Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук

Лабораторна робота №11

з навчальної дисципліни

«Операційні системи»

Виконав:
студент групи КУ-31
Нечипоренко Д.І.

Харків – 2022

Завдання

Задание №1

Напишите программу, в которой определите пользовательскую реакцию процесса на сигналы SIGINT, SIGTERM (вывод информации о захвате соответствующего сигнала) и SIGUSR1 (завершение работы программы), восстановите поведение по умолчанию для сигнала SIGPROF и игнорируйте сигнал SIGHUP. Для задержки выполнения процесса используйте функцию pause() в бесконечном цикле. Убедитесь в работоспособности программы (с помощью команды kill оболочки отошлите процессу нужный сигнал и проследите за реакцией). Реализуйте две версии программы: с помощью функции signal и с помощью функции sigaction.

Задание №2

Напишите программу «*будильник*»: программа через командную строку получает требуемый интервал времени (в секундах, для удобства отладки и демонстрации) и текстовое сообщение; завершает основной процесс, а в дочернем с помощью функций alarm и pause через заданное время выводит в стандартный поток вывода заданный текст.

Задание №3

Напишите программу, в которой создаются два процесса (родительский и дочерний). Эти процессы должны поочередно, синхронизировано выводить сообщения в стандартный поток вывода. Процессы синхронизируют свою работу, посылая друг другу сигнал SIGUSR1 при помощи вызова kill.

Задание №4

Напишите программу, в которой создаются два процесса (родительский и дочерний). Родительский процесс заданное количество раз посылает дочернему процессу сигнал SIGUSR1 (можно взять один из сигналов реального времени) с дополнительной информацией (целое число - номер вызова). Дочерний процесс обрабатывает сигнал и в стандартный поток вывода выводит полученный номер, текстовое представление сигнала и дополнительную информацию. Затем основной процесс посылает дочернему сигнал SIGTERM, завершающие его работу, ожидает завершение дочернего процесса и завершает работу сам.

Код:

Task 1

signal

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>


// kill -s SIGUSR1 3422
```

```
/* Handler for SIGINT */
void signal_handler(int signalNumber) {
    printf("%d\n", signalNumber);
    if (signalNumber == SIGINT) {
        printf("Catch signal SIGINT\n");
    } else if (signalNumber == SIGTERM) {
        printf("Catch signal SIGTERM\n");
    } else if (signalNumber == SIGUSR1){
        printf("Catch signal SIGUSR1\n");
        exit(EXIT_SUCCESS);
    } else {
        fprintf(stderr, "Wrong Signal\n");
        exit(EXIT_FAILURE);
    }

    // exit(EXIT_SUCCESS);
}

int main() {
    setbuf(stdout, NULL);
    printf("PID: %d\n", (int)getpid());

    // Register signal_handler as handler for SIGINT
    if (signal(SIGINT, signal_handler) == SIG_ERR) {
        fprintf(stderr, "It is impossible to handle SIGINT!\n");
        exit(EXIT_FAILURE);
```

```
}

if (signal(SIGTERM, signal_handler) == SIG_ERR) {
    fprintf(stderr, "It is impossible to handle SIGTERM!\n");
    exit(EXIT_FAILURE);
}

if (signal(SIGPROF, SIG_DFL) == SIG_ERR) {
    fprintf(stderr, "It is impossible to reset SIGPROF!\n");
    exit(EXIT_FAILURE);
}

if (signal(SIGHUP, SIG_IGN) == SIG_ERR) {
    fprintf(stderr, "It is impossible to ignore SIGHUP!\n");
    exit(EXIT_FAILURE);
}

if (signal(SIGUSR1, signal_handler) == SIG_ERR) {
    fprintf(stderr, "It is impossible to handle SIGUSR1!\n");
    exit(EXIT_FAILURE);
}

while(1) {
    printf("The process is waiting for signal\n");
    pause();
}
```

```
    fprintf(stderr, "The program is finished\n");


    return EXIT_SUCCESS;
}
```

Sigastion

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>


static void signal_handler(int signalNumber) {
    printf("Signal No %d (%s) has been received.\n", signalNumber,
strsignal(signalNumber));
    if (signalNumber == SIGUSR1) {
        exit(EXIT_SUCCESS);
    }
}


int main() {
    static struct sigaction act;


    setbuf(stdout, NULL);
    printf("PID: %d\n", (int)getpid());


    act.sa_handler = signal_handler;
```

```
if (sigaction(SIGINT, &act, NULL) == -1) {
    fprintf(stderr, "It is impossible to handle SIGINT!\n");
    exit(EXIT_FAILURE);
}

if (sigaction(SIGTERM, &act, NULL) == -1) {
    fprintf(stderr, "It is impossible to handle SIGTERM!\n");
    exit(EXIT_FAILURE);
}

if (sigaction(SIGUSR1, &act, NULL) == -1) {
    fprintf(stderr, "It is impossible to handle SIGUSR1!\n");
    exit(EXIT_FAILURE);
}

act.sa_handler = SIG_DFL;
if (sigaction(SIGPROF, &act, NULL) == -1) {
    fprintf(stderr, "It is impossible to reset SIGPROF!\n");
    exit(EXIT_FAILURE);
}

act.sa_handler = SIG_IGN;
if (sigaction(SIGHUP, &act, NULL) == -1) {
    fprintf(stderr, "It is impossible to ignore SIGHUP!\n");
    exit(EXIT_FAILURE);
}
```

```
    while(1) {
        printf("The process is waiting for signal\n");
        pause();
    }

    fprintf(stderr, "The program is finished\n");

    return EXIT_SUCCESS;
}
```

Task 2

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>

#define TRUE 1
#define FALSE 0
#define BELLS "\007\007\007"  /* Sound signal ASCII */

sig_atomic_t alarm_flag = FALSE;

/* Signal SIGALRM handler */
void setFlag(int sig) {
    alarm_flag = TRUE;
```

```
}

int main(int argc, char **argv) {
    int nsecs, j;
    pid_t pid;
    static struct sigaction act;

    if (argc < 2) {
        fprintf(stderr, "Wrong strating command.\nMust be: Task2 seconds_number message\n");
        exit(EXIT_FAILURE);
    }

    if ((nsecs = atoi(argv[1])) <= 0) {
        fprintf(stderr, "Task2: invalid time\n");
        exit(EXIT_FAILURE);
    }

    switch(pid = fork()) {
        case -1: /* Error */
            perror("Task2");
            exit(EXIT_FAILURE);
        case 0: /* Child process */
            break;
        default: /* Parent process */
            printf("Chile process Task2 with id %d is working\n", pid);
            printf("Wait a signal after %d second\n", nsecs);
```

```
            exit(EXIT_SUCCESS);
        }

        /* Set timer handler */
        act.sa_handler = setFlag;
        sigaction(SIGALRM, &act, NULL);

        /* Set timer */
        alarm(nsecs);

        /* Stop execution antil signal is catched */
        pause();

        /* If signal SIGALRM was catched put message */
        if (alarm_flag == TRUE) {
            printf(BELLS);
            for (j = 2; j < argc; j++) {
                printf("%s ", argv[j]);
            }
            printf("\n");
        }

        return EXIT_SUCCESS;
}
```

Task 3

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void handler(int signo);

int main() {
    pid_t pid;
    static struct sigaction act;
    sigset_t set1, set2;

    sigfillset(&set1);
    sigdelset(&set1, SIGUSR1);
    sigdelset(&set1, SIGINT); // To Stop By Ctrl+C
    sigfillset(&set2);
    sigdelset(&set2, SIGINT); // To Stop By Ctrl+C
    sigprocmask(SIG_SETMASK, &set2, NULL);
    act.sa_handler = handler;
    sigfillset(&act.sa_mask);
    sigdelset(&act.sa_mask, SIGUSR1);
    act.sa_flags = SA_RESTART;
    sigaction(SIGUSR1, &act, NULL);
    pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Fork ERROR!\n");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { /*Child*/
```

```
        fprintf(stderr, "Child pid %d\n", (int) getpid());
        while (1) {
            kill(getppid(), SIGUSR1);
            sigsuspend(&set1);
        }
    } else { /*parent*/
        fprintf(stderr, "Parent pid %d\n", (int) getpid());
        while (1) {
            sigsuspend(&set1);
            kill(pid, SIGUSR1);
        }
    }

    return EXIT_SUCCESS;
}

void handler(int signo) {
    fprintf(stderr, "The pid: %d. ", (int) getpid());
    psignal(signo, "The received");
}
```

Task 4

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
```

```
int main(void) {
    int i, status;
    pid_t pid;
    void my_handler(int signo, siginfo_t *si, void *ucontext);

    static struct sigaction act;
    sigval_t value;

    switch (pid = fork()) {
        case -1: /* Error */
            perror("task4");
            exit(EXIT_FAILURE);

        case 0: /* Child Process */
            /* Define handler of signal SIGUSR1 in parent process */
            sigfillset(&act.sa_mask);
            act.sa_flags = SA_SIGINFO;
            act.sa_sigaction = my_handler;
            sigaction(SIGUSR1, &act, NULL);
            while(1) {
                pause();
            }
            break;
        default: /* Parent Process */
            /* Infinite loop */
            for (i = 1; i <= 10; i++) {
                sleep(1);
```

```
            value.sival_int = 404 + i;
            if (sigqueue(pid, SIGUSR1, value) != 0) {
                perror("synchro");
                exit(EXIT_FAILURE);
            }
        }
        sleep(2);
        kill(pid, SIGTERM);
        waitpid(pid, &status, 0);
    }

    return EXIT_SUCCESS;
}

void my_handler(int signo, siginfo_t *si, void *ucontext) {
    extern const char *const sys_siglist[];
    printf("Signal %d (%s). Data from parent process: %d\n", si->si_signo,
sys_siglist[si->si_signo], si->si_value.sival_int);
}
```

Результат:

Task 1

```
gcc sigaction-function.c -o sigaction-function.out
gcc signal-function.c -o signal-function.out
./signal-function.out
PID: 21378
The process is waiting for signal
SIGTERM
```

Task 2

```
make
gcc main.c -o main.out
./main.out 5
Chile process Task2 with id 21452 is working
Wait a signal after 5 second
nadia@nadia-VirtualBox:~/Documents/labs/lab8/2$
```

Task 3

```
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
The pid: 21467. The received: User defined signal 1
The pid: 21468. The received: User defined signal 1
```

Task 4

```
Signal 10 (User defined signal 1). Data from parent process: 405
Signal 10 (User defined signal 1). Data from parent process: 406
Signal 10 (User defined signal 1). Data from parent process: 407
Signal 10 (User defined signal 1). Data from parent process: 408
Signal 10 (User defined signal 1). Data from parent process: 409
Signal 10 (User defined signal 1). Data from parent process: 410
Signal 10 (User defined signal 1). Data from parent process: 411
Signal 10 (User defined signal 1). Data from parent process: 412
Signal 10 (User defined signal 1). Data from parent process: 413
```