МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: IPC Linux

Студентка гр. 2384 _____ Соц Е.А.

Преподаватель _____ Душутина Е.В.

Санкт-Петербург
2024

Цель работы.

Целью данной работы является изучение средств межпроцессного взаимодействия в ОС семейства Unix

Задание.

1. Используя функцию sigaction(), продемонстрируйте возможности управления линейкой сигналов, включая собственные обработчики и маскирование для разных сигналов, а также вариативность, предоставляемую структурами sigaction (act / oldact).

2. Реализуйте надежные сигналы, организуйте эксперимент для доказательства отложенной обработки (например, в случае поступления сигнала во время уже выполняющейся обработки другого сигнала). Сгенерируйте ситуацию с несколькими отложенными сигналами.

3. Экспериментально продемонстрируйте разницу между надежными и ненадежным сигналами.

4. Организуйте «вложенные» надежные сигналы, для этого из обработчика одного сигнала необходимо произвести отправку другого сигнала, а также отправку такого же сигнала.

5. Проведите эксперимент, доказывающий возможность организации очереди для различных типов сигналов, обычных и сигналов реального времени; Проверьте возможность организации очереди для «вложенных» сигналов РВ.

6. Опытным путем подтвердите наличие приоритетов сигналов реального времени.

7. Экспериментально подтвердите, что обработка равно-приоритетных сигналов реального времени происходит в порядке FIFO.

Выполнение работы.

Информация о системе:

Linux katya 6.5.0-28-generic #29~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Apr 4 14:39:20 UTC 2 x86_64 x86_64 x86_64 GNU/Linux

Задание 1. Используя функцию sigaction(), продемонстрируйте возможности управления линейкой сигналов, включая собственные обработчики и маскирование для разных сигналов, а также вариативность, предоставляемую структурами sigaction (act / oldact).

Сигналы позволяют осуществить самый примитивный способ коммуникации между двумя процессами. Функция sigaction() позволяет модифицировать действие, связанное с сигналом, и используется для управления обработкой сигналов в программе.

Структура sigaction определяет действие, связанное с сигналом. Она включает в себя функцию обработчика сигнала (sa_handler), набор сигналов, которые должны быть добавлены в маску сигналов (sa_mask), и флаги сигнала (sa_flags), которые контролируют поведение обработчика сигнала.

```
(base) katya@katya:~/os/lb5$ cat task1.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

// переменная для изначального обработчика
struct sigaction old_hand;

// 2ой обработчик сигнала
void handler2(int sig){
    printf("SIGRTMIN во 2ом обработчике\n");

    // восстановление старого обработчика
```

```
        sigaction(SIGRTMIN, &old_hand, NULL);
        sleep(10);
        printf("2ой обработчик завершился после 10 сек\n");
    }

    // 1ый обработчик сигнала
    void handler1(int sig){
        printf("SIGRTMIN во 1ом обработчике\n");

        // надежная обработка сигналов
        struct sigaction sa;
        sa.sa_handler = handler2;
        sigemptyset(&sa.sa_mask);
        sigaddset(&sa.sa_mask, SIGINT);

        // установка втрого обработчика
        sigaction(SIGRTMIN, &sa, &old_hand);
        sleep(10);
        printf("1ый обработчик завершился после 10 сек\n");
    }

    int main(){
        // надежная обработка сигналов
        struct sigaction act;
        act.sa_handler = handler1;
        // пустая маска
        sigemptyset(&act.sa_mask);

          // добавление маски в сигнал, блокирующийся во время
обработки других
        sigaddset(&act.sa_mask, SIGINT);

        // установка первого обработчика для SIGRTMIN
        sigaction(SIGRTMIN, &act, NULL);

        // ждем поступления сигналов
        for(;;) pause();

        return 0;
    }
    (base) katya@katya:~/os/lb5$ gcc task1.c -o task1

    (base) katya@katya:~/os/lb5$ ./task1 &
    [1] 4260
    (base) katya@katya:~/os/lb5$ kill -SIGRTMIN %1
    SIGRTMIN во 1ом обработчике
    (base) katya@katya:~/os/lb5$ jobs -l
    [1]+  4260 Запущен          ./task1 &
    (base) katya@katya:~/os/lb5$ 1ый обработчик завершился после
10 сек
    SIGRTMIN во 2ом обработчике
    (base) katya@katya:~/os/lb5$ kill -SIGRTMIN %1
    (base) katya@katya:~/os/lb5$ 2ой обработчик завершился после
10 сек
    SIGRTMIN во 1ом обработчике
```

```
(base) katya@katya:~/os/lb5$ 1ый обработчик завершился после
10 сек

(base) katya@katya:~/os/lb5$ kill -SIGRTMIN %1
SIGRTMIN во 2ом обработчике
(base) katya@katya:~/os/lb5$ kill -SIGINT %1
(base) katya@katya:~/os/lb5$ jobs -l
[1]+  4260 Запущен          ./task1 &
(base) katya@katya:~/os/lb5$ 2ой обработчик завершился после
10 сек

[1]+  Прерывание    ./task1
```

Разработанная программа отражает, как сигналы могут быть использованы для управления потоком выполнения программы, включая переключение между различными обработчиками сигналов и управления жизненным циклом программы.

Также видно, что при использовании самого системного вызова можно в такую же структуру запомнить текущую диспозицию для ее восстановления в последующем: первый обработчик заменился вторым, но была запомнена начальная диспозиция, которая и была восстановлена.

Задание 2. Реализуйте надежные сигналы, организуйте эксперимент для доказательства отложенной обработки (например, в случае поступления сигнала во время уже выполняющейся обработки другого сигнала). Сгенерируйте ситуацию с несколькими отложенными сигналами.

Надежность означает, что если при возникновении сигнала система занята обработкой другого сигнала (назовем его «текущим»), то возникший сигнал не будет потерян, а его обработка будет отложена до окончания текущего обработчика.

```
(base) katya@katya:~/os/lb5$ cat task2.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
```

```
void (*mysig(int sig, void (*hnd)(int)))(int) {
    // надежная обработка сигналов
    struct sigaction act;
    act.sa_handler = hnd;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, SIGRTMIN);
    sigaddset(&act.sa_mask, SIGRTMAX);
    sigaddset(&act.sa_mask, SIGINT);
    act.sa_flags = 0;
    sigaction(sig, &act, 0);
    return act.sa_handler;
}


// обработчик сигнала
void handler1(int sig){
    printf("SIGUSR1 обработчик\n");
    sleep(30);
    printf("SIGUSR1 обработчик завершился после 30 сек\n");
}

// обработчик сигнала
void handler2(int sig){
    printf("SIGRTMIN обработчик\n");
    sleep(30);
    printf("SIGRTMIN обработчик завершился после 30 сек\n");
}

// обработчик сигнала
void handler3(int sig){
    printf("SIGRTMAX обработчик\n");
    sleep(30);
    printf("SIGRTMAX обработчик завершился после 30 сек\n");
}


int main(){
    mysig(SIGUSR1, handler1);
    mysig(SIGRTMIN, handler2);
    mysig(SIGRTMAX, handler3);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
}
(base) katya@katya:~/os/lb5$ gcc task2.c -o task2
(base) katya@katya:~/os/lb5$ ./task2 &
[1] 13944
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик

(base) katya@katya:~/os/lb5$ kill -SIGRTMIN %1
(base) katya@katya:~/os/lb5$ kill -SIGRTMAX %1
```

```
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик завершился
после 30 сек
SIGRTMIN обработчик

(base) katya@katya:~/os/lb5$ kill -SIGINT %1
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 13944 Запущен          ./task2 &
(base) katya@katya:~/os/lb5$ SIGRTMIN обработчик завершился
после 30 сек

[1]+  Прерывание    ./task2
(base) katya@katya:~/os/lb5$ ./task2 &
[1] 14956
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик

(base) katya@katya:~/os/lb5$ kill -SIGRTMIN %1
(base) katya@katya:~/os/lb5$ kill -SIGRTMAX %1
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик завершился
после 30 сек
SIGRTMIN обработчик

(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 14956 Запущен          ./task2 &
(base) katya@katya:~/os/lb5$ SIGRTMIN обработчик завершился
после 30 сек
SIGRTMAX обработчик

(base) katya@katya:~/os/lb5$ kill -SIGINT %1
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 14956 Запущен          ./task2 &
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 14956 Запущен          ./task2 &
(base) katya@katya:~/os/lb5$ SIGRTMAX обработчик завершился
после 30 сек

[1]+  Прерывание    ./task2
```

Созданная программа трем сигналам устанавливает одинаковую маску и обработчики, работающие по 30 секунд. После запуска программы видно, как при отправке сигнала во время работы другого обработчика, сигналы ждали завершения работы текущего обработчика и только потом начинали “свою работу”. Следует отметить, что если отправить несколько сигналов подряд, сначала будет исполняться сигнал с наименьшим номером, что было показано в первом тестировании программы (SIGINT завершил программу до исполнения SIGRTMAX).

Задание 3. Экспериментально продемонстрируйте разницу между надежными и ненадежным сигналами.

Программа, реализующая надежную передачу сигнала с помощью функции sigaction:

```
(base) katya@katya:~/os/lb5$ cat task3.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

void (*mysig(int sig, void (*hnd)(int)))(int) {
    // надежная обработка сигналов
    struct sigaction act;
    act.sa_handler = hnd;
    sigemptyset(&act.sa_mask);

    sigaddset(&act.sa_mask, SIGINT);
    act.sa_flags = 0;
    sigaction(sig, &act, NULL);
    return act.sa_handler;
}


// обработчик сигнала
void handler1(int sig){
    printf("SIGUSR1 обработчик\n");
    sleep(30);
    printf("SIGUSR1 обработчик завершился после 30 сек\n");
}


int main(){
    //надежный обработчик
    mysig(SIGUSR1, handler1);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
}
(base) katya@katya:~/os/lb5$ gcc task3.c -o task3
(base) katya@katya:~/os/lb5$ ./task3 &
[1] 15380
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
SIGUSR1 обработчик
(base) katya@katya:~/os/lb5$ kill -SIGINT %1
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 15380 Запущен          ./task3 &
```

```
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик завершился после 30 сек

[1]+  Прерывание    ./task3
```

В результате выполнения программы обработчик полностью закончил свою работу, даже после отправления SIGINT.

Программа, реализующая ненадежную передачу с помощью функции signal:

```
(base) katya@katya:~/os/lb5$ cat task3.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

void (*mysig(int sig, void (*hnd)(int)))(int) {
    // надежная обработка сигналов
    struct sigaction act;
    act.sa_handler = hnd;
    sigemptyset(&act.sa_mask);

    sigaddset(&act.sa_mask, SIGINT);
    act.sa_flags = 0;
    sigaction(sig, &act, NULL);
    return act.sa_handler;
}


// обработчик сигнала
void handler1(int sig){
    printf("SIGUSR1 обработчик\n");
    sleep(30);
    printf("SIGUSR1 обработчик завершился после 30 сек\n");
}


int main(){
    //надежный обработчик
    //mysig(SIGUSR1, handler1);

    //ненадежный
    signal(SIGUSR1, handler1);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
```

```
}
(base) katya@katya:~/os/lb5$ gcc task3.c -o task3
(base) katya@katya:~/os/lb5$ ./task3 &
[1] 15624
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
SIGUSR1 обработчик
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 15624 Запущен          ./task3 &
(base) katya@katya:~/os/lb5$ kill -SIGINT %1
(base) katya@katya:~/os/lb5$ jobs -l
[1]+ 15624 Прерывание    ./task3
```

Видно, что после отправки SIGINT программа сразу же завершила свою работу, не дожидаясь завершения текущего обработчика.

Таким образом, наглядно видна разница между надежнымми и ненадежными сигналами.

Задание 4. Организуйте «вложенные» надежные сигналы, для этого из обработчика одного сигнала необходимо произвести отправку другого сигнала, а также отправку такого же сигнала.

Была разработана программа, в которой задавались обработчики сигналов для SIGUSR1 и SIGFPE. При SIGUSR1 максировался SIGFPE.

```
(base) katya@katya:~/os/lb5$ cat task4.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

void (*mysig(int sig, void (*hnd)(int)))(int) {
    // надежная обработка сигналов
    struct sigaction act;
    act.sa_handler = hnd;
    sigemptyset(&act.sa_mask);

    //добавление в маску сигнала, который должен быть отложен
    if(sig == SIGUSR1)
        sigaddset(&act.sa_mask, SIGFPE);

    act.sa_flags = 0;
    sigaction(sig, &act, NULL);
```

```
    return act.sa_handler;
}


// обработчик сигнала
void handler1(int sig){
    printf("SIGUSR1 обработчик\n");
    // отпрвка другого сигнала
    kill(getpid(), SIGFPE);
    // отправка такого же сигнала
    kill(getpid(), SIGUSR1);
    sleep(20);
    printf("SIGUSR1 обработчик завершился после 20 сек\n");
}

// обработчик сигнала
void handler2(int sig){
    printf("SIGFPE обработчик\n");
    sleep(20);
    printf("SIGFPE обработчик завершился после 20 сек\n");
}


int main(){
    //надежный обработчик
    mysig(SIGUSR1, handler1);
    mysig(SIGFPE, handler2);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
}
(base) katya@katya:~/os/lb5$ gcc task4.c -o task4
(base) katya@katya:~/os/lb5$ ./task4 &
[1] 7825
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
SIGUSR1 обработчик
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик завершился
после 20 сек
SIGUSR1 обработчик
SIGUSR1 обработчик завершился после 20 сек
SIGUSR1 обработчик

(base) katya@katya:~/os/lb5$ kill %1
```

При отправке обеих сигналов в SIGUSR1 можно увидеть, что SIGFPE никогда не обрабатывался, несмотря на то, что у него меньший номер. Таким образом можно сказать, что повышенный приоритет имеет тот сигнал, из чьего обработчика отправляется текущий сигнал.

Проведем эксперимент с отправкой только SIGFPE:

```
(base) katya@katya:~/os/lb5$ cat task4.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

void (*mysig(int sig, void (*hnd)(int)))(int) {
    // надежная обработка сигналов
    struct sigaction act;
    act.sa_handler = hnd;
    sigemptyset(&act.sa_mask);

    //добавление в маску сигнала, который должен быть отложен
    if(sig == SIGUSR1)
        sigaddset(&act.sa_mask, SIGFPE);

    act.sa_flags = 0;
    sigaction(sig, &act, NULL);
    return act.sa_handler;
}


// обработчик сигнала
void handler1(int sig){
    printf("SIGUSR1 обработчик\n");
    // отпрвка другого сигнала
    kill(getpid(), SIGFPE);
    // отправка такого же сигнала
    //kill(getpid(), SIGUSR1);
    sleep(20);
    printf("SIGUSR1 обработчик завершился после 20 сек\n");
}

// обработчик сигнала
void handler2(int sig){
    printf("SIGFPE обработчик\n");
    sleep(20);
    printf("SIGFPE обработчик завершился после 20 сек\n");
}


int main(){
    //надежный обработчик
    mysig(SIGUSR1, handler1);
    mysig(SIGFPE, handler2);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
}
(base) katya@katya:~/os/lb5$ gcc task4.c -o task4
```

```
(base) katya@katya:~/os/lb5$ ./task4 &
[1] 20746
(base) katya@katya:~/os/lb5$ kill -SIGUSR1 %1
(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик

(base) katya@katya:~/os/lb5$ SIGUSR1 обработчик завершился
после 20 сек
SIGFPE обработчик

(base) katya@katya:~/os/lb5$ SIGFPE обработчик завершился
после 20 сек

(base) katya@katya:~/os/lb5$ kill %1
```

Теперь внутри SIGUSR1 посылается SIGFPE, поэтому результатом является выполнение обработчика SIGFPE после завершения SIGUSR1.

Оба примера сохраняют надежность, так как обработка нового сигнала осуществлялась только после завершения обработки текущего.

Задание 5-7. Проведите эксперимент, доказывающий возможность организации очереди для различных типов сигналов, обычных и сигналов реального времени; Проверьте возможность организации очереди для «вложенных» сигналов РВ.

Опытным путем подтвердите наличие приоритетов сигналов реального времени.

Экспериментально подтвердите, что обработка равно-приоритетных сигналов реального времени происходит в порядке FIFO.

Если мы хотим, чтобы сигналы обрабатывались как сигналы реального времени, мы должны:

1) использовать сигналы с номерами в диапазоне от SIGRTMIN до SIGRTMAX

2) должны указать флаг SA_SIGINFO при вызове sigaction()с установкой обработчика сигнала

3) обработчик сигнала реального времени, устанавливаемый с флагом SA_SIGINFO, объявляется как:void func(intsigno, siginfo_t*info, void*context);

где signo—номер сигнала,

siginfo_t — структура, определяемая как

typedef struct {

int si_signo; /* тоже, чтои signo */

int si_code; /* SI_{USER,QUEUE,TIMER,ASYNCIO,MESGQ} */

union sigval si_value; /* целое или указатель от отправителя */}

siginfo_t;

на что указывает context — зависит от реализации.

Таким образом, сигналы реального времени несут больше информации, чем прочие сигналы (при отправке сигнала, не обрабатываемого как сигнал реального времени, единственным аргументом обработчика является номер сигнала).

Создадим эксперимент для сигналов SIGRTMIN, SIGRTMIN+1, SIGRTMIN+2, SIGUSR1, SIGUSR2:

```
(base) katya@katya:~/os/lb5$ cat task5.c
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>


// обработчик сигнала
void handler1(int sig, siginfo_t *info, void *context){
    if (sig == SIGRTMIN){
        printf("SIGRTMIN: %d\n", info->si_value.sival_int);

        //вложенная отправка
        if(info->si_value.sival_int == 0){
            union sigval val;
            val.sival_int = 3;
            sigqueue(getpid(), SIGRTMIN, val);
        }
        sleep(1);
    }
```

```
    else if(sig == SIGRTMIN+1){
        printf("SIGRTMIN+1: %d\n", info->si_value.sival_int);
        sleep(1);
    }
    else if(sig == SIGRTMIN+2){
        printf("SIGRTMIN+2: %d\n", info->si_value.sival_int);
        sleep(1);
    }
    else if(sig == SIGUSR1){
        printf("SIGUSR1: %d\n", info->si_value.sival_int);
        sleep(1);
    }
    else if(sig == SIGUSR2){
        printf("SIGUSR2: %d\n", info->si_value.sival_int);
        sleep(1);
    }
}

// обработчик сигнала
void handler2(int sig, siginfo_t *info, void *context){
    union sigval val;
    for(int i = 0; i<3; ++i){
        val.sival_int = i;
        sigqueue(getpid(), SIGRTMIN, val);
    }
    for(int i = 0; i<3; ++i){
        val.sival_int = i;
        sigqueue(getpid(), SIGRTMIN+1, val);
    }
    for(int i = 0; i<3; ++i){
        val.sival_int = i;
        sigqueue(getpid(), SIGRTMIN+2, val);
    }
    for(int i = 0; i<3; ++i){
        val.sival_int = i;
        sigqueue(getpid(), SIGUSR1, val);
    }
    for(int i = 0; i<3; ++i){
        val.sival_int = i;
        sigqueue(getpid(), SIGUSR2, val);
    }
    sleep(2);
}

int main(){
    // сигналы обрабатываются как сигналы реального времени
    struct sigaction sa;
    sa.sa_sigaction = handler1;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_SIGINFO;
    sigaddset(&sa.sa_mask, SIGRTMIN);
    sigaddset(&sa.sa_mask, SIGRTMIN+1);
    sigaddset(&sa.sa_mask, SIGRTMIN+2);
    sigaddset(&sa.sa_mask, SIGUSR1);
    sigaddset(&sa.sa_mask, SIGUSR2);
```

```
    sigaction(SIGRTMIN, &sa, NULL);
    sigaction(SIGRTMIN+1, &sa, NULL);
    sigaction(SIGRTMIN+2, &sa, NULL);
    sigaction(SIGUSR1, &sa, NULL);
    sigaction(SIGUSR2, &sa, NULL);

    //сигнал, который создаст очередь
    struct sigaction act;
    act.sa_sigaction = handler2;
    sigemptyset(&act.sa_mask);

    sigaddset(&act.sa_mask, SIGRTMIN);
    sigaddset(&act.sa_mask, SIGRTMIN+1);
    sigaddset(&act.sa_mask, SIGRTMIN+2);
    sigaddset(&act.sa_mask, SIGUSR1);
    sigaddset(&act.sa_mask, SIGUSR2);

    act.sa_flags = SA_SIGINFO;

    sigaction(SIGFPE, &act, NULL);
    union sigval val;
    sigqueue(getpid(), SIGFPE, val);

    // ждем поступления сигналов
    for(;;) pause();

    return 0;
}
(base) katya@katya:~/os/lb5$ gcc task5.c -o task5
(base) katya@katya:~/os/lb5$ ./task5
SIGUSR1: 0
SIGUSR2: 0
SIGRTMIN: 0
SIGRTMIN: 1
SIGRTMIN: 2
SIGRTMIN: 3
SIGRTMIN+1: 0
SIGRTMIN+1: 1
SIGRTMIN+1: 2
SIGRTMIN+2: 0
SIGRTMIN+2: 1
SIGRTMIN+2: 2
```

В обработчике handler1 для сигналов реального времени выводится имя сигнала и значение, которое с ним пришло. В независимом обработчике handler2 создавалась очередь из трех сигналов каждого типа, при отправке передавалось число от 0 до 2 (чем меньше число, тем раньше сигнал был отправлен).

Как можно заметить из результата работы программы, даже при инициализации как сигналы реального времени сигналы SIGUSR1 и

SIGUSR2 ведут себя “обычно”: для каждого типа был обработан только первый.

Для сигналов SIGRTMIN+i, i=0,1,2 более раннее отправленные приходили раньше, чем позднее отправленные при одном приоритете, меньшие по значению сигналу обрабатывались раньше, что полностью сходится с теорией, описанной выше.

Так как была произведена вложенная отправка, SIGRTMIN можно заметить со значением 3. Видно, что эта вложенная отправка хорошо вошла в общую очередь, не мешая остальным сигналам очереди.

Список литературы

1. https://elib.spbstu.ru/dl/2/s17-72.pdf/en/view - Методическое пособие «Межпроцессные взаимодействия в операционных системах», Душутина Е.В
2. https://elib.spbstu.ru/dl/2/s17-71.pdf/view - Методическое пособие «Практические вопросы разработки системных приложений», Душутина Е.В
3. https://studfile.net/preview/4631958/page:38/ - Сигналы