовать тип int

    sa.sa\_flags = 0;

    if (sigaction(SIGHUP, &sa, NULL) == -1)

        err\_quit("%s:  невозможно игнорировать сигнал SIGHUP", cmd);

    // Назначить корневой каталог текущим рабочим каталогом,

    // чтобы впоследствии можно было отмонтировать файловую систему.

    if (chdir("/") == -1)

        err\_quit("%s:  невозможно сделать текущим рабочим каталогом  /", cmd);

    // Закрыть все открытые файловые дескрипторы.

    if (rl.rlim\_max == RLIM\_INFINITY)

        rl.rlim\_max = 1024; // макс число файловых дескрипторов которые могут быть открыты

    for (i = 0; i < rl.rlim\_max; i++)

        close(i);

    // Присоединить файловые дескрипторы 0, 1 и 2 к /dev/null.

    fd0 = open("/dev/null", O\_RDWR);

    fd1 = dup(0);

    fd2 = dup(0);

    // Инициализировать файл журнала.

    openlog(cmd, LOG\_CONS, LOG\_DAEMON);

    if (fd0 != 0 || fd1 != 1 || fd2 != 2){

        syslog(LOG\_ERR, "ошибочные файловые дескрипторы %d %d %d",

               fd0, fd1, fd2);

        exit(1);

    }}

// Установка блокировки для записи на весь файл

int lockfile(int fd)

{

    struct flock fl;

    fl.l\_type = F\_WRLCK;

    fl.l\_start = 0;

    fl.l\_whence = SEEK\_SET;

    fl.l\_len = 0;

    return (fcntl(fd, F\_SETLK, &fl));

}

// Функция, которая гарантирует запуск только одной копии демона

int already\_running(void)

{

    int fd;

    int rc;

    int conffd;

    char buf[16];

    char conf\_buf[256];

    int perms = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH;

    fd = open(LOCKFILE, O\_RDWR | O\_CREAT, perms);

    if (fd == -1) {

        syslog(LOG\_ERR, "невозможно открыть %s: %s", LOCKFILE, strerror(errno));

        exit(1);

    }

    if (lockfile(fd) == -1) {

        if (errno == EACCES || errno == EAGAIN)

        {

            close(fd);

            return 1;

        }

        syslog(LOG\_ERR, "невозможно установить блокировку на %s: %s", LOCKFILE, strerror(errno));

        exit(1);

    }

    ftruncate(fd, 0);

    sprintf(buf, "%ld", (long)getpid());

    write(fd, buf, strlen(buf) + 1);

    if ((conffd = open(CONFFILE, O\_RDWR | O\_CREAT, perms)) == -1) // username userid {

        syslog(LOG\_ERR, "ошибка открытия файла %s", CONFFILE);

        exit(EXIT\_FAILURE);

    }

    ftruncate(conffd, 0);

    sprintf(conf\_buf, "%ld\n%s", (long) getuid(), getlogin());

    rc = write(conffd, conf\_buf, strlen(conf\_buf) + 1);

    if (rc == -1) {

        syslog(LOG\_ERR, "ошибка записис в файл %s", CONFFILE);

        exit(EXIT\_FAILURE);

    }

    close(conffd);

    return 0;

}

void reread(void) {

    FILE \*fd;

    int pid,userid;

    char login[50];

    if ((fd = fopen(CONFFILE, "r")) == NULL) {

        syslog(LOG\_INFO, "Ошибка fopen " CONFFILE "");

        exit(EXIT\_FAILURE);

    }

    fscanf(fd, "%d\n", &userid);

    fscanf(fd, "%s\n", login);

    fclose(fd);

    syslog(LOG\_INFO, "UserName: %s", login);

    syslog(LOG\_INFO, "UserID: %d", userid);

}

void \*thr\_fn(void \*arg) {

    sigset\_t \*mask = arg;

    int err, signo;

    for (;;)

    {

        err = sigwait(mask, &signo);

        if (err != 0)

        {

            syslog(LOG\_ERR, "sigwait failed");

            exit(EXIT\_FAILURE);

        }

        switch (signo){

        case SIGHUP:

            syslog(LOG\_INFO, "Re-reading configuration file");

            reread();

            break;

        case SIGTERM:

            syslog(LOG\_INFO, "got SIGTERM; exiting");

            if (remove(CONFFILE) == -1)

                syslog(LOG\_ERR, "error remove %s", CONFFILE);

            exit(EXIT\_SUCCESS);

        default:

            syslog(LOG\_INFO, "unexpected signal %d\n", signo);

        }}}

int main(int argc, char \*argv[]) {

    int err;

    pthread\_t tid;

    char \*cmd;

    struct sigaction sa;

    sigset\_t mask;

    // cmd - имя исполняемого файла (без пути)

    if ((cmd = strrchr(argv[0], '/')) == NULL)

        cmd = argv[0];

    else

        cmd++;

    // Перейти в режим демона.

    daemonize(cmd);

    // Убедиться, что ранее не была запущена другая копия демона.

    if (already\_running()) {

        syslog(LOG\_ERR, "демон уже запущен");

        exit(EXIT\_FAILURE);

    }

    // до этого момента SIGHUP игнорируется

    // Восстановить действие по умолчанию для сигнала SIGHUP

    sa.sa\_handler = SIG\_DFL;

    sigemptyset(&sa.sa\_mask);

    sa.sa\_flags = 0;

    if (sigaction(SIGHUP, &sa, NULL) == -1)

        err\_quit("%s:  невозможно восстановить действие SIG\_DFL для SIGHUP");

    // и заблокировать все сигналы.

    sigfillset(&mask);

    if ((err = pthread\_sigmask(SIG\_BLOCK, &mask, NULL)) != 0)

        err\_exit(err, "ошибка выполнения операции SIG\_BLOCK");

    // Создать поток для обработки SIGHUP и SIGTERM.

    err = pthread\_create(&tid, NULL, thr\_fn, &mask);

    if (err == -1)

        err\_exit(err, "can't create thread");

    while (1)

    {

        time\_t cur\_time = time(NULL);

        syslog(LOG\_NOTICE, "Time: %s", ctime(&cur\_time));

        sleep(10);

    }

    return 0;

}

**PC**

struct sembuf Pprod[2] = {{BEMPTY, P, SEM\_UNDO}, {BINARY, P, SEM\_UNDO}};

struct sembuf Vprod[2] = {{BINARY, V, SEM\_UNDO}, {BFULL, V, SEM\_UNDO}};

struct sembuf Pcons[2] = {{BFULL, P, SEM\_UNDO}, {BINARY, P, SEM\_UNDO}};

struct sembuf Vcons[2] = {{BINARY, V, SEM\_UNDO}, {BEMPTY, V, SEM\_UNDO}};

producer:

        if (semop(semid, Pprod, 2) == -1) {

            char err\_msg[100];

            sprintf(err\_msg, "ERR: semop PID=%d, errno=%d (EINTR=%d)", getpid(), errno, EINTR);

            errExit(err\_msg);

        }

        printf("Производитель PID=%d положил '%c'\n", getpid(), \*(\*prod\_ptr));

        if (semop(semid, Vprod, 2) == -1)

        errExit("semop");

consumer

        if (semop(semid, Pcons, 2) == -1) {

            char err\_msg[100];

            sprintf(err\_msg, "ERR: semop PID=%d, errno=%d (EINTR=%d)", getpid(), errno, EINTR);

            errExit(err\_msg);

        }

        printf("Потребитель   PID=%d взял    '%c'\n", getpid(), \*(\*cons\_ptr));

        if (semop(semid, Vcons, 2) == -1)

            errExit("semop");

main:

    if (signal(SIGINT, myhandler) == SIG\_ERR)

        errExit("signal");

    key = ftok(argv[0], 1);

    if (key == -1)

        errExit("ftok");

    shmid = shmget(key, N\_BUF + 3, IPC\_CREAT | perms);

    if (shmid == -1)

        errExit("shmget");

    addr = shmat(shmid, NULL, 0);

    if (addr == (void \*) -1)

        errExit("shmat");

    semid = semget(key, 3, IPC\_CREAT | perms);

    if (semid == -1)

        errExit("semget");

    if (semctl(semid, BINARY, SETVAL, 1) == -1)

        errExit("semctl");

    if (semctl(semid, BEMPTY, SETVAL, N\_BUF) == -1)

        errExit("semctl");

    if (semctl(semid, BFULL, SETVAL, 0) == -1)

        errExit("semctl");

**WR**

struct sembuf start\_read[] = {

    {readers\_queue, 1, 0},

    {active\_writer, 0, 0},

    {writers\_queue, 0, 0},

    {numb\_of\_readers, 1, 0},

    {readers\_queue, -1, 0}

};

struct sembuf stop\_read[] = { {numb\_of\_readers, -1, 0} };

struct sembuf start\_write[] = {

    {writers\_queue, 1, 0},

    {numb\_of\_readers, 0, 0},

    {bin\_sem, -1, 0},

    {active\_writer, 1, 0},

    {writers\_queue, -1, 0}

};

struct sembuf stop\_write[] = { {active\_writer, -1, 0}, {bin\_sem, 1, 0} };

READER

        if (semop(semid, start\_read, 5) == -1) {

            exit(EXIT\_FAILURE);

        }

        printf("Читатель PID=%d считал '%c'\n", getpid(), \*buf);

        if (semop(semid, stop\_read, 1) == -1) {

            exit(EXIT\_FAILURE);

        }

WRITER

        if (semop(semid, start\_write, 5) == -1) {

            exit(EXIT\_FAILURE);

        }

        printf("Писатель PID=%d записал '%c'\n", getpid(), \*buf);

        if (semop(semid, stop\_write, 2) == -1){

            exit(EXIT\_FAILURE);

        }

**WINDOWS**

void start\_read() {

    InterlockedIncrement(&readers\_queue);

    if (active\_writer || writers\_queue)

        WaitForSingleObject(can\_read, INFINITE);

    WaitForSingleObject(mutex, INFINITE);

    InterlockedDecrement(&readers\_queue);

    InterlockedIncrement(&active\_readers);

    SetEvent(can\_read);

    ReleaseMutex(mutex);

}

void stop\_read() {

    InterlockedDecrement(&active\_readers);

    if (active\_readers == 0)

        SetEvent(can\_write);

}

void start\_write() {

    InterlockedIncrement(&writers\_queue);

    if (active\_readers || active\_writer)

        WaitForSingleObject(can\_write, INFINITE);

    InterlockedDecrement(&writers\_queue);

    InterlockedIncrement(&active\_writer);

}

void stop\_write() {

    InterlockedDecrement(&active\_writer);

    ResetEvent(can\_write);

    if (readers\_queue > 0)

        SetEvent(can\_read);

    else

        SetEvent(can\_write);

}