

**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: Взаимодействие родственных процессов. Управление процессами посредством сигналов. Многонитевое функционирование.**

Студентка гр. 1304

Чернякова В.А.

Преподаватель

Душутина Е.В.

Санкт-Петербург

2023

## Цель работы.

Изучить системное программирование в ОС семейства UNIX.

## Выполнение работы.

Модель ОС:

```
Linux Valeriya 4.15.0-142-generic #146~16.04.1-Ubuntu SMP Tue  
Apr 13 09:27:15 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

## Взаимодействие родственных процессов

**1. Изменяя длительности выполнения процессов и параметры системных вызовов, рассмотрите 3 ситуации и получите соответствующие таблицы процессов: процесс-отец запускает процесс-сын и ожидает его завершения; процесс-отец запускает процесс сын и завершает своё выполнение; процесс-отец запускает процесс-сын и не ожидает его завершения.**

Родственными считаются процессы, ближайшие в дереве процессов, т.е. породивший и порожденные им процессы. Их взаимодействие основывается на наследовании. Оно существенно проще по сравнению с взаимодействием независимых процессов, поскольку независимые процессы полностью изолированы друг от друга и нуждаются в посреднике при обмене информацией в виде ядра ОС. Ядро предоставляет им специальные механизмы доступа и синхронизации (IPC) и управляет адресным пространством.

Для упрощения анализа результатов изменения таблицы процессов будем использовать системную функцию `system()`, а в качестве ее аргументов формировать командную строку с вызовом утилиты статуса с нужными ключами и фильтрацией вывода, а также перенаправлением этого вывода не только на терминал, но и в файл.

Создадим программы, где будем перенаправлять результаты в файл.

*father.c*

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

```

#include <sys/types.h>
#include <wait.h>
#include <string.h>
int main(int argc, char *argv[])
{
    int sid, pid, pid1, ppid, status;
    char command[50];
    if (argc < 2)
        return -1;
    pid = getpid();
    ppid = getppid();
    sid = getsid(pid);
    sprintf(command, "ps xjf | grep \"STAT\\||%d\\\" > %s", sid, argv[1]);
    printf("FATHER PARAMS: sid = %i pid=%i ppid=%i \n", sid, pid, ppid);
    if ((pid1 = fork()) == 0)
        execl("son1", "son1", NULL);
    if (fork() == 0)
        execl("son2", "son2", argv[1], NULL);
    if (fork() == 0)
        execl("son3", "son3", NULL);
    system(command);
    waitpid(pid1, &status, WNOHANG); // эта строка исключается в п.б) и в)
}

```

*son1.c*

```

#include <stdio.h>
void main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("SON_1 PARAMS: pid=%i ppid=%i\nFather creates and waits \n", pid, ppid);
    sleep(3);
}

```

*son2.c (father завершает свое выполнение раньше son2)*

```

#include <stdio.h>
void main(int argc, char *argv[])
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    char command[50];
    sprintf(command, "ps xjf | grep son2 >> %s", argv[1]);
    printf("SON_2 PARAMS: pid=%i ppid=%i\nFather finished before son termination without waiting for it \n", pid, ppid);
}

```

```

    sleep(20);
    ppid=getppid();
    printf("SON_2 PARAMS ARE CHANGED: pid=%i ppid=%i\n",pid,ppid);
    system(command);
}

```

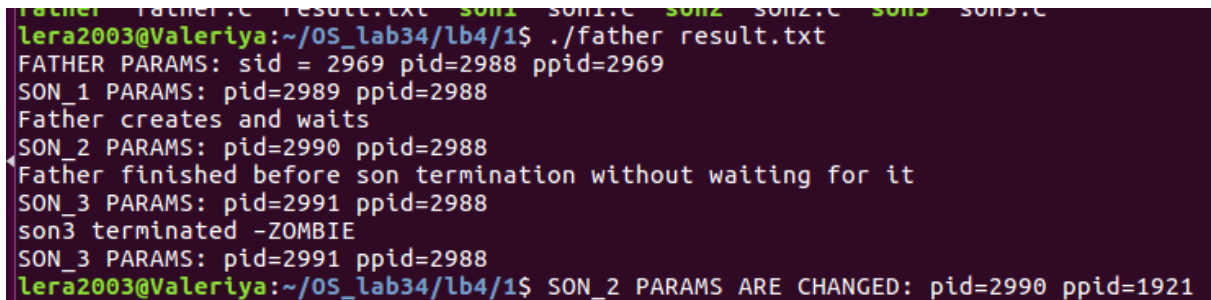
*son3.c (father не ожидает его завершения; son3 завершается)*

```

#include <stdio.h>
void main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("SON_3 PARAMS: pid=%i ppid=%i\n",pid,ppid);
    printf("son3 terminated - ZOMBIE\n",pid,ppid);
    ppid=getppid();
    printf("SON_3 PARAMS: pid=%i ppid=%i\n",pid,ppid);
}

```

Согласно коду, результаты выполнения всех трех ситуаций отображаются на консоли, а в итоговый файл, который передается в качестве параметра father.c, записываются результаты выполнения (ps\_xjf) во время выполнения программ.



```

father.c father.c result.txt son1.c son1.c son2.c son2.c son3.c son3.c
lera2003@Valeriya:~/OS_lab34/lb4/1$ ./father result.txt
FATHER PARAMS: sid = 2969 pid=2988 ppid=2969
SON_1 PARAMS: pid=2989 ppid=2988
Father creates and waits
SON_2 PARAMS: pid=2990 ppid=2988
Father finished before son termination without waiting for it
SON_3 PARAMS: pid=2991 ppid=2988
son3 terminated -ZOMBIE
SON_3 PARAMS: pid=2991 ppid=2988
lera2003@Valeriya:~/OS_lab34/lb4/1$ SON_2 PARAMS ARE CHANGED: pid=2990 ppid=1921

```

В коде son2 для увеличения длительности существования потомка используется задержка в 20 сек, в результате более раннего завершения родителя потомок становится наследником init, PID которого равен единице, т.е. son2 становится «самостоятельным» процессом с PPID=1, что и фиксируется в выходных данных в результате исполнения.

Как видно из результатов, как только процесс-отец завершается, на консоли сразу появляется приглашение на ввод команды. А son2 продолжает свое выполнение в фоновом режиме. Т.к. Время выполнения

son2 много дольше, то результат выполнения процесса- потомка, появится уже после приглашения.

```
lera2003@ValerTya:~/OS_lab34/lb4/1$ cat result.txt
PPID  PID  PGID  SID  TTY      TPGID  STAT  UID    TIME COMMAND
2963  2969  2969  2969  pts/2    3688  Ss     1000   0:00  \_  bash
2969  3688  3688  2969  pts/2    3688  S+     1000   0:00  \_  ./father result.txt
3688  3689  3688  2969  pts/2    3688  S+     1000   0:00  \_  son1
3688  3690  3688  2969  pts/2    3688  S+     1000   0:00  \_  son2 result.txt
3688  3691  3688  2969  pts/2    3688  Z+     1000   0:00  \_  [son3] <defunct>
3688  3692  3688  2969  pts/2    3688  S+     1000   0:00  \_  sh -c ps xjf | grep "STAT\|2969" > result.txt
3692  3693  3688  2969  pts/2    3688  R+     1000   0:00  \_  ps xjf
3692  3694  3688  2969  pts/2    3688  S+     1000   0:00  \_  grep STAT\|2969
1921  3690  3688  2969  pts/2    2969  S      1000   0:00  \_  son2 result.txt
3690  3695  3688  2969  pts/2    2969  S      1000   0:00  \_  sh -c ps xjf | grep son2 >> result.txt
3695  3697  3688  2969  pts/2    2969  S      1000   0:00  \_  grep son2
lera2003@ValerTya:~/OS_lab34/lb4/1$ ps -xjf
PPID  PID  PGID  SID  TTY      TPGID  STAT  UID    TIME COMMAND
1781  1921  1921  1921  ?        -1     Ss     1000   0:00  /sbin/upstart --user
```

В файле отображаются выполняемые процессы, условное дерево порождения процессов, их атрибуты и состояния.

В выводе зафиксированы: нормальное выполнение потомка son1.

Смена родителя son2 и его переход в самостоятельную ветку. В коде son2 выполняется задержка в 20 секунд, что обеспечивает то, что он будет работать дольше процесса-родителя. В результате видно, что во PPID son2 во время работы программы равен 3688, а через 20 секунд изменились на 1921.

А также состояние zombie для son3 (ситуация, когда потомок выполняется быстрее процесса-отца, при этом отец не дожидается и никак не фиксирует завершение потомка). Рассмотрим, что произошло с процессом son3. Поскольку процесс-отец не дождался завершения дочернего процесса, он находится в состоянии Zombie (STAT = Z+). Это означает, что процесс остается формально существующим, но ресурсы, отведенные для него освобождены. Такие процессы остаются в таблице на случай, если кто-то запросит статистику использования ресурсов этим потомком или статус о его завершении.

## Управление процессами посредством сигналов

**2. С помощью команды kill -l ознакомьтесь с перечнем сигналов, поддерживаемых процессами.**

Для передачи сигналов процессам в Linux используется утилита kill. Ее синтаксис очень прост:

```
$ kill -сигнал pid_процесса
```

Системный вызов `kill()` позволяет отправить сигнал процессу (или группе процессов). При успешном выполнении (т. е. хотя-бы один сигнал отправлен) возвращает 0, при ошибке – -1.

Утилита `kill` позволяет задавать сигнал как числом, так и символьным значением. Базовый перечень сигналов, поддерживаемый практически в любой POSIX-ориентированной ОС, составляет не более тридцати двух (количество бит в тридцати двухразрядном слове), и в большинстве современных систем их номера смещены к началу нумерации. Наряду с базовыми в POSIX ОС дополнительно может поддерживаться свой уникальный набор сигналов.

Ознакомиться с полным перечнем сигналов можно с помощью команды `kill -l`.

```
lera2003@Valeriya:~/OS_lab34/lb4/2$ 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
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Следует заметить, что именованное базовых сигналов, как правило, совпадает в разных Unix-подобных ОС, чего нельзя сказать о нумерации, поэтому целесообразно сначала ознакомиться со списком.

Рассмотрим некоторые из сигналов базового списка:

1) `SIGHUP` предназначен для того, чтобы информировать программу о потере связи с управляющим терминалом, так же и в том случае, если процесс-лидер сессии завершил свою работу. Многие программы-демоны, у которых нет лидера сессии, так же обрабатывают этот сигнал. В ответ на получение `SIGHUP` демон обычно перезапускается. По умолчанию программа, получившая этот сигнал, завершается.

2)SIGINT посылается процессу, если пользователь с консоли отправил команду прервать процесс комбинацией клавиш (Ctrl+C).

6)SIGABRT посылается программе в результате вызова функции abort (3). В результате программа завершается с сохранением на диске образа памяти.

9)SIGKILL завершает работу программы. Программа не может ни обработать, ни игнорировать этот сигнал.

11)SIGSEGV посылается процессу, который пытается обратиться к не принадлежащей ему области памяти. Если обработчик сигнала не установлен, программа завершается с сохранением на диске образа памяти.

15)SIGTERM вызывает «вежливое» завершение программы. Получив этот сигнал, программа может выполнить необходимые перед завершением операции (например, высвободить занятые ресурсы). Получение SIGTERM свидетельствует не об ошибке в программе, а о желании ОС или пользователя завершить ее.

17)SIGCHLD посылается процессу в том случае, если его дочерний процесс завершился или был приостановлен. Родительский процесс также получит этот сигнал, если он установил режим отслеживания сигналов дочернего процесса и дочерний процесс получил какой-либо сигнал. По умолчанию сигнал SIGCHLD игнорируется.

18)SIGCONT возобновляет выполнение процесса, остановленного сигналом SIGSTOP.

19)SIGSTOP приостанавливает выполнение процесса. Как и SIGKILL, этот сигнал невозможно перехватить или игнорировать.

20)SIGTSTP приостанавливает процесс по команде пользователя (Ctrl+Z).

29)SIGIO сообщает процессу, что на одном из дескрипторов, открытых асинхронно, появились данные. По умолчанию этот сигнал завершает работу программы.

10) и 12) SIGUSR1 и SIGUSR2 предназначены для прикладных задач и передачи ими произвольной информации.

Сигнал может быть отправлен процессу либо ядром, либо другим процессом с помощью системного вызова kill():

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

Аргумент pid адресует процесс, которому посылается сигнал. Аргумент sig определяет тип отправляемого сигнала. С помощью системного вызова kill() процесс может послать сигнал, как самому себе, так и другому процессу или группе процессов. В этом случае процесс, посылающий сигнал, должен иметь те же реальный и эффективный идентификаторы, что и процесс, которому сигнал отправляется.

Процесс может выбрать одно из трех возможных действий при получении сигнала:

- 1.игнорировать сигнал,
- 2.перехватить и самостоятельно обработать сигнал,
- 3.позволить действие по умолчанию.

Текущее действие при получении сигнала называется диспозицией сигнала.

Порожденный вызовом fork() процесс наследует диспозицию сигналов от своего родителя. Однако при вызове exec() диспозиция всех перехватываемых сигналов будет установлена ядром на действие по умолчанию. Далее это будет представлено экспериментально.

В ОС поддерживается ряд функций, позволяющих управлять диспозицией сигналов.

Наиболее простой в использовании является функция signal(). Она позволяет устанавливать и изменять диспозицию сигнала.

```
#include <signal.h>
void (*signal (int sig, void (*disp) (int))) (int);
```



Системный вызов `signal(signum, handler)` позволяет установить свой обработчик сигнала: устанавливает `signum` в обработчик `handler`, который может быть нами написанной функцией (обработчиком сигнала).

Напишем программу `father.c`, порождающий `son1`, `son2`, `son3`. Сделаем так, что `son1` реагирует на сигнал по умолчанию, `son2` – игнорирует сигнал, `son3` – перехватывает и обрабатывает сигнал.

*father.c*

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

int defaultExample(){
    int wstatus;
    pid_t c_pid = fork();
    if (c_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (c_pid == 0) {
        printf("printed from child proc\n");
        execl("son1", "son1", NULL);
    } else {
        printf("printed from parent process - %d\n", getpid());
        int ret;
        sleep(1);
        system("echo >> file.txt");
        system("echo After 1st child: >> file.txt");
        system("ps -s >> file.txt");
        ret = kill(c_pid, SIGTERM);
        if (ret == -1) {
            perror("kill");
            exit(EXIT_FAILURE);
        }
        if (waitpid(c_pid, &wstatus, WUNTRACED | WCONTINUED) == -1) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }
    }
    else{
        puts("son1 exited sucessfully\n");
    }
}
```

```

    return 0;
}

int ignoreSignalExample(){
    int wstatus;
    pid_t c_pid = fork();
    if (c_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (c_pid == 0) {
        printf("printed from child proc\n");
        execl("son2", "son2", NULL);
    } else {
        printf("printed from parent process - %d\n", getpid());
        int ret;
        sleep(1);
        system("echo >> file.txt");
        system("echo After 2nd child: >> file.txt");
        system("ps -s >> file.txt");
        ret = kill(c_pid, SIGTERM);
        if (ret == -1) {
            perror("kill");
            exit(EXIT_FAILURE);
        }
        if (waitpid(c_pid, &wstatus, WUNTRACED | WCONTINUED) == -1) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }
        else{
            puts("son2 exited sucessfully\n");
        }
    }
    return 0;
}

int interceptSignalExample(){
    int wstatus;
    pid_t c_pid = fork();
    if (c_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (c_pid == 0) {
        printf("printed from child proc\n");
        execl("son3", "son3", NULL);
    }

```

```

    } else {
        printf("printed from parent process - %d\n", getpid());
        int ret;
        sleep(1);
        system("echo >> file.txt");
        system("echo After 3rd child: >> file.txt");
        system("ps -s >> file.txt");
        ret = kill(c_pid, SIGTERM);
        if (ret == -1) {
            perror("kill");
            exit(EXIT_FAILURE);
        }
        if (waitpid(c_pid, &wstatus, WUNTRACED | WCONTINUED) == -1) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }
        else{
            puts("son3 exited sucessfully\n");
        }
    }
    return 0;
}

int main(int argc, char *argv[]) {
    system("echo Before: >> file.txt");
    system("ps -s >> file.txt");
    return defaultExample() + ignoreSignalExample() +
    interceptSignalExample();
}

```

*son1.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>
int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("SON_1 PARAMS: pid=%i ppid=%i\n", pid, ppid);
    // while (1);
    sleep(3);
    return 0;
}

```

*son2.c*

```

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

int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("SON_2 PARAMS: pid=%i ppid=%i\n", pid, ppid);
    for(int signum = 1; signum <=31 ; signum++){
        signal(signum, SIG_IGN);
    }
    sleep(10);
    return 0;
}

```

*son3.c*

```

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

void int_handler()
{
    puts("signal intercepted! some handling");
    signal(SIGTERM, SIG_DFL); //восстановление диспозиции по умолчанию
}

int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("SON_3 PARAMS: pid=%i ppid=%i\n", pid, ppid);
    signal(SIGTERM, int_handler);
    sleep(3);
    return 0;
}

```

```

lera2003@Valeriya:~/OS_lab34/lb4/2$ ./father
printed from parent process - 2829
printed from child proc
SON_1 PARAMS: pid=2833 ppid=2829
son1 exited sucessfully

printed from parent process - 2829
printed from child proc
SON_2 PARAMS: pid=2838 ppid=2829
son2 exited sucessfully

printed from parent process - 2829
printed from child proc
SON_3 PARAMS: pid=2843 ppid=2829
signal intercepted! some handling
son3 exited sucessfully

```

Каждой программе подаётся сигнал SIGTERM. Первая программа обрабатывает его стандартно и завершает свою работу. Вторая программа игнорирует его (в результате чего sleep(10) выполняется все десять секунд), третья – перехватывает, выводя в консоль строку, говорящую о том, что сигнал обработан. Рассмотрим таблицу процессов до и после посылки сигналов.

Before:										
UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND	
1000	2278	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/2	0:00	bash	
1000	2418	0000000000000000	0000000000010000	0000000000000004	0000000000010002	S+	pts/2	0:00	/bin/bash ./write.sh	
1000	2420	0000000000000000	0000000000140006	0000000000000000	0000000180000000	S+	pts/2	0:00	script -f session.log	
1000	2421	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/18	0:00	bash -i	
1000	2829	0000000000000000	0000000000010000	0000000000000006	0000000000000000	S+	pts/18	0:00	./father	
1000	2831	0000000000000000	0000000000000000	0000000000000000	0000000000010002	S+	pts/18	0:00	sh -c ps -s >> file.txt	
1000	2832	0000000000000000	0000000000000000	0000000000000000	00000001f3d1fef9	R+	pts/18	0:00	ps -s	
After 1st child:										
UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND	
1000	2278	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/2	0:00	bash	
1000	2418	0000000000000000	0000000000010000	0000000000000004	0000000000010002	S+	pts/2	0:00	/bin/bash ./write.sh	
1000	2420	0000000000000000	0000000000140006	0000000000000000	0000000180000000	S+	pts/2	0:00	script -f session.log	
1000	2421	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/18	0:00	bash -i	
1000	2829	0000000000000000	0000000000010000	0000000000000006	0000000000000000	S+	pts/18	0:00	./father	
1000	2833	0000000000000000	0000000000000000	0000000000000000	0000000000000000	S+	pts/18	0:00	son1	
1000	2836	0000000000000000	0000000000000000	0000000000000000	0000000000010002	S+	pts/18	0:00	sh -c ps -s >> file.txt	
1000	2837	0000000000000000	0000000000000000	0000000000000000	00000001f3d1fef9	R+	pts/18	0:00	ps -s	
After 2nd child:										
UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND	
1000	2278	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/2	0:00	bash	
1000	2418	0000000000000000	0000000000010000	0000000000000004	0000000000010002	S+	pts/2	0:00	/bin/bash ./write.sh	
1000	2420	0000000000000000	0000000000140006	0000000000000000	0000000180000000	S+	pts/2	0:00	script -f session.log	
1000	2421	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/18	0:00	bash -i	
1000	2829	0000000000000000	0000000000010000	0000000000000006	0000000000000000	S+	pts/18	0:00	./father	
1000	2838	0000000000000000	0000000000000000	000000007ffbfeff	0000000000000000	S+	pts/18	0:00	son2	
1000	2841	0000000000000000	0000000000000000	0000000000000000	0000000000010002	S+	pts/18	0:00	sh -c ps -s >> file.txt	
1000	2842	0000000000000000	0000000000000000	0000000000000000	00000001f3d1fef9	R+	pts/18	0:00	ps -s	
After 3rd child:										
UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND	
1000	2278	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/2	0:00	bash	
1000	2418	0000000000000000	0000000000010000	0000000000000004	0000000000010002	S+	pts/2	0:00	/bin/bash ./write.sh	
1000	2420	0000000000000000	0000000000140006	0000000000000000	0000000180000000	S+	pts/2	0:00	script -f session.log	
1000	2421	0000000000000000	0000000000010000	0000000000380004	000000004b817efb	Ss	pts/18	0:00	bash -i	
1000	2829	0000000000000000	0000000000010000	0000000000000006	0000000000000000	S+	pts/18	0:00	./father	
1000	2843	0000000000000000	0000000000000000	0000000000000000	0000000000004000	S+	pts/18	0:00	son3	
1000	2846	0000000000000000	0000000000000000	0000000000000000	0000000000010002	S+	pts/18	0:00	sh -c ps -s >> file.txt	
1000	2847	0000000000000000	0000000000000000	0000000000000000	00000001f3d1fef9	R+	pts/18	0:00	ps -s	

Из вывода `ps -s` видно, что после вызова 1-го дочернего процесса, никакие сигналы не были пойманы/игнорированы/заблокированы, что и ожидалось увидеть, поскольку 1-я программа обрабатывает сигнал.

После вызова второй программы, видно, что напротив строки с `son2` везде нули, кроме столбца `IGNORED`, таким образом убеждаемся, что наш сигнал был проигнорирован `son2`.

После запуска 3-го дочернего процесса, видим, что столбец с `CAUGHT`, как и ожидалось, ненулевой (поскольку третья подпрограмма перехватывает процесс и обрабатывает его).

Проанализируем наследование диспозиции сигналов при создании процессов на этапе `fork()` и `exec()`. Напишем программу `sig_father.c`, которая меняет диспозицию сигналов. Она задаёт обработчик многопользовательских сигналов `SIGUSR1` и `SIGUSR2`. Родительский процесс порождает процесс-копию с помощью `fork()` и уходит в ожидание сигналов. Процесс-потомок, при этом, не ждёт никаких сигналов, не назначает им обработчиков. Обработчик `SIGUSR1` и `SIGUSR2` содержит восстановление диспозиции и оповещение на экране об удачно или неудачно полученном сигнале и идентификаторе родительского процесса. Сигналы генерируются из командной строки. Рассмотрим работу программы

*sig\_father.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
static void handler(int sig)
{
    printf("Caught signal %s\n", sig == SIGUSR1 ? "SIGUSR1" : "SIGUSR2");
    printf("Parent = %d\n", getppid());
    signal(sig, SIG_DFL);
}
int main()
{
    printf("Father's params: pid = %d, ppid = %d\n", getpid(), getppid());
```

```

    signal(SIGUSR1, handler);
    signal(SIGUSR2, handler);
    if (fork() == 0)
        printf("Son's params: pid = %d, ppid = %d\n", getpid(),getppid());
    wait(NULL);
    while (1)
        pause();
    return 0;
}

```

*sig\_son.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
int main()
{
    printf("Son's params: pid = %d, ppid = %d\n", getpid(), getppid());
    while (1)
        pause();
    return 0;
}

```

```

lera2003@Valeriya:~/OS_lab34/lb4/2$ ./sig_father1 &
[10] 2984
lera2003@Valeriya:~/OS_lab34/lb4/2$ Father's params: pid = 2984, ppid = 2421
Son's params: pid = 2985, ppid = 2984
^C
lera2003@Valeriya:~/OS_lab34/lb4/2$ kill -SIGUSR1 2984
Caught signal SIGUSR1
Parent = 2421
lera2003@Valeriya:~/OS_lab34/lb4/2$ kill -SIGUSR1 2985
lera2003@Valeriya:~/OS_lab34/lb4/2$ Caught signal SIGUSR1
Parent = 2984

```

Как видно, при отправке сигнала разным процессам результат совпадает, что говорит о том, что потомок использует один и тот же обработчик, что и родитель. Это свидетельствует о наследовании диспозиции при порождении потомка на этапе `fork()`. Диспозиция сигналов для дочернего процесса, созданного с помощью `fork()` сохраняется даже после завершения процесса-родителя.

Повторим эксперимент для процесса-родителя, порождающего дочерний процесс с помощью `fork()` и `exec()`. Теперь потомок загружается с помощью `exec1()`.

*sig\_father2.c*

```

#include <stdio.h>

```

```

#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
static void handler(int sig)
{
    printf("Caught signal %s\n", sig == SIGUSR1 ? "SIGUSR1" : "SIGUSR2");
    printf("Parent = %d\n", getppid());
    signal(sig, SIG_DFL);
}
int main()
{
    printf("Father's params: pid = %d, ppid = %d\n", getpid(), getppid());
    signal(SIGUSR1, handler);
    signal(SIGUSR2, handler);
    if (fork() == 0)
        execl("sig_son", "sig_son", NULL);
    wait(NULL);
    while (1)
        pause();
    return 0;
}

```

```

lera2003@Valeriya:~/OS_lab34/lb4/2$ ./sig_father2 &
[11] 3008
lera2003@Valeriya:~/OS_lab34/lb4/2$ Father's params: pid = 3008, ppid = 2421
Son's params: pid = 3009, ppid = 3008
^C
lera2003@Valeriya:~/OS_lab34/lb4/2$ kill -SIGUSR1 3008
Caught signal SIGUSR1
Parent = 2421
lera2003@Valeriya:~/OS_lab34/lb4/2$ kill -SIGUSR1 3009
lera2003@Valeriya:~/OS_lab34/lb4/2$

```

При отправке сигнала процессу-отцу срабатывает его обработчик, а при отправке процессу-потомку диспозиция этого сигнала не сохраняется и срабатывает обработчик по умолчанию (ничего не происходит). Из этого можно сделать вывод, что при создании процесса с помощью `fork()` и `exec()` диспозиция сигналов не наследуется.

### 3. Запустите в фоновом режиме несколько утилит.

Запустим в фоновом режиме несколько утилит:

```

lera2003@Valeriya:~/OS_lab34/lb4/3$ sleep 15 & sleep 20 & sleep 25 & sleep 30 &
[1] 3031
[2] 3032
[3] 3033
[4] 3034

```



С помощью утилиты `jobs -l` можно проанализировать порядок выполнения поставленных задач:

```
lera2003@Valeriya:~/OS_lab34/lb4/3$ jobs -l
[1] 3031 Готово sleep 15
[2] 3032 Выполняется sleep 20 &
[3]- 3033 Выполняется sleep 25 &
[4]+ 3034 Выполняется sleep 30 &
```

Выполнение задач начинается с начала. С помощью утилиты `fg` повысим приоритет задачи 23. В результате сразу начинается выполняться задача 23, не в фоновом режиме.

```
lera2003@Valeriya:~/OS_lab34/lb4/3$ sleep 15 & sleep 20 & sleep 25 & sleep 30 &
[21] 3067
[22] 3068
[23] 3069
[24] 3070

lera2003@Valeriya:~/OS_lab34/lb4/3$ fg %23
sleep 25
```

С помощью команды `kill` отменим одно из невыполненных заданий

```
lera2003@Valeriya:~/OS_lab34/lb4/3$ sleep 15 & sleep 20 & sleep 25 & sleep 30 &
[29] 3087
[30] 3088
[31] 3089
[32] 3090
[26] Готово sleep 20
[27] Готово sleep 25
lera2003@Valeriya:~/OS_lab34/lb4/3$ jobs -l
[28] 3084 Готово sleep 30
[29] 3087 Выполняется sleep 15 &
[30] 3088 Выполняется sleep 20 &
[31]- 3089 Выполняется sleep 25 &
[32]+ 3090 Выполняется sleep 30 &
lera2003@Valeriya:~/OS_lab34/lb4/3$ kill 3089
lera2003@Valeriya:~/OS_lab34/lb4/3$ jobs -l
[29] 3087 Готово sleep 15
[30] 3088 Выполняется sleep 20 &
[31]- 3089 Завершено sleep 25
[32]+ 3090 Выполняется sleep 30 &
```

#### 4. Ознакомьтесь с выполнением команды и системного вызова `nice()` и `getpriority()`.

`nice1)` — утилита, запускающая программу с измененным приоритетом. Если не указано ни одного аргумента, команда выводит текущий унаследованный приоритет. В противном случае, `nice` запускает команду с указанным приоритетом. Если смещение не указано, то приоритет команды увеличивается на 10. команда `nice` может смещать

приоритет в диапазоне от -20 до 19 включительно, когда используются права суперпользователя. Когда команда выполняется обычным пользователем, диапазон изменяется от 0 до 19.

Кроме упоминаемой ранее функции `nice()`, часто используются функции:

```
#include <sys/time.h> #include <sys/resource.h>
int getpriority(int which, int who);
int setpriority(int which, int who, int prio);
```

Функциями `getpriority()` и `setpriority()` можно получить и установить приоритет для процесса, группы, и пользователя, в зависимости от заданных значений `which` и `who`:

Which: `PRIO_PROCESS`, `PRIO_USER`, `PRIO_PGRP`,

Who: идентификатор PID, Prio: приоритет.

*nice.c*

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

int main()
{
    printf("Приоритет процесса %d = %d\n", getpid(),
getpriority(PRIO_PROCESS, getpid()));
    printf("Попытка изменения приоритета на %d\n", 30);
    nice(30);
    printf("Новый приоритет = %d\n", getpriority(PRIO_PROCESS, getpid()));

    printf("Попытка уменьшения приоритета на %d\n", 20);
    nice(-20);
    printf("Новый приоритет = %d\n", getpriority(PRIO_PROCESS, getpid()));

    printf("Попытка уменьшить приоритет на %d\n", 150);
    nice(-150);
    printf("Новый приоритет = %d\n", getpriority(PRIO_PROCESS, getpid()));

    return 0;
}
```

```

lera2003@Valeriya:~/05_lab34/lb4/4$ ./nice
Приоритет процесса 3203 = 0
Попытка изменения приоритета на 30
Новый приоритет = 19
Попытка уменьшения приоритета на 20
Новый приоритет = 19
Попытка уменьшить приоритет на 150
Новый приоритет = 19
lera2003@Valeriya:~/05_lab34/lb4/4$ sudo ./nice
Приоритет процесса 3207 = 0
Попытка изменения приоритета на 30
Новый приоритет = 19
Попытка уменьшения приоритета на 20
Новый приоритет = -1
Попытка уменьшить приоритет на 150
Новый приоритет = -20

```

Как видно из рисунка выше, уменьшать приоритет можно только имея права суперпользователя. Для увеличения приоритета такие права не нужны. Также видно, что изменять приоритет можно в рамках от -20 до 19.

Разница в приоритетах для системных и пользовательских процессов есть: обычно большинство системных процессов, отвечающих за управление системы имеют более высокий приоритет, чем пользовательские. Это обусловлено тем, что системные процессы обеспечивают более низкоуровневые функции.

```

lera2003@Valeriya:~/05_lab34/lb4/4$ ps -e -o uid,pid,ppid,pri,ni,cmd

```

UID	PID	PPID	PRI	NI	CMD
0	1	0	19	0	/sbin/init splash
0	2	0	19	0	[kthreadd]
0	4	2	39	-20	[kworker/0:0H]
0	6	2	39	-20	[mm_percpu_wq]
0	7	2	19	0	[ksoftirqd/0]
0	8	2	19	0	[rcu_sched]
0	9	2	19	0	[rcu_bh]
0	10	2	139	-	[migration/0]
0	11	2	139	-	[watchdog/0]
0	12	2	19	0	[cpuhp/0]
0	13	2	19	0	[cpuhp/1]
0	14	2	139	-	[watchdog/1]
0	15	2	139	-	[migration/1]
0	16	2	19	0	[ksoftirqd/1]
0	18	2	39	-20	[kworker/1:0H]
0	19	2	19	0	[cpuhp/2]
0	20	2	139	-	[watchdog/2]
0	21	2	139	-	[migration/2]
0	22	2	19	0	[ksoftirqd/2]
0	24	2	39	-20	[kworker/2:0H]

Приоритеты реального времени используются для процессов, которые должны быть выполнены в строго определенные моменты времени, чтобы обеспечить отзывчивость системы на события внешней среды.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2170	rtkit	RT	1	179M	2580	2304	S	0.0	0.3	0:00.00	/usr/lib/rtkit/rtkit-daemon

При запуске из оболочки пользовательский приоритет по умолчанию равен 20. Он может быть изменен используя, например, утилиту `nice`.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3134	lera2003	20	0	27800	4152	3200	R	0.7	0.4	0:00.90	htop

## 5. Ознакомьтесь с командой `nohup(1)`.

`nohup(1)` — утилита, позволяющая запустить команду, невосприимчивую к сигналам потери связи (`hangup`), и чей вывод будет направлен не на терминал, а в файл `nohup.out`. Таким образом, команда будет выполняться в фоновом режиме даже тогда, когда пользователь выйдет из системы. Запустим длительный процесс с помощью `nohup()`:

*nohup1.c*

```
#include <stdio.h>
void main()
{
    int i;
    for(i = 0; i < 999999999999; i++);
}
```

```
lera2003@Valeriya:~/OS_lab34/lb4/5$ nohup ./nohup1 &
[1] 3276
```

```
3257 ?      I      0:00 [kworker/0:0]
3276 pts/11  R      0:42 ./nohup1
3360 ?      S      0:00 systemctl daemon-reload
3376 pts/11  R+     0:00 ps xa
```

Таким образом, при выходе из системы процесс `nohup1` не завершился, так как команда `nohup` позволила этому процессу игнорировать сигнал `SIGHUP`, который высылается при выходе из системы. Рассмотрим ещё один пример с командой `nohup`, выводящей на экран строку.

*nohup2.c*

```
#include <stdio.h>
void main()
{
    int i;
    for(i = 0; i < 10; i++)
        printf("%d ", i);
}
```

```
lera2003@Valeriya:~/OS_lab34/lb4/5$ nohup ./nohup2
nohup: ввод игнорируется, вывод добавляется в 'nohup.out'
lera2003@Valeriya:~/OS_lab34/lb4/5$ cat nohup.out
0 1 2 3 4 5 6 7 8 9 lera2003@Valeriya:~/OS_lab34/lb4/5$
```

Как видно, в результате работы команды `nohup` запись производилась не в консоль, а в файл `nohup.out`.

**6. Определите `uid` процесса, каково минимальное значение и кому оно принадлежит. Каково минимальное и максимальное значение `pid`, каким процессам принадлежат?**

UUID (universally unique Identifier) — это 128-битное число, которое в разработке ПО используется в качестве уникального идентификатора элементов.

Минимальное значение `uid` равно нулю и принадлежит `root`. В файле `/etc/passwd` на 1-м месте пользователь, а на третьем — его `uid`.

```

lera2003@Valeriya:~/OS_lab34/lb4/6$ sudo cat /etc/passwd | cut -f 1-3 -d ":"
root:x:0
daemon:x:1
bin:x:2
sys:x:3
sync:x:4
games:x:5
man:x:6
lp:x:7
mail:x:8
news:x:9
uucp:x:10
proxy:x:13
www-data:x:33
backup:x:34
list:x:38
irc:x:39
gnats:x:41
nobody:x:65534
systemd-timesync:x:100
systemd-network:x:101
systemd-resolve:x:102
systemd-bus-proxy:x:103
syslog:x:104
_apt:x:105
messagebus:x:106
uidd:x:107
lightdm:x:108
whoopsie:x:109
avahi-autoipd:x:110
avahi:x:111
dnsmasq:x:112
colord:x:113
speech-dispatcher:x:114
hplip:x:115
kernoops:x:116
pulse:x:117
rtkit:x:118
saned:x:119
usbmux:x:120
lera2003:x:1000
guest-254cck:x:999
studentLETI:x:1001
testinguser:x:1002
user1:x:1003
us1:x:1004
sshd:x:121
vboxadd:x:998
guest-dw4cdq:x:997
u1:x:1005

```

Максимальное значение uid зависит от используемой файловой системы и версии ядра. Рассмотрим файловую систему и версию ядра на моей машине

```

u1:x:1005
lera2003@Valeriya:~/OS_lab34/lb4/6$ uname -a
Linux Valeriya 4.15.0-142-generic #146~16.04.1-Ubuntu SMP Tue Apr 13 09:27:15 UTC 2021 x86_64
x86_64 x86_64 GNU/Linux
lera2003@Valeriya:~/OS_lab34/lb4/6$ df -Th | grep "^/dev"
/dev/sda1      ext4          19G          9,2G  8,5G          52% /

```

Для файловой системы ext4 максимальное значение uid составляет 4294967295 ( $2^{32} - 1$ ). Данный параметр можно найти в файле /etc/login.defs.

```
#
# Min/max values for automatic uid selection in useradd
#
UID_MIN          1000
UID_MAX          60000
```

Это минимальное и максимальное значения, которые могут быть выданы пользователю. Остальные зарезервированы под нужды ОС.

Минимальное значение PID равно одному и принадлежит процессу init. Этот процесс запускается после загрузки системы и является родительским процессом для всех остальных. PID 0 зарезервирован ядром. Таким образом, любой новый процесс будет иметь pid больше либо равный двум.

```
lera2003@Valeriya:~/OS_lab34/lb4/6$ ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.4 185488  4304 ?        Ss   май02   0:01 /sbin/init splash
root         2  0.0  0.0      0      0 ?        S    май02   0:00 [kthreadd]
root         4  0.0  0.0      0      0 ?        I<   май02   0:00 [kworker/0:0H]
root         6  0.0  0.0      0      0 ?        I<   май02   0:00 [mm_percpu_wq]
root         7  0.0  0.0      0      0 ?        S    май02   0:00 [ksoftirqd/0]
root         8  0.0  0.0      0      0 ?        I    май02   0:00 [rcu_sched]
```

Максимальное значение PID может варьироваться, при том его можно изменять. Оно находится в файле /proc/sys/kernel/pid\_max

```
lera2003@Valeriya:~/OS_lab34/lb4/6$ cat /proc/sys/kernel/pid_max
32768
```

Посмотреть на множество системных процессов можно с помощью утилиты top -с. Различаются системные процессы от обычных тем, что их имя показано в квадратных скобках.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2190	lera2003	20	0	1324716	124264	36160	S	1,0	12,3	2:48.07	compiz
1619	root	20	0	437428	47292	5388	S	0,6	4,7	0:56.21	/usr/lib/xorg/Xorg -core
1705	lera2003	20	0	245200	44	0	S	0,3	0,0	0:01.56	/usr/bin/VBoxClient --se
1931	lera2003	20	0	526508	4820	12	S	0,3	0,5	0:00.90	/usr/lib/x86_64-linux-gn
2591	lera2003	20	0	663156	18072	10212	S	0,3	1,8	0:17.66	/usr/lib/gnome-terminal/-
3482	lera2003	20	0	43388	3844	3156	R	0,3	0,4	0:00.05	top -c
1	root	20	0	185488	4304	2592	S	0,0	0,4	0:01.80	/sbin/init splash
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	[kthreadd]
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	[kworker/0:0H]
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	[mm_percpu_wq]
7	root	20	0	0	0	0	S	0,0	0,0	0:00.02	[ksoftirqd/0]
8	root	20	0	0	0	0	I	0,0	0,0	0:00.92	[rcu_sched]
9	root	20	0	0	0	0	I	0,0	0,0	0:00.00	[rcu_bh]
10	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	[migration/0]
11	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	[watchdog/0]
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00	[cpuhp/0]
13	root	20	0	0	0	0	S	0,0	0,0	0:00.00	[cpuhp/1]
14	root	rt	0	0	0	0	S	0,0	0,0	0:00.01	[watchdog/1]
15	root	rt	0	0	0	0	S	0,0	0,0	0:00.11	[migration/1]
16	root	20	0	0	0	0	S	0,0	0,0	0:00.09	[ksoftirqd/1]
18	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	[kworker/1:0H]
19	root	20	0	0	0	0	S	0,0	0,0	0:00.00	[cpuhp/2]

## 7. Подготовьте программу, формирующую несколько нитей.

**Нити для эксперимента могут быть практически идентичны.**

Потоки (или нити) стандартизованы в Unix-подобных системах с 2004г., в различных ОС этого семейства допускают различную интерпретацию этого термина, но во всех случаях потоки рассматриваются обязательно как часть процесса, в который они входят, и разделяют ресурсы этого процесса наравне с другими потоками этого процесса (адресное пространство, файловые дескрипторы, обработчики сигналов и т.д.). При создании новых потоков в рамках существующего процесса им нет необходимости создавать собственную копию адресного пространства (и других ресурсов) своего родителя, поэтому требуется значительно меньше затрат, чем при создании нового дочернего процесса. В связи с этим в Linux для обозначения потоков иногда используют термин – легкие процессы (англ. *lightweight processes*).

Потоки одного процесса имеют общий PID, именно этот идентификатор используется при «общении» с многопоточным приложением. Функция `getpid(2)`, возвращает значение идентификатора процесса, фактически группы входящих в него потоков, независимо от того, из какого потока она вызвана. Функции `kill()` `waitpid()` и им подобные по умолчанию также используют идентификаторы групп потоков (англ. *thread*



groups), а не отдельных потоков. Как правило, идентификатор группы равен идентификатору первого потока, входящего в многопоточное приложение.

`pthread_t` — идентификатор потока.

Напишем программу, формирующую несколько нитей. Каждая нить выводит на печать собственное имя и инкрементирует переменную `counter`, соответствующую своему счётчику, выводит на экран сколько в текущий момент времени прошло односекундных или пятисекундных интервалов и текущее время.

*thread.c*

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

void* thread_function1(void *arg)
{
    printf("Thread_1 create\n");
    for (int i = 0; i < 10; i++){
        sleep(5);
        printf("Thread_1. %d 5 sec intervals\n", i+1);
    }
    printf("Thread_1 end\n");
    return NULL;
}

void* thread_function2(void *arg)
{
    printf("Thread_2 create\n");
    for (int i = 0; i < 10; i++){
        sleep(1);
        printf("Thread_2. %d 1 sec intervals\n", i+1);
    }
    printf("Thread_2 end\n");
    return NULL;
}

int main()
{
    pthread_t thread1, thread2;
    int res1, res2;

    res1 = pthread_create(&thread1, NULL, thread_function1, NULL);
    if (res1 != 0){
```

```

    perror("Create 1 flow\n");
    exit(EXIT_FAILURE);
}

res2 = pthread_create(&thread2, NULL, thread_function2, NULL);
if (res2 != 0){
    perror("Create 2 flow\n");
    exit(EXIT_FAILURE);
}

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);

printf("Main flow end\n");
exit(EXIT_SUCCESS);
}

```

```
cc -pthread -o thread thread.c
```

```

lera2003@Valeriya:~/OS_lab34/lb4/7$ ./thread
Thread_1 create
Thread_2 create
Thread_2. 1 1 sec intervals
Thread_2. 2 1 sec intervals
Thread_2. 3 1 sec intervals
Thread_2. 4 1 sec intervals
Thread_1. 1 5 sec intervals
Thread_2. 5 1 sec intervals
Thread_2. 6 1 sec intervals
Thread_2. 7 1 sec intervals
Thread_2. 8 1 sec intervals
Thread_2. 9 1 sec intervals
Thread_1. 2 5 sec intervals
Thread_2. 10 1 sec intervals
Thread_2 end
Thread_1. 3 5 sec intervals
Thread_1. 4 5 sec intervals
Thread_1. 5 5 sec intervals
Thread_1. 6 5 sec intervals
Thread_1. 7 5 sec intervals
Thread_1. 8 5 sec intervals
Thread_1. 9 5 sec intervals
Thread_1. 10 5 sec intervals
Thread_1 end
Main flow end

```

8. После запуска программы проанализируйте выполнение нитей, распределение во времени. Используйте для этого вывод таблицы процессов командой `ps -axhf`.

Проанализируем выполнение нитей, распределение во времени. Для этого используем вывод таблицы `ps -axhf` в разных местах программы (когда, например, будет работать первый и второй поток и когда будет работать только второй поток).

*thread1.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
void* thread_function1(void *arg);
void* thread_function2(void *arg);
char* get_cur_time();

void *thread_function1(void *arg)
{
    int i;
    system("echo Поток 1 создан >> out_file.txt");
    for (i = 0; i < 10; i++) {
        sleep(5); // Засыпаем на 5 секунд
    }
    system("echo Только первый поток работает >> out_file.txt");
    system("ps -axhf >> out_file.txt");
    system("echo Поток 1 завершен >> out_file.txt");
    return NULL;
}

void *thread_function2(void *arg)
{
    int i;
    system("echo Поток 2 создан >> out_file.txt");
    system("echo Работает первый и второй поток >> out_file.txt");
    system("ps -axhf >> out_file.txt");
    for (i = 0; i < 20; i++) {
        sleep(1); // Засыпаем на 1 секунду
    }
    system("echo Поток 2 завершен >> out_file.txt");
    return NULL;
}

char *get_cur_time()
{
    time_t mytime = time(NULL);
    char * time_str = ctime(&mytime);
```

```

    time_str[strlen(time_str)-1] = '\\0';
    return time_str;
}

int main()
{
    pthread_t thread1, thread2;
    int result1, result2;
    system("rm out_file.txt");
    // Создание первого потока
    result1 = pthread_create(&thread1, NULL, thread_function1, NULL);
    // Создание второго потока
    result2 = pthread_create(&thread2, NULL, thread_function2, NULL);

    //system("echo работают оба потока, вызов ps из main:");
    // system("ps -axhf");
    // Ждем завершения потоков
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    system("echo Главный поток завершен >> out_file.txt");
    exit(EXIT_SUCCESS);
}

```

*outfile.txt*

LERA2003@VALERIYA:~/OS\_LAB34/LB4/8\$ CAT OUT\_FILE.TXT

ПОТОК 2 СОЗДАН

ПОТОК 1 СОЗДАН

РАБОТАЕТ ПЕРВЫЙ И ВТОРОЙ ПОТОК

```

 2 ?      S      0:00 [KTHREADD]
 4 ?      I<     0:00 \_ [KWORKER/0:0H]
 6 ?      I<     0:00 \_ [MM_PERCPU_WQ]
 7 ?      S      0:00 \_ [KSOFTIRQD/0]
 8 ?      I      0:01 \_ [RCU_SCHED]
 9 ?      I      0:00 \_ [RCU_BH]
10 ?      S      0:00 \_ [MIGRATION/0]
11 ?      S      0:00 \_ [WATCHDOG/0]
12 ?      S      0:00 \_ [CPUHP/0]
13 ?      S      0:00 \_ [CPUHP/1]
14 ?      S      0:00 \_ [WATCHDOG/1]
15 ?      S      0:00 \_ [MIGRATION/1]
16 ?      S      0:00 \_ [KSOFTIRQD/1]

```

18 ?	I<	0:00	\_ [KWORKER/1:0H]
19 ?	S	0:00	\_ [CPUHP/2]
20 ?	S	0:00	\_ [WATCHDOG/2]
21 ?	S	0:00	\_ [MIGRATION/2]
22 ?	S	0:00	\_ [KSOFITIRQD/2]
24 ?	I<	0:00	\_ [KWORKER/2:0H]
25 ?	S	0:00	\_ [KDEVTMPFS]
26 ?	I<	0:00	\_ [NETNS]
27 ?	S	0:00	\_ [RCU_TASKS_KTHRE]
28 ?	S	0:00	\_ [KAUDITD]
31 ?	I	0:00	\_ [KWORKER/2:1]
32 ?	S	0:00	\_ [KHUNGTASKD]
33 ?	S	0:00	\_ [OOM_REAPER]
34 ?	I<	0:00	\_ [WRITEBACK]
35 ?	S	0:00	\_ [KCOMPACTD0]
36 ?	SN	0:00	\_ [KSMD]
37 ?	SN	0:00	\_ [KHUGEPAGED]
38 ?	I<	0:00	\_ [CRYPTO]
39 ?	I<	0:00	\_ [KINTEGRITYD]
40 ?	I<	0:00	\_ [KBLOCKD]
41 ?	I<	0:00	\_ [ATA_SFF]
42 ?	I<	0:00	\_ [MD]
43 ?	I<	0:00	\_ [EDAC-POLLER]
44 ?	I<	0:00	\_ [DEVFREQ_WQ]
45 ?	I<	0:00	\_ [WATCHDOGD]
48 ?	S	0:01	\_ [KSWAPD0]
49 ?	I<	0:00	\_ [KWORKER/U7:0]
50 ?	S	0:00	\_ [ECRYPTFS-KTHREA]
92 ?	I<	0:00	\_ [KTHROTLD]
93 ?	I<	0:00	\_ [ACPI_THERMAL_PM]
94 ?	S	0:00	\_ [SCSI_EH_0]
95 ?	I<	0:00	\_ [SCSI_TMF_0]
96 ?	S	0:00	\_ [SCSI_EH_1]
97 ?	I<	0:00	\_ [SCSI_TMF_1]
103 ?	I<	0:00	\_ [IPV6_ADDRCONF]
112 ?	I<	0:00	\_ [KSTRP]
129 ?	I<	0:00	\_ [CHARGER_MANAGER]

171 ?	S	0:00	\_ [SCSI_EH_2]
172 ?	I<	0:00	\_ [SCSI_TMF_2]
173 ?	I<	0:00	\_ [TTM_SWAP]
174 ?	S	0:00	\_ [IRQ/18-VMWGFY]
230 ?	I	0:00	\_ [KWORKER/1:2]
232 ?	I<	0:00	\_ [KWORKER/0:1H]
233 ?	I<	0:00	\_ [KWORKER/2:1H]
255 ?	S	0:00	\_ [JBD2/SDA1-8]
256 ?	I<	0:00	\_ [EXT4-RSV-CONVER]
262 ?	I<	0:00	\_ [KWORKER/1:1H]
313 ?	I	0:00	\_ [KWORKER/0:3]
343 ?	S<	0:00	\_ [LOOP0]
344 ?	S<	0:00	\_ [LOOP1]
345 ?	S<	0:00	\_ [LOOP2]
346 ?	S<	0:00	\_ [LOOP3]
347 ?	S<	0:00	\_ [LOOP4]
348 ?	S<	0:00	\_ [LOOP5]
350 ?	S<	0:00	\_ [LOOP7]
351 ?	S<	0:00	\_ [LOOP8]
352 ?	S<	0:00	\_ [LOOP9]
353 ?	S<	0:00	\_ [LOOP10]
459 ?	I<	0:00	\_ [IPRT-VBoxWQUEUE]
3128 ?	I	0:00	\_ [KWORKER/2:0]
3255 ?	S<	0:00	\_ [LOOP11]
3257 ?	I	0:00	\_ [KWORKER/0:0]
3380 ?	S<	0:00	\_ [LOOP12]
3424 ?	I	0:00	\_ [KWORKER/1:0]
3574 ?	I	0:00	\_ [KWORKER/U6:2]
3706 ?	I	0:00	\_ [KWORKER/U6:0]
3729 ?	I	0:00	\_ [KWORKER/U6:1]
1 ?	Ss	0:01	/SBIN/INIT SPLASH
289 ?	Ss	0:00	/LIB/SYSTEMD/SYSTEMD-JOURNALD
324 ?	Ss	0:00	/LIB/SYSTEMD/SYSTEMD-UDEV
940 ?	Ss	0:00	/LIB/SYSTEMD/SYSTEMD-LOGIND
943 ?	Ss	0:00	/USR/SBIN/CUPSD -L
1027 ?	S	0:00	\_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
1029 ?	S	0:00	\_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://

```

1030 ?      S      0:00  \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
946 ?      Ss     0:00  /USR/SBIN/CRON -F
983 ?      Ss     0:00  /USR/SBIN/ACPID
984 ?      SSL    0:00  /USR/SBIN/RSYSLOGD -N
985 ?      Ss     0:00  /USR/BIN/DBUS-DAEMON --SYSTEM --ADDRESS=SYSTEMD: --NOFORK --
NOPIDFILE --SYSTEMD-ACTIVATION
1032 ?      SSL    0:00  /USR/SBIN/CUPS-BROWSED
1034 ?      SSL    0:00  /USR/SBIN/NETWORKMANAGER --NO-DAEMON
1042 ?      SSL    0:00  /USR/LIB/ACCOUNTSSERVICE/ACCOUNTS-DAEMON
1073 ?      SSL    0:22  /USR/LIB/SNAPD/SNAPD
1173 ?      Ss     0:00  /USR/SBIN/IRQBALANCE --PID=/VAR/RUN/IRQBALANCE.PID
1187 ?      SSL    0:00  /USR/LIB/POLICYKIT-1/POLKITD --NO-DEBUG
1310 ?      Ss     0:00  /SBIN/DHCLIENT -1 -V -PF /RUN/DHCLIENT.ENP0S3.PID -LF
/VAR/LIB/DHCP/DHCLIENT.ENP0S3.LEASES -I -DF /VAR/LIB/DHCP/DHCLIENT6.ENP0S3.LEASES ENP0S3
1557 ?      SSL    0:00  /USR/BIN/PYTHON3 /USR/SHARE/UNATTENDED-UPGRADES/UNATTENDED-
UPGRADE-SHUTDOWN --WAIT-FOR-SIGNAL
1558 ?      SSL    0:00  /USR/BIN/WHOOPSIE -F
1590 ?      SSL    0:00  /USR/SBIN/LIGHTDM
1619 TTY7    SSL+   1:23  \_ /USR/LIB/XORG/XORG -CORE :0 -SEAT SEAT0 -AUTH
/VAR/RUN/LIGHTDM/ROOT/:0 -NOLISTEN TCP VT7 -NOVTSWITCH
1670 ?      SL     0:00  \_ LIGHTDM --SESSION-CHILD 12 15
1679 ?      Ss     0:00  \_ /SBIN/UPSTART --USER
1844 ?      S      0:00  \_ UPSTART-UDEV-BRIDGE --DAEMON --USER
1845 ?      Ss     0:00  \_ DBUS-DAEMON --FORK --SESSION --
ADDRESS=UNIX;ABSTRACT=/TMP/DBUS-BJDY7Mn4FX
1857 ?      Ss     0:00  \_ /USR/LIB/x86_64-LINUX-GNU/HUD/WINDOW-STACK-BRIDGE
1879 ?      S      0:00  \_ UPSTART-DBUS-BRIDGE --DAEMON --SESSION --USER --
BUS-NAME SESSION
1881 ?      S      0:00  \_ UPSTART-DBUS-BRIDGE --DAEMON --SYSTEM --USER --
BUS-NAME SYSTEM
1891 ?      S      0:00  \_ UPSTART-FILE-BRIDGE --DAEMON --USER
1894 ?      SSL    0:07  \_ /USR/BIN/IBUS-DAEMON --DAEMONIZE --XIM --ADDRESS
UNIX:TMPDIR=/TMP/IBUS
1928 ?      SL     0:00  | \_ /USR/LIB/IBUS/IBUS-DCONF
1932 ?      SL     0:00  | \_ /USR/LIB/IBUS/IBUS-UI-GTK3
2049 ?      SL     0:02  | \_ /USR/LIB/IBUS/IBUS-ENGINE-SIMPLE
1907 ?      SL     0:00  \_ /USR/LIB/GVFS/GVFS

```

1912 ?	SLL	0:00	\_ GNOME-KEYRING-DAEMON --START --COMPONENTS
PKCS11,SECRETS			
1914 ?	SL	0:00	\_ /USR/LIB/GVFS/GVFS-D-FUSE /RUN/USER/1000/GVFS -F
-O BIG_WRITES			
1920 ?	SS	0:00	\_ GPG-AGENT --HOMEDIR /HOME/LERA2003/.GNUPG --USE-
STANDARD-SOCKET --DAEMON			
1931 ?	SSL	0:01	\_ /USR/LIB/X86_64-LINUX-GNU/BAMF/BAMFDAEMON
1954 ?	SL	0:00	\_ /USR/LIB/AT-SPI2-CORE/AT-SPI-BUS-LAUNCHER
1981 ?	S	0:00	\_ /USR/BIN/DBUS-DAEMON --CONFIG-FILE=/ETC/AT-
SPI2/ACCESSIBILITY.CONF --NOFORK --PRINT-ADDRESS 3			
1963 ?	SL	0:00	\_ /USR/LIB/IBUS/IBUS-X11 --KILL-DAEMON
1970 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/HUD/HUD-SERVICE
1972 ?	SSL	0:01	\_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-SETTINGS-
DAEMON			
1990 ?	SSL	0:00	\_ /USR/LIB/GNOME-SESSION/GNOME-SESSION-BINARY --
SESSION=UBUNTU			
2200 ?	SL	0:00	\_ NM-APPLET
2215 ?	SLL	0:03	\_ /USR/BIN/GNOME-SOFTWARE --GAPPLICATION-SERVICE
2219 ?	SL	0:01	\_ NAUTILUS -N
2221 ?	SL	0:00	\_ /USR/LIB/POLICYKIT-1-GNOME/POLKIT-GNOME-
AUTHENTICATION-AGENT-1			
2244 ?	SL	0:00	\_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-FALLBACK-
MOUNT-HELPER			
2550 ?	SL	0:00	\_ UPDATE-NOTIFIER
2629 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/DEJA-DUP/DEJA-DUP-
MONITOR			
2001 ?	SL	0:00	\_ /USR/LIB/AT-SPI2-CORE/AT-SPI2-REGISTRYD --USE-
GNOME-SESSION			
2002 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/UNITY/UNITY-PANEL-
SERVICE			
2048 ?	SL	0:00	\_ /USR/LIB/DCONF/DCONF-SERVICE
2055 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/NOTIFY-OSD
2065 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
MESSAGES/INDICATOR-MESSAGES-SERVICE			
2066 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
BLUETOOTH/INDICATOR-BLUETOOTH-SERVICE			



```

2067 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
POWER/INDICATOR-POWER-SERVICE
2068 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
DATETIME/INDICATOR-DATETIME-SERVICE
2069 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
KEYBOARD/INDICATOR-KEYBOARD-SERVICE --USE-GTK
2070 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
SOUND/INDICATOR-SOUND-SERVICE
2071 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
PRINTERS/INDICATOR-PRINTERS-SERVICE
2072 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
SESSION/INDICATOR-SESSION-SERVICE
2093 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
APPLICATION/INDICATOR-APPLICATION-SERVICE
2145 ?      S<L    0:00      \_ /USR/BIN/PULSEAUDIO --START --LOG-TARGET=SYSLOG
2152 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-SOURCE-REGISTRY
2159 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY
2296 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-
SUBPROCESS --FACTORY CONTACTS --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159X2 --OWN-PATH
/ORG/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/2
2316 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-
SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159X3 --OWN-PATH
/ORG/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/3
2190 ?      SSL    4:15      \_ COMPIZ
2235 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-UDISKS2-VOLUME-MONITOR
2311 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-GOA-VOLUME-MONITOR
2314 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-FACTORY
2341 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-
FACTORY-SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.ADDRESSBOOKX2314X2 --OWN-PATH
/ORG/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/ADDRESSBOOK/2314/2
2321 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-MTP-VOLUME-MONITOR
2335 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-GPHOTO2-VOLUME-MONITOR
2347 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-AFC-VOLUME-MONITOR

```

```

2402 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-TRASH --SPAWNER :1.3
/ORG/GTK/GVFS/EXEC_SPAW/0
2427 ?      S       0:00      \_ /BIN/SH -C /USR/LIB/X86_64-LINUX-
GNU/ZEITGEIST/ZEITGEIST-MAYBE-VACUUM; /USR/BIN/ZEITGEIST-DAEMON
2431 ?      SL      0:00      | \_ /USR/BIN/ZEITGEIST-DAEMON
2438 ?      SL      0:00      \_ /USR/LIB/X86_64-LINUX-GNU/ZEITGEIST-FTS
2440 ?      SL      0:00      \_ ZEITGEIST-DATAHUB
2520 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-NETWORK --SPAWNER :1.3
/ORG/GTK/GVFS/EXEC_SPAW/2
2574 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-DNSSD --SPAWNER :1.3
/ORG/GTK/GVFS/EXEC_SPAW/7
2591 ?      SL      0:27      \_ /USR/LIB/GNOME-TERMINAL/GNOME-TERMINAL-SERVER
2598 PTS/4   Ss      0:00      \_ BASH
2637 PTS/4   S+      0:00      \_ /BIN/BASH ./WRITE.SH
2639 PTS/4   S+      0:00      \_ SCRIPT -F SESSION2.LOG
2640 PTS/11  Ss      0:00      \_ BASH -I
3276 PTS/11  R       74:38      \_ ./NOHUP1
3744 PTS/11  SL+     0:00      \_ ./THREAD1
3752 PTS/11  S+      0:00      \_ SH -C PS -AXHF >>
OUT_FILE.TXT
3753 PTS/11  R+      0:00      \_ PS -AXHF
1601 ?      Ss      0:00 /USR/SBIN/SSHD -D
1623 TTY1     Ss+     0:00 /SBIN/AGETTY --NOCLEAR TTY1 LINUX
1675 ?      Ss      0:00 /LIB/SYSTEMD/SYSTEMD --USER
1676 ?      S       0:00 \_ (SD-PAM)
1694 ?      S       0:00 /USR/BIN/VBoxCLIENT --CLIPBOARD
1695 ?      SL      0:00 \_ /USR/BIN/VBoxCLIENT --CLIPBOARD
1704 ?      S       0:00 /USR/BIN/VBoxCLIENT --SEAMLESS
1705 ?      SL      0:03 \_ /USR/BIN/VBoxCLIENT --SEAMLESS
1709 ?      S       0:00 /USR/BIN/VBoxCLIENT --DRAGANDDROP
1710 ?      SL      0:07 \_ /USR/BIN/VBoxCLIENT --DRAGANDDROP
1756 ?      S       0:00 /USR/BIN/VBoxCLIENT --VMSVGA
1757 ?      SL      0:00 \_ /USR/BIN/VBoxCLIENT --VMSVGA
2040 ?      SSL     0:00 /USR/LIB/UPOWER/UPOWERD
2160 ?      SNsL    0:00 /USR/LIB/RTKIT/RTKIT-DAEMON
2232 ?      SSL     0:00 /USR/LIB/X86_64-LINUX-GNU/FWUPD/FWUPD
2280 ?      SSL     0:00 /USR/LIB/UDISKS2/UDISKSD --NO-DEBUG

```

2461 ?        SSL    0:00 /USR/LIB/COLORD/COLORD

ПОТОК 2 ЗАВЕРШЕН

ПОТОК 1 СОЗДАН

ПОТОК 2 СОЗДАН

РАБОТАЕТ ПЕРВЫЙ И ВТОРОЙ ПОТОК

2 ?	S	0:00	[KTHREADD]
4 ?	I<	0:00	\_ [KWORKER/0:0H]
6 ?	I<	0:00	\_ [MM_PERCPU_WQ]
7 ?	S	0:00	\_ [KSOFTIRQD/0]
8 ?	I	0:01	\_ [RCU_SCHED]
9 ?	I	0:00	\_ [RCU_BH]
10 ?	S	0:00	\_ [MIGRATION/0]
11 ?	S	0:00	\_ [WATCHDOG/0]
12 ?	S	0:00	\_ [CPUHP/0]
13 ?	S	0:00	\_ [CPUHP/1]
14 ?	S	0:00	\_ [WATCHDOG/1]
15 ?	S	0:00	\_ [MIGRATION/1]
16 ?	S	0:00	\_ [KSOFTIRQD/1]
18 ?	I<	0:00	\_ [KWORKER/1:0H]
19 ?	S	0:00	\_ [CPUHP/2]
20 ?	S	0:00	\_ [WATCHDOG/2]
21 ?	S	0:00	\_ [MIGRATION/2]
22 ?	S	0:00	\_ [KSOFTIRQD/2]
24 ?	I<	0:00	\_ [KWORKER/2:0H]
25 ?	S	0:00	\_ [KDEVTMPFS]
26 ?	I<	0:00	\_ [NETNS]
27 ?	S	0:00	\_ [RCU_TASKS_KTHRE]
28 ?	S	0:00	\_ [KAUDITD]
31 ?	I	0:00	\_ [KWORKER/2:1]
32 ?	S	0:00	\_ [KHUNGTASKD]
33 ?	S	0:00	\_ [OOM_REAPER]
34 ?	I<	0:00	\_ [WRITEBACK]
35 ?	S	0:00	\_ [KCOMPACTD0]
36 ?	SN	0:00	\_ [KSMD]
37 ?	SN	0:00	\_ [KHUGEPAGED]
38 ?	I<	0:00	\_ [CRYPTO]
39 ?	I<	0:00	\_ [KINTEGRITYD]

40 ?	I<	0:00	\_ [KBLOCKD]
41 ?	I<	0:00	\_ [ATA_SFF]
42 ?	I<	0:00	\_ [MD]
43 ?	I<	0:00	\_ [EDAC-POLLER]
44 ?	I<	0:00	\_ [DEVFREQ_WQ]
45 ?	I<	0:00	\_ [WATCHDOG]
48 ?	S	0:01	\_ [KSWAPD0]
49 ?	I<	0:00	\_ [KWORKER/U7:0]
50 ?	S	0:00	\_ [ECRYPTFS-KTHREA]
92 ?	I<	0:00	\_ [KTHROTL]
93 ?	I<	0:00	\_ [ACPI_THERMAL_PM]
94 ?	S	0:00	\_ [SCSI_EH_0]
95 ?	I<	0:00	\_ [SCSI_TMF_0]
96 ?	S	0:00	\_ [SCSI_EH_1]
97 ?	I<	0:00	\_ [SCSI_TMF_1]
103 ?	I<	0:00	\_ [IPV6_ADDRCONF]
112 ?	I<	0:00	\_ [KSTRP]
129 ?	I<	0:00	\_ [CHARGER_MANAGER]
171 ?	S	0:00	\_ [SCSI_EH_2]
172 ?	I<	0:00	\_ [SCSI_TMF_2]
173 ?	I<	0:00	\_ [TTM_SWAP]
174 ?	S	0:00	\_ [IRQ/18-VMWGFX]
230 ?	I	0:00	\_ [KWORKER/1:2]
232 ?	I<	0:00	\_ [KWORKER/0:1H]
233 ?	I<	0:00	\_ [KWORKER/2:1H]
255 ?	S	0:00	\_ [JBD2/SDA1-8]
256 ?	I<	0:00	\_ [EXT4-RSV-CONVER]
262 ?	I<	0:00	\_ [KWORKER/1:1H]
313 ?	I	0:00	\_ [KWORKER/0:3]
343 ?	S<	0:00	\_ [LOOP0]
344 ?	S<	0:00	\_ [LOOP1]
345 ?	S<	0:00	\_ [LOOP2]
346 ?	S<	0:00	\_ [LOOP3]
347 ?	S<	0:00	\_ [LOOP4]
348 ?	S<	0:00	\_ [LOOP5]
350 ?	S<	0:00	\_ [LOOP7]
351 ?	S<	0:00	\_ [LOOP8]

```

352 ?      S<    0:00 \_ [LOOP9]
353 ?      S<    0:00 \_ [LOOP10]
459 ?      I<    0:00 \_ [IPRT-VBoxWQUEUE]
3128 ?     I      0:00 \_ [KWORKER/2:0]
3255 ?     S<    0:00 \_ [LOOP11]
3257 ?     I      0:00 \_ [KWORKER/0:0]
3380 ?     S<    0:00 \_ [LOOP12]
3424 ?     I      0:00 \_ [KWORKER/1:0]
3574 ?     I      0:00 \_ [KWORKER/U6:2]
3706 ?     I      0:00 \_ [KWORKER/U6:0]
3729 ?     I      0:00 \_ [KWORKER/U6:1]
  1 ?      Ss     0:01 /SBIN/INIT SPLASH
289 ?      Ss     0:00 /LIB/SYSTEMD/SYSTEMD-JOURNALD
324 ?      Ss     0:00 /LIB/SYSTEMD/SYSTEMD-UDEV
940 ?      Ss     0:00 /LIB/SYSTEMD/SYSTEMD-LOGIND
943 ?      Ss     0:00 /USR/SBIN/CUPSD -L
1027 ?     S      0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
1029 ?     S      0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
1030 ?     S      0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
946 ?      Ss     0:00 /USR/SBIN/CRON -F
983 ?      Ss     0:00 /USR/SBIN/ACPID
984 ?      Ssl    0:00 /USR/SBIN/RSYSLOGD -N
985 ?      Ss     0:00 /USR/BIN/DBUS-DAEMON --SYSTEM --ADDRESS=SYSTEMD: --NOFORK --
NOPIFILE --SYSTEMD-ACTIVATION
1032 ?     Ssl    0:00 /USR/SBIN/CUPS-BROWSED
1034 ?     Ssl    0:00 /USR/SBIN/NETWORKMANAGER --NO-DAEMON
1042 ?     Ssl    0:00 /USR/LIB/ACCOUNTSSERVICE/ACCOUNTS-DAEMON
1073 ?     Ssl    0:22 /USR/LIB/SNAPD/SNAPD
1173 ?     Ss     0:00 /USR/SBIN/IRQBALANCE --PID=/VAR/RUN/IRQBALANCE.PID
1187 ?     Ssl    0:00 /USR/LIB/POLICYKIT-1/POLKITD --NO-DEBUG
1310 ?     Ss     0:00 /SBIN/DHCLIENT -1 -V -PF /RUN/DHCLIENT.ENP0S3.PID -LF
/VAR/LIB/DHCP/DHCLIENT.ENP0S3.LEASES -I -DF /VAR/LIB/DHCP/DHCLIENT6.ENP0S3.LEASES ENP0S3
1557 ?     Ssl    0:00 /USR/BIN/PYTHON3 /USR/SHARE/UNATTENDED-UPGRADES/UNATTENDED-
UPGRADE-SHUTDOWN --WAIT-FOR-SIGNAL
1558 ?     Ssl    0:00 /USR/BIN/WHOOPSIE -F
1590 ?     Ssl    0:00 /USR/SBIN/LIGHTDM

```

```

1619 TTY7      SSL+   1:24  \_ /USR/LIB/XORG/XORG -CORE :0 -SEAT SEAT0 -AUTH
/VAR/RUN/LIGHTDM/ROOT/:0 -NOLISTEN TCP VT7 -NOVTSWITCH
1670 ?        SL      0:00  \_ LIGHTDM --SESSION-CHILD 12 15
1679 ?        Ss      0:00    \_ /SBIN/UPSTART --USER
1844 ?        S       0:00    \_ UPSTART-UDEV-BRIDGE --DAEMON --USER
1845 ?        Ss      0:00    \_ DBUS-DAEMON --FORK --SESSION --
ADDRESS=UNIX:ABSTRACT=/TMP/DBUS-BJDY7Mn4FX
1857 ?        Ss      0:00    \_ /USR/LIB/X86_64-LINUX-GNU/HUD/WINDOW-STACK-BRIDGE
1879 ?        S       0:00    \_ UPSTART-DBUS-BRIDGE --DAEMON --SESSION --USER --
BUS-NAME SESSION
1881 ?        S       0:00    \_ UPSTART-DBUS-BRIDGE --DAEMON --SYSTEM --USER --
BUS-NAME SYSTEM
1891 ?        S       0:00    \_ UPSTART-FILE-BRIDGE --DAEMON --USER
1894 ?        SSL     0:08    \_ /USR/BIN/IBUS-DAEMON --DAEMONIZE --XIM --ADDRESS
UNIX:TMPDIR=/TMP/IBUS
1928 ?        SL      0:00    | \_ /USR/LIB/IBUS/IBUS-DCONF
1932 ?        SL      0:00    | \_ /USR/LIB/IBUS/IBUS-UI-GTK3
2049 ?        SL      0:03    | \_ /USR/LIB/IBUS/IBUS-ENGINE-SIMPLE
1907 ?        SL      0:00    \_ /USR/LIB/GVFS/GVFS
1912 ?        SLL     0:00    \_ GNOME-KEYRING-DAEMON --START --COMPONENTS
PKCS11,SECRETS
1914 ?        SL      0:00    \_ /USR/LIB/GVFS/GVFS-FUSE /RUN/USER/1000/GVFS -F
-O BIG_WRITES
1920 ?        Ss      0:00    \_ GPG-AGENT --HOMEDIR /HOME/LERA2003/.GNUPG --USE-
STANDARD-SOCKET --DAEMON
1931 ?        SSL     0:01    \_ /USR/LIB/X86_64-LINUX-GNU/BAMF/BAMFDAEMON
1954 ?        SL      0:00    \_ /USR/LIB/AT-SPI2-CORE/AT-SPI-BUS-LAUNCHER
1981 ?        S       0:00    | \_ /USR/BIN/DBUS-DAEMON --CONFIG-FILE=/ETC/AT-
SPI2/ACCESSIBILITY.CONF --NOFORK --PRINT-ADDRESS 3
1963 ?        SL      0:00    \_ /USR/LIB/IBUS/IBUS-X11 --KILL-DAEMON
1970 ?        SSL     0:00    \_ /USR/LIB/X86_64-LINUX-GNU/HUD/HUD-SERVICE
1972 ?        SSL     0:01    \_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-SETTINGS-
DAEMON
1990 ?        SSL     0:00    \_ /USR/LIB/GNOME-SESSION/GNOME-SESSION-BINARY --
SESSION=UBUNTU
2200 ?        SL      0:00    | \_ NM-APPLET
2215 ?        SLL     0:03    | \_ /USR/BIN/GNOME-SOFTWARE --GAPPLICATION-SERVICE

```

2219 ?	SL	0:01	\_ NAUTILUS -N
2221 ?	SL	0:00	\_ /USR/LIB/POLICYKIT-1-GNOME/POLKIT-GNOME-AUTHENTICATION-AGENT-1
2244 ?	SL	0:00	\_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-FALLBACK-MOUNT-HELPER
2550 ?	SL	0:00	\_ UPDATE-NOTIFIER
2629 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/DEJA-DUP/DEJA-DUP-MONITOR
2001 ?	SL	0:00	\_ /USR/LIB/AT-SPI2-CORE/AT-SPI2-REGISTRYD --USE-GNOME-SESSION
2002 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/UNITY/UNITY-PANEL-SERVICE
2048 ?	SL	0:00	\_ /USR/LIB/DCONF/DCONF-SERVICE
2055 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/NOTIFY-OSD
2065 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-MESSAGES/INDICATOR-MESSAGES-SERVICE
2066 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-BLUETOOTH/INDICATOR-BLUETOOTH-SERVICE
2067 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-POWER/INDICATOR-POWER-SERVICE
2068 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-DATETIME/INDICATOR-DATETIME-SERVICE
2069 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-KEYBOARD/INDICATOR-KEYBOARD-SERVICE --USE-GTK
2070 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-SOUND/INDICATOR-SOUND-SERVICE
2071 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-PRINTERS/INDICATOR-PRINTERS-SERVICE
2072 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-SESSION/INDICATOR-SESSION-SERVICE
2093 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-APPLICATION/INDICATOR-APPLICATION-SERVICE
2145 ?	S<L	0:00	\_ /USR/BIN/PULSEAUDIO --START --LOG-TARGET=SYSLOG
2152 ?	SL	0:00	\_ /USR/LIB/EVOLUTION/EVOLUTION-SOURCE-REGISTRY
2159 ?	SL	0:00	\_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY
2296 ?	SL	0:00	\_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-SUBPROCESS --FACTORY CONTACTS --BUS-NAME

```

ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159x2 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/2
2316 ?      SL      0:00      |   \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-
SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159x3 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/3
2190 ?      SSL     4:18      \_ COMPIZ
2235 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-UDISKS2-VOLUME-MONITOR
2311 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-GOA-VOLUME-MONITOR
2314 ?      SL      0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-FACTORY
2341 ?      SL      0:00      |   \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-
FACTORY-SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.ADDRESSBOOKX2314x2 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/ADDRESSBOOK/2314/2
2321 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-MTP-VOLUME-MONITOR
2335 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-GPHOTO2-VOLUME-MONITOR
2347 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-AFC-VOLUME-MONITOR
2402 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFSD-TRASH --SPAWNER :1.3
/ORC/GTK/GVFS/EXEC_SPAW/0
2427 ?      S       0:00      \_ /BIN/SH -C /USR/LIB/X86_64-LINUX-
GNU/ZEITGEIST/ZEITGEIST-MAYBE-VACUUM; /USR/BIN/ZEITGEIST-DAEMON
2431 ?      SL      0:00      |   \_ /USR/BIN/ZEITGEIST-DAEMON
2438 ?      SL      0:00      \_ /USR/LIB/X86_64-LINUX-GNU/ZEITGEIST-FTS
2440 ?      SL      0:00      \_ ZEITGEIST-DATAHUB
2520 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFSD-NETWORK --SPAWNER :1.3
/ORC/GTK/GVFS/EXEC_SPAW/2
2574 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFSD-DNSSD --SPAWNER :1.3
/ORC/GTK/GVFS/EXEC_SPAW/7
2591 ?      RL      0:27      \_ /USR/LIB/GNOME-TERMINAL/GNOME-TERMINAL-SERVER
2598 PTS/4   SS      0:00      \_ BASH
2637 PTS/4   S+      0:00      \_ /BIN/BASH ./WRITE.SH
2639 PTS/4   S+      0:00      \_ SCRIPT -F SESSION2.LOG
2640 PTS/11  SS      0:00      \_ BASH -I
3276 PTS/11  R       75:27      \_ ./NOHUP1
3763 PTS/11  SL+     0:00      \_ ./THREAD1
3771 PTS/11  S+      0:00      \_ SH -C PS -AXHF >>
OUT_FILE.TXT

```



```

3772 PTS/11 R+ 0:00 \_ PS -AXHF
1601 ? Ss 0:00 /USR/SBIN/SSHD -D
1623 TTY1 Ss+ 0:00 /SBIN/AGETTY --NOCLEAR TTY1 LINUX
1675 ? Ss 0:00 /LIB/SYSTEMD/SYSTEMD --USER
1676 ? S 0:00 \_ (SD-PAM)
1694 ? S 0:00 /USR/BIN/VBoxClient --CLIPBOARD
1695 ? SL 0:00 \_ /USR/BIN/VBoxClient --CLIPBOARD
1704 ? S 0:00 /USR/BIN/VBoxClient --SEAMLESS
1705 ? SL 0:03 \_ /USR/BIN/VBoxClient --SEAMLESS
1709 ? S 0:00 /USR/BIN/VBoxClient --DRAGANDDROP
1710 ? SL 0:07 \_ /USR/BIN/VBoxClient --DRAGANDDROP
1756 ? S 0:00 /USR/BIN/VBoxClient --VMSVGA
1757 ? SL 0:00 \_ /USR/BIN/VBoxClient --VMSVGA
2040 ? SSL 0:00 /USR/LIB/UPOWER/UPOWERD
2160 ? SNSL 0:00 /USR/LIB/RTKIT/RTKIT-DAEMON
2232 ? SSL 0:00 /USR/LIB/X86_64-LINUX-GNU/FWUPD/FWUPD
2280 ? SSL 0:00 /USR/LIB/UDISKS2/UDISKSD --NO-DEBUG
2461 ? SSL 0:00 /USR/LIB/COLOR/0/COLOR

```

ПОТОК 2 ЗАВЕРШЕН

ТОЛЬКО ПЕРВЫЙ ПОТОК РАБОТАЕТ

```

2 ? S 0:00 [KTHREADD]
4 ? I< 0:00 \_ [KWORKER/0:0H]
6 ? I< 0:00 \_ [MM_PERCPU_WQ]
7 ? S 0:00 \_ [KSOFTIRQD/0]
8 ? I 0:01 \_ [RCU_SCHED]
9 ? I 0:00 \_ [RCU_BH]
10 ? S 0:00 \_ [MIGRATION/0]
11 ? S 0:00 \_ [WATCHDOG/0]
12 ? S 0:00 \_ [CPUHP/0]
13 ? S 0:00 \_ [CPUHP/1]
14 ? S 0:00 \_ [WATCHDOG/1]
15 ? S 0:00 \_ [MIGRATION/1]
16 ? S 0:00 \_ [KSOFTIRQD/1]
18 ? I< 0:00 \_ [KWORKER/1:0H]
19 ? S 0:00 \_ [CPUHP/2]
20 ? S 0:00 \_ [WATCHDOG/2]
21 ? S 0:00 \_ [MIGRATION/2]

```

22 ?	S	0:00	\_ [KSOFTIRQD/2]
24 ?	I<	0:00	\_ [KWORKER/2:0H]
25 ?	S	0:00	\_ [KDEVTMPFS]
26 ?	I<	0:00	\_ [NETNS]
27 ?	S	0:00	\_ [RCU_TASKS_KTHRE]
28 ?	S	0:00	\_ [KAUDITD]
31 ?	I	0:00	\_ [KWORKER/2:1]
32 ?	S	0:00	\_ [KHUNGTASKD]
33 ?	S	0:00	\_ [OOM_REAPER]
34 ?	I<	0:00	\_ [WRITEBACK]
35 ?	S	0:00	\_ [KCOMPACTD0]
36 ?	SN	0:00	\_ [KSMD]
37 ?	SN	0:00	\_ [KHUGEPAGED]
38 ?	I<	0:00	\_ [CRYPTO]
39 ?	I<	0:00	\_ [KINTEGRITYD]
40 ?	I<	0:00	\_ [KBLOCKD]
41 ?	I<	0:00	\_ [ATA_SFF]
42 ?	I<	0:00	\_ [MD]
43 ?	I<	0:00	\_ [EDAC-POLLER]
44 ?	I<	0:00	\_ [DEVFREQ_WQ]
45 ?	I<	0:00	\_ [WATCHDOG]
48 ?	S	0:01	\_ [KSWAPD0]
49 ?	I<	0:00	\_ [KWORKER/U7:0]
50 ?	S	0:00	\_ [ECRYPTFS-KTHREA]
92 ?	I<	0:00	\_ [KTHROTLD]
93 ?	I<	0:00	\_ [ACPI_THERMAL_PM]
94 ?	S	0:00	\_ [SCSI_EH_0]
95 ?	I<	0:00	\_ [SCSI_TMF_0]
96 ?	S	0:00	\_ [SCSI_EH_1]
97 ?	I<	0:00	\_ [SCSI_TMF_1]
103 ?	I<	0:00	\_ [IPV6_ADDRCONF]
112 ?	I<	0:00	\_ [KSTRP]
129 ?	I<	0:00	\_ [CHARGER_MANAGER]
171 ?	S	0:00	\_ [SCSI_EH_2]
172 ?	I<	0:00	\_ [SCSI_TMF_2]
173 ?	I<	0:00	\_ [TTM_SWAP]
174 ?	S	0:00	\_ [IRQ/18-VMWGFY]

230 ?	I	0:00 \_ [KWORKER/1:2]
232 ?	I<	0:00 \_ [KWORKER/0:1H]
233 ?	I<	0:00 \_ [KWORKER/2:1H]
255 ?	S	0:00 \_ [JBD2/SDA1-8]
256 ?	I<	0:00 \_ [EXT4-RSV-CONVER]
262 ?	I<	0:00 \_ [KWORKER/1:1H]
313 ?	I	0:00 \_ [KWORKER/0:3]
343 ?	S<	0:00 \_ [LOOP0]
344 ?	S<	0:00 \_ [LOOP1]
345 ?	S<	0:00 \_ [LOOP2]
346 ?	S<	0:00 \_ [LOOP3]
347 ?	S<	0:00 \_ [LOOP4]
348 ?	S<	0:00 \_ [LOOP5]
350 ?	S<	0:00 \_ [LOOP7]
351 ?	S<	0:00 \_ [LOOP8]
352 ?	S<	0:00 \_ [LOOP9]
353 ?	S<	0:00 \_ [LOOP10]
459 ?	I<	0:00 \_ [IPRT-VBoxWQUEUE]
3128 ?	I	0:00 \_ [KWORKER/2:0]
3255 ?	S<	0:00 \_ [LOOP11]
3257 ?	I	0:00 \_ [KWORKER/0:0]
3380 ?	S<	0:00 \_ [LOOP12]
3424 ?	I	0:00 \_ [KWORKER/1:0]
3574 ?	I	0:00 \_ [KWORKER/U6:2]
3706 ?	I	0:00 \_ [KWORKER/U6:0]
3729 ?	I	0:00 \_ [KWORKER/U6:1]
1 ?	SS	0:01 /SBIN/INIT SPLASH
289 ?	SS	0:00 /LIB/SYSTEMD/SYSTEMD-JOURNALD
324 ?	SS	0:00 /LIB/SYSTEMD/SYSTEMD-UDEV
940 ?	SS	0:00 /LIB/SYSTEMD/SYSTEMD-LOGIND
943 ?	SS	0:00 /USR/SBIN/CUPSD -L
1027 ?	S	0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
1029 ?	S	0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
1030 ?	S	0:00 \_ /USR/LIB/CUPS/NOTIFIER/DBUS DBUS://
946 ?	SS	0:00 /USR/SBIN/CRON -F
983 ?	SS	0:00 /USR/SBIN/ACPID
984 ?	SSL	0:00 /USR/SBIN/RSYSLOGD -N

```

985 ?      Ss      0:00 /USR/BIN/DBUS-DAEMON --SYSTEM --ADDRESS=SYSTEMD: --NOFORK --
NOPIDFILE --SYSTEMD-ACTIVATION
1032 ?      Ssl      0:00 /USR/SBIN/CUPS-BROWSED
1034 ?      Ssl      0:00 /USR/SBIN/NETWORKMANAGER --NO-DAEMON
1042 ?      Ssl      0:00 /USR/LIB/ACCOUNTSSERVICE/ACCOUNTS-DAEMON
1073 ?      Ssl      0:22 /USR/LIB/SNAPD/SNAPD
1173 ?      Ss      0:00 /USR/SBIN/IRQBALANCE --PID=/VAR/RUN/IRQBALANCE.PID
1187 ?      Ssl      0:00 /USR/LIB/POLICYKIT-1/POLKITD --NO-DEBUG
1310 ?      Ss      0:00 /SBIN/DHCLIENT -1 -V -PF /RUN/DHCLIENT.ENP0S3.PID -LF
/VAR/LIB/DHCP/DHCLIENT.ENP0S3.LEASES -I -DF /VAR/LIB/DHCP/DHCLIENT6.ENP0S3.LEASES ENP0S3
1557 ?      Ssl      0:00 /USR/BIN/PYTHON3 /USR/SHARE/UNATTENDED-UPGRADES/UNATTENDED-
UPGRADE-SHUTDOWN --WAIT-FOR-SIGNAL
1558 ?      Ssl      0:00 /USR/BIN/WHOOPIE -F
1590 ?      Ssl      0:00 /USR/SBIN/LIGHTDM
1619 TTY7    Ssl+    1:24 \_ /USR/LIB/XORG/XORG -CORE :0 -SEAT SEAT0 -AUTH
/VAR/RUN/LIGHTDM/ROOT/:0 -NOLISTEN TCP VT7 -NOVTSWITCH
1670 ?      Sl      0:00 \_ LIGHTDM --SESSION-CHILD 12 15
1679 ?      Ss      0:00 \_ /SBIN/UPSTART --USER
1844 ?      S      0:00 \_ UPSTART-UDEV-BRIDGE --DAEMON --USER
1845 ?      Ss      0:00 \_ DBUS-DAEMON --FORK --SESSION --
ADDRESS=UNIX:ABSTRACT=/TMP/DBUS-BJDY7Mn4FX
1857 ?      Ss      0:00 \_ /USR/LIB/X86_64-LINUX-GNU/HUD/WINDOW-STACK-BRIDGE
1879 ?      S      0:00 \_ UPSTART-DBUS-BRIDGE --DAEMON --SESSION --USER --
BUS-NAME SESSION
1881 ?      S      0:00 \_ UPSTART-DBUS-BRIDGE --DAEMON --SYSTEM --USER --
BUS-NAME SYSTEM
1891 ?      S      0:00 \_ UPSTART-FILE-BRIDGE --DAEMON --USER
1894 ?      Ssl      0:08 \_ /USR/BIN/IBUS-DAEMON --DAEMONIZE --XIM --ADDRESS
UNIX:TMPDIR=/TMP/IBUS
1928 ?      Sl      0:00 | \_ /USR/LIB/IBUS/IBUS-DCONF
1932 ?      Sl      0:00 | \_ /USR/LIB/IBUS/IBUS-UI-GTK3
2049 ?      Sl      0:03 | \_ /USR/LIB/IBUS/IBUS-ENGINE-SIMPLE
1907 ?      Sl      0:00 \_ /USR/LIB/GVFS/GVFS
1912 ?      Sll      0:00 \_ GNOME-KEYRING-DAEMON --START --COMPONENTS
PKCS11,SECRETS
1914 ?      Sl      0:00 \_ /USR/LIB/GVFS/GVFS-FUSE /RUN/USER/1000/GVFS -F
-O BIG_WRITES

```

1920 ?	Ss	0:00	\_ GPG-AGENT --HOMEDIR /HOME/LERA2003/.GNUPG --USE- STANDARD-SOCKET --DAEMON
1931 ?	SSL	0:01	\_ /USR/LIB/X86_64-LINUX-GNU/BAMF/BAMFDAEMON
1954 ?	SL	0:00	\_ /USR/LIB/AT-SPI2-CORE/AT-SPI-BUS-LAUNCHER
1981 ?	S	0:00	\_ /USR/BIN/DBUS-DAEMON --CONFIG-FILE=/ETC/AT- SPI2/ACCESSIBILITY.CONF --NOFORK --PRINT-ADDRESS 3
1963 ?	SL	0:00	\_ /USR/LIB/IBUS/IBUS-X11 --KILL-DAEMON
1970 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/HUD/HUD-SERVICE
1972 ?	SSL	0:01	\_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-SETTINGS- DAEMON
1990 ?	SSL	0:00	\_ /USR/LIB/GNOME-SESSION/GNOME-SESSION-BINARY -- SESSION=UBUNTU
2200 ?	SL	0:00	\_ NM-APPLET
2215 ?	SLL	0:03	\_ /USR/BIN/GNOME-SOFTWARE --GAPPLICATION-SERVICE
2219 ?	SL	0:01	\_ NAUTILUS -N
2221 ?	SL	0:00	\_ /USR/LIB/POLICYKIT-1-GNOME/POLKIT-GNOME- AUTHENTICATION-AGENT-1
2244 ?	SL	0:00	\_ /USR/LIB/UNITY-SETTINGS-DAEMON/UNITY-FALLBACK- MOUNT-HELPER
2550 ?	SL	0:00	\_ UPDATE-NOTIFIER
2629 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/DEJA-DUP/DEJA-DUP- MONITOR
2001 ?	SL	0:00	\_ /USR/LIB/AT-SPI2-CORE/AT-SPI2-REGISTRYD --USE- GNOME-SESSION
2002 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/UNITY/UNITY-PANEL- SERVICE
2048 ?	SL	0:00	\_ /USR/LIB/DCONF/DCONF-SERVICE
2055 ?	SL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/NOTIFY-OSD
2065 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR- MESSAGES/INDICATOR-MESSAGES-SERVICE
2066 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR- BLUETOOTH/INDICATOR-BLUETOOTH-SERVICE
2067 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR- POWER/INDICATOR-POWER-SERVICE
2068 ?	SSL	0:00	\_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR- DATETIME/INDICATOR-DATETIME-SERVICE

```

2069 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
KEYBOARD/INDICATOR-KEYBOARD-SERVICE --USE-GTK
2070 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
SOUND/INDICATOR-SOUND-SERVICE
2071 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
PRINTERS/INDICATOR-PRINTERS-SERVICE
2072 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
SESSION/INDICATOR-SESSION-SERVICE
2093 ?      SSL    0:00      \_ /USR/LIB/X86_64-LINUX-GNU/INDICATOR-
APPLICATION/INDICATOR-APPLICATION-SERVICE
2145 ?      S<L    0:00      \_ /USR/BIN/PULSEAUDIO --START --LOG-TARGET=SYSLOG
2152 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-SOURCE-REGISTRY
2159 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY
2296 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-
SUBPROCESS --FACTORY CONTACTS --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159x2 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/2
2316 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-CALENDAR-FACTORY-
SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.CALENDARX2159x3 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/CALENDAR/2159/3
2190 ?      SSL    4:19      \_ COMPIZ
2235 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-UDISKS2-VOLUME-MONITOR
2311 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-GOA-VOLUME-MONITOR
2314 ?      SL     0:00      \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-FACTORY
2341 ?      SL     0:00      | \_ /USR/LIB/EVOLUTION/EVOLUTION-ADDRESSBOOK-
FACTORY-SUBPROCESS --FACTORY LOCAL --BUS-NAME
ORG.GNOME.EVOLUTION.DATASERVER.SUBPROCESS.BACKEND.ADDRESSBOOKX2314x2 --OWN-PATH
/ORC/GNOME/EVOLUTION/DATASERVER/SUBPROCESS/BACKEND/ADDRESSBOOK/2314/2
2321 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-MTP-VOLUME-MONITOR
2335 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-GPHOTO2-VOLUME-MONITOR
2347 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-AFC-VOLUME-MONITOR
2402 ?      SL     0:00      \_ /USR/LIB/GVFS/GVFS-TRASH --SPAWNER :1.3
/ORC/GTK/GVFS/EXEC_SPAW/0
2427 ?      S      0:00      \_ /BIN/SH -C /USR/LIB/X86_64-LINUX-
GNU/ZEITGEIST/ZEITGEIST-MAYBE-VACUUM; /USR/BIN/ZEITGEIST-DAEMON
2431 ?      SL     0:00      | \_ /USR/BIN/ZEITGEIST-DAEMON

```

```

2438 ?      SL      0:00      \_ /USR/LIB/X86_64-LINUX-GNU/ZEITGEIST-FTS
2440 ?      SL      0:00      \_ ZEITGEIST-DATAHUB
2520 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-D-NETWORK --SPAWNER :1.3
/ORG/GTK/GVFS/EXEC_SPAW/2
2574 ?      SL      0:00      \_ /USR/LIB/GVFS/GVFS-D-DNSD --SPAWNER :1.3
/ORG/GTK/GVFS/EXEC_SPAW/7
2591 ?      SL      0:27      \_ /USR/LIB/GNOME-TERMINAL/GNOME-TERMINAL-SERVER
2598 PTS/4   SS      0:00      \_ BASH
2637 PTS/4   S+      0:00      \_ /BIN/BASH ./WRITE.SH
2639 PTS/4   S+      0:00      \_ SCRIPT -F SESSION2.LOG
2640 PTS/11  SS      0:00      \_ BASH -I
3276 PTS/11  R       76:17      \_ ./NOHUP1
3763 PTS/11  SL+     0:00      \_ ./THREAD1
3775 PTS/11  S+      0:00      \_ SH -C PS -AXHF >>
OUT_FILE.TXT
3776 PTS/11  R+      0:00      \_ PS -AXHF
1601 ?      SS      0:00 /USR/SBIN/SSHD -D
1623 TTY1     SS+     0:00 /SBIN/AGETTY --NOCLEAR TTY1 LINUX
1675 ?      SS      0:00 /LIB/SYSTEMD/SYSTEMD --USER
1676 ?      S       0:00 \_ (SD-PAM)
1694 ?      S       0:00 /USR/BIN/VBoxCLIENT --CLIPBOARD
1695 ?      SL      0:00 \_ /USR/BIN/VBoxCLIENT --CLIPBOARD
1704 ?      S       0:00 /USR/BIN/VBoxCLIENT --SEAMLESS
1705 ?      SL      0:03 \_ /USR/BIN/VBoxCLIENT --SEAMLESS
1709 ?      S       0:00 /USR/BIN/VBoxCLIENT --DRAGANDDROP
1710 ?      SL      0:07 \_ /USR/BIN/VBoxCLIENT --DRAGANDDROP
1756 ?      S       0:00 /USR/BIN/VBoxCLIENT --VMSVGA
1757 ?      SL      0:00 \_ /USR/BIN/VBoxCLIENT --VMSVGA
2040 ?      SSL     0:00 /USR/LIB/UPOWER/UPOWERD
2160 ?      SNLS    0:00 /USR/LIB/RTKIT/RTKIT-DAEMON
2232 ?      SSL     0:00 /USR/LIB/X86_64-LINUX-GNU/FWUPD/FWUPD
2280 ?      SSL     0:00 /USR/LIB/UDISKS2/UDISKSD --NO-DEBUG
2461 ?      SSL     0:00 /USR/LIB/COLOR/COLOR

```

ПОТОК 1 ЗАВЕРШЕН

ГЛАВНЫЙ ПОТОК ЗАВЕРШЕН

Для наглядности изменим программу так: все выводы перенаправим в файл (чтобы можно было без проблем запустить её в фон), а анализировать работу будем через утилиту htop.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1729	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1728	lera2003	20	0	303M	96	0	S	0.0	0.0	0:07.77	VBoxClient
1727	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1726	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.00	VBoxClient
1704	lera2003	20	0	46908	32	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --seamless
1705	lera2003	20	0	239M	44	0	S	0.0	0.0	0:04.00	/usr/bin/VBoxClient --seamless
1725	lera2003	20	0	239M	44	0	S	0.0	0.0	0:03.17	VBoxClient
1724	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.81	VBoxClient
1723	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.00	VBoxClient
1694	lera2003	20	0	46908	36	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --clipboard
1695	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.08	/usr/bin/VBoxClient --clipboard
1722	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.05	VBoxClient
1721	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.02	VBoxClient
1720	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.00	VBoxClient
1675	lera2003	20	0	45320	1100	724	S	0.0	0.1	0:00.04	/lib/systemd/systemd --user
1676	lera2003	20	0	63568	456	0	S	0.0	0.0	0:00.00	(sd-pan)
1623	root	20	0	17452	0	0	S	0.0	0.0	0:00.00	/sbin/agetty --noclear tty1 linux
1601	root	20	0	65512	0	0	S	0.0	0.0	0:00.00	/usr/sbin/sshd -D
1590	root	20	0	342M	764	456	S	0.0	0.1	0:00.02	/usr/sbin/lightdm
1670	root	20	0	222M	440	440	S	0.0	0.0	0:00.00	lightdm --session-child 12 15
1679	lera2003	20	0	48224	2328	1468	S	0.0	0.2	0:00.20	/sbin/upstart --user
2591	lera2003	20	0	650M	26704	15824	S	0.7	2.6	0:31.94	/usr/lib/gnome-terminal/gnome-terminal-server
2598	lera2003	20	0	24196	1988	260	S	0.0	0.2	0:00.04	bash
2637	lera2003	20	0	14056	468	260	S	0.0	0.0	0:00.00	/bin/bash ./write.sh
2639	lera2003	20	0	23724	224	44	S	0.0	0.0	0:00.75	script -f session2.log
2640	lera2003	20	0	24092	4572	2748	S	0.0	0.5	0:00.20	bash -i
3873	lera2003	20	0	28452	4760	3220	R	0.7	0.5	0:00.15	htop
3863	lera2003	20	0	22912	700	620	S	0.0	0.1	0:00.00	./thread1
3867	lera2003	20	0	22912	700	620	S	0.0	0.1	0:00.00	thread1
3866	lera2003	20	0	22912	700	620	S	0.0	0.1	0:00.00	thread1
3276	lera2003	20	0	4220	604	536	R	100.	0.1	1h23:40	./nohup1

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1729	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1728	lera2003	20	0	303M	96	0	S	0.0	0.0	0:07.79	VBoxClient
1727	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1726	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.00	VBoxClient
1704	lera2003	20	0	46908	32	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --seamless
1705	lera2003	20	0	239M	44	0	S	0.0	0.0	0:04.01	/usr/bin/VBoxClient --seamless
1725	lera2003	20	0	239M	44	0	S	0.0	0.0	0:03.18	VBoxClient
1724	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.81	VBoxClient
1723	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.00	VBoxClient
1694	lera2003	20	0	46908	36	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --clipboard
1695	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.08	/usr/bin/VBoxClient --clipboard
1722	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.05	VBoxClient
1721	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.02	VBoxClient
1720	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.00	VBoxClient
1675	lera2003	20	0	45320	1100	724	S	0.0	0.1	0:00.04	/lib/systemd/systemd --user
1676	lera2003	20	0	63568	456	0	S	0.0	0.0	0:00.00	(sd-pan)
1623	root	20	0	17452	0	0	S	0.0	0.0	0:00.00	/sbin/agetty --noclear tty1 linux
1601	root	20	0	65512	0	0	S	0.0	0.0	0:00.00	/usr/sbin/sshd -D
1590	root	20	0	342M	764	456	S	0.0	0.1	0:00.02	/usr/sbin/lightdm
1670	root	20	0	222M	440	440	S	0.0	0.0	0:00.00	lightdm --session-child 12 15
1679	lera2003	20	0	48224	2328	1468	S	0.0	0.2	0:00.20	/sbin/upstart --user
2591	lera2003	20	0	650M	26704	15824	S	2.0	2.6	0:32.03	/usr/lib/gnome-terminal/gnome-terminal-server
2598	lera2003	20	0	24196	1988	260	S	0.0	0.2	0:00.04	bash
2637	lera2003	20	0	14056	468	260	S	0.0	0.0	0:00.00	/bin/bash ./write.sh
2639	lera2003	20	0	23724	224	44	S	0.0	0.0	0:00.75	script -f session2.log
2640	lera2003	20	0	24092	4572	2748	S	0.0	0.5	0:00.20	bash -i
3873	lera2003	20	0	28452	4760	3220	R	1.3	0.5	0:00.26	htop
3863	lera2003	20	0	22912	764	684	S	0.0	0.1	0:00.00	./thread1
3866	lera2003	20	0	22912	764	684	S	0.0	0.1	0:00.00	thread1

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1729	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1728	lera2003	20	0	303M	96	0	S	0.0	0.0	0:07.84	VBoxClient
1727	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.02	VBoxClient
1726	lera2003	20	0	303M	96	0	S	0.0	0.0	0:00.00	VBoxClient
1704	lera2003	20	0	46908	32	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --seamless
1705	lera2003	20	0	239M	44	0	S	0.7	0.0	0:04.03	/usr/bin/VBoxClient --seamless
1725	lera2003	20	0	239M	44	0	S	0.0	0.0	0:03.19	VBoxClient
1724	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.82	VBoxClient
1723	lera2003	20	0	239M	44	0	S	0.0	0.0	0:00.00	VBoxClient
1694	lera2003	20	0	46908	36	0	S	0.0	0.0	0:00.00	/usr/bin/VBoxClient --clipboard
1695	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.08	/usr/bin/VBoxClient --clipboard
1722	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.05	VBoxClient
1721	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.02	VBoxClient
1720	lera2003	20	0	241M	360	0	S	0.0	0.0	0:00.00	VBoxClient
1675	lera2003	20	0	45320	1100	724	S	0.0	0.1	0:00.04	/lib/systemd/systemd --user
1676	lera2003	20	0	63568	456	0	S	0.0	0.0	0:00.00	(sd-pan)
1623	root	20	0	17452	0	0	S	0.0	0.0	0:00.00	/sbin/agetty --noclear tty1 linux
1601	root	20	0	65512	0	0	S	0.0	0.0	0:00.00	/usr/sbin/sshd -D
1590	root	20	0	342M	764	456	S	0.0	0.1	0:00.02	/usr/sbin/lightdm
1670	root	20	0	222M	440	440	S	0.0	0.0	0:00.00	lightdm --session-child 12 15
1679	lera2003	20	0	48224	2328	1468	S	0.0	0.2	0:00.20	/sbin/upstart --user
2591	lera2003	20	0	650M	26704	15824	S	2.7	2.6	0:32.12	/usr/lib/gnome-terminal/gnome-terminal-server
2598	lera2003	20	0	24196	1988	260	S	0.0	0.2	0:00.04	bash
2637	lera2003	20	0	14056	468	260	S	0.0	0.0	0:00.00	/bin/bash ./write.sh
2639	lera2003	20	0	23724	224	44	S	0.0	0.0	0:00.75	script -f session2.log
2640	lera2003	20	0	24092	4572	2748	S	0.0	0.5	0:00.20	bash -i
3873	lera2003	20	0	28452	4760	3220	R	0.7	0.5	0:00.44	htop
3276	lera2003	20	0	4220	604	536	R	102.	0.1	1h24:26	./nohup1
2594	lera2003	20	0	650M	26704	15824	S	0.8	2.6	0:02:23	gdbus
2593	lera2003	20	0	650M	26704	15824	S	0.8	2.6	0:00.00	gdbus



Как видно, сначала работало два потока, далее один закончил свою работу и, наконец, второй закончил работу, и программа завершилась. Попробуем завершить один из потоков, послав ему SIGTERM.

Command	PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
0 Cancel	1623	root	20	0	17452	0	0	S	0.0	0.0	0:00.00	/sbin/agetty --noclear tty1 linux
1 SIGHUP	1601	root	20	0	65512	0	0	S	0.0	0.0	0:00.00	/usr/sbin/sshd -D
2 SIGINT	1590	root	20	0	342M	764	456	S	0.0	0.1	0:00.02	/usr/sbin/lightdm
3 SIGQUIT	1670	root	20	0	222M	440	440	S	0.0	0.0	0:00.00	lightdm --session-child 12 15
4 SIGILL	1679	lera2003	20	0	48224	2328	1468	S	0.0	0.2	0:00.21	/sbin/upstart --user
5 SIGTRAP	2591	lera2003	20	0	650M	26704	15824	S	1.6	2.6	0:32.70	/usr/lib/gnome-terminal/gnome-terminal-server
6 SIGABRT	2598	lera2003	20	0	24196	1988	260	S	0.0	0.2	0:00.04	bash
7 SIGTSTP	2637	lera2003	20	0	14056	468	260	S	0.0	0.0	0:00.00	/bin/bash ./write.sh
8 SIGFPE	2640	lera2003	20	0	24092	4572	2748	S	0.0	0.5	0:00.20	script -f session2.log
9 SIGKILL	3905	lera2003	20	0	28452	4704	3168	R	1.6	0.5	0:00.21	bash -t
10 SIGUSR1	3895	lera2003	20	0	22912	712	632	S	0.0	0.1	0:00.00	htop
11 SIGSEGV	3899	lera2003	20	0	22912	712	632	S	0.0	0.1	0:00.00	./thread1
12 SIGUSR2	3898	lera2003	20	0	22912	712	632	S	0.0	0.1	0:00.00	thread1
13 SIGPIPE	3276	lera2003	20	0	4220	604	536	R	122.0	0.1	1h26:22	./thread1
14 SIGALRM	2594	lera2003	20	0	650M	26704	15824	S	0.0	2.6	0:02:26	./nohup1
15 SIGTTOU	2593	lera2003	20	0	650M	26704	15824	S	0.0	2.6	0:00.00	gdbus
16 SIGSTKFLT	2592	lera2003	20	0	650M	26704	15824	S	0.0	2.6	0:00.00	gmain
17 SIGCHLD	2574	lera2003	20	0	354M	808	48	S	0.0	0.1	0:00.02	dconf worker
18 SIGCONT	2576	lera2003	20	0	354M	808	48	S	0.0	0.1	0:00.00	/usr/lib/gvfs/gvfsd-dnssd --spawner :1.3 /org/gtk/gvfs/exec_spaw/7
19 SIGSTOP	2575	lera2003	20	0	354M	808	48	S	0.0	0.1	0:00.00	gdbus
20 SIGTSTP	2520	lera2003	20	0	346M	864	152	S	0.0	0.1	0:00.01	gmain
21 SIGTSTP	2520	lera2003	20	0	346M	864	152	S	0.0	0.1	0:00.00	/usr/lib/gvfs/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec_spaw/2

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1670	root	20	0	222M	440	440	S	0.0	0.0	0:00.00	lightdm --session-child 12 15
1679	lera2003	20	0	48224	2328	1468	S	0.0	0.2	0:00.21	/sbin/upstart --user
2591	lera2003	20	0	650M	26704	15824	S	2.3	2.6	0:33.16	/usr/lib/gnome-terminal/gnome-terminal-server
2598	lera2003	20	0	24196	1988	260	S	0.0	0.2	0:00.04	bash
2637	lera2003	20	0	14056	468	260	S	0.0	0.0	0:00.00	/bin/bash ./write.sh
2639	lera2003	20	0	23724	224	44	S	0.0	0.0	0:00.76	script -f session2.log
2640	lera2003	20	0	24092	4572	2748	S	0.0	0.5	0:00.20	bash -t
3905	lera2003	20	0	28452	4704	3168	R	0.8	0.5	0:00.53	htop
3276	lera2003	20	0	4220	604	536	R	116.0	0.1	1h27:04	./nohup1
2594	lera2003	20	0	650M	26704	15824	S	0.0	2.6	0:02:27	gdbus

Как видно, после послания одному потоку SIGTERM завершилась вся программа, что подтверждает информацию из теоретических сведений.

## 9. Модифицируйте программу так, чтобы управление второй нитью осуществлялось посредством сигнала SIGUSR1 из первой нити.

Для этого напишем программу так: в main() передадим управление сигналом SIGUSR1 в функцию sigusr1\_handler (внутри неё выведется сообщение об успешном принятии сигнала и завершится поток 2). Также внутри головной функции вызовем поток 1. В потоке 1 создаём поток 2, ждём 5 секунд и посылаем сигнал SIGUSR1, в результате чего поток завершает свою работу.

*thread2.c*

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

void *thread_function1(void *arg);
void *thread_function2(void *arg);
void sigusr1_handler(int signal_number);
pthread_t thread2;
```

```

int main(int argc, char *argv[])
{
    pthread_t thread1;
    int result1;
    // Установка обработчика сигнала SIGUSR1
    signal(SIGUSR1, sigusr1_handler);
    // Создание первого потока
    result1 = pthread_create(&thread1, NULL, thread_function1, NULL);
    // Ожидание завершения первого потока
    pthread_join(thread1, NULL);
    printf("Главный поток завершен\n");
    exit(EXIT_SUCCESS);
}

void *thread_function1(void *arg)
{
    int i;
    printf("Поток 1 создан\n");
    // Создание второго потока
    pthread_create(&thread2, NULL, thread_function2, NULL);
    // Ждем 5 секунд и затем останавливаем второй поток
    sleep(5);
    // thread2_running = 0;
    pthread_kill(thread2, SIGUSR1);
    // Ожидание завершения второго потока
    pthread_join(thread2, NULL);
    printf("Поток 1 завершен\n");
    return NULL;
}

void *thread_function2(void *arg)
{
    int i = 0;
    printf("Поток 2 создан\n");
    while (1) {
        sleep(1); // Засыпаем на 1 секунду
        printf("Поток 2: %d секундных интервалов\n", ++i);
    }
    return NULL;
}

void sigusr1_handler(int signal_number)
{
    printf("Принят сигнал SIGUSR1.\nПоток 2 завершён\n");
    pthread_cancel(thread2);
}

```

```

lera2003@Valeriya:~/OS_lab34/lb4$ ./thread2
Поток 1 создан
Поток 2 создан
Поток 2: 1 секундных интервалов
Поток 2: 2 секундных интервалов
Поток 2: 3 секундных интервалов
Поток 2: 4 секундных интервалов
Принят сигнал SIGUSR1.
Поток 2 завершён
Поток 1 завершён
Главный поток завершён

```

Как и ожидалось, поток 2 завершён из потока 1 в течение 5 секундных интервалов.

**10. Последняя модификация предполагает создание собственного обработчика сигнала, содержащего уведомление о начале его работы и возврат посредством функции `pthread_exit(NULL)`.**

*thread3.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <signal.h>

void *thread_function1(void *arg);
void *thread_function2(void *arg);
void sigusr1_handler(int signal_number);
pthread_t thread2;

int main(int argc, char *argv[])
{
    pthread_t thread1;
    int result1;
    // Установка обработчика сигнала SIGUSR1
    signal(SIGUSR1, sigusr1_handler);
    // Создание первого потока
    result1 = pthread_create(&thread1, NULL, thread_function1, NULL);
    // Ожидание завершения первого потока
    pthread_join(thread1, NULL);
    printf("Главный поток завершён\n");
    exit(EXIT_SUCCESS);
}

void *thread_function1(void *arg)
{

```

```

int i;
printf("Поток 1 создан\n");
// Создание второго потока
pthread_create(&thread2, NULL, thread_function2, NULL);
// Ждем 5 секунд и затем останавливаем второй поток
sleep(5);
// thread2_running = 0;
pthread_kill(thread2, SIGUSR1);
// Ожидание завершения второго потока
pthread_join(thread2, NULL);
printf("Поток 1 завершен\n");
return NULL;
}

void *thread_function2(void *arg)
{
    int i = 0;
    printf("Поток 2 создан\n");
    while (1) {
        sleep(1); // Засыпаем на 1 секунду
        printf("Поток 2: %d секундных интервалов\n", ++i);
    }
    return NULL;
}

void sigusr1_handler(int signal_number)
{
    printf("Принят сигнал SIGUSR1.\nПоток 2 завершён\n");
    //pthread_cancel(thread2);
    pthread_exit(NULL);
}

```

```

lera2003@valeriya:~/OS_lab34/lb4/10$ cc -pthread thread3.c -o thread3
lera2003@Valeriya:~/OS_lab34/lb4/10$ ./thread3
Поток 1 создан
Поток 2 создан
Поток 2: 1 секундных интервалов
Поток 2: 2 секундных интервалов
Поток 2: 3 секундных интервалов
Поток 2: 4 секундных интервалов
Принят сигнал SIGUSR1.
Поток 2 завершён
Поток 1 завершен
Главный поток завершен

```

Хотя результат работы совпал, между данными функциями есть некоторая разница, а именно то, в какой момент функции завершают поток. Pthread\_exit() позволяет потоку выполнить все предварительные или

завершающие действия для корректной остановки, в то время как `pthread_cancel()` может прервать поток в любой точке его выполнения.

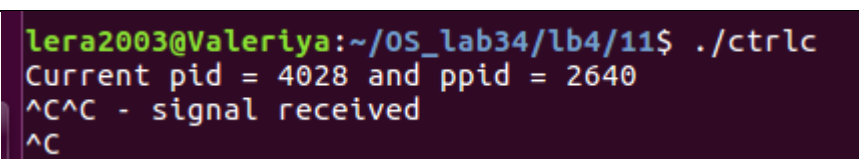
Именно поэтому вывод может различаться (т. е. `pthread_exit()` немного дольше работает).

**11. Перехватите сигнал «CTRL C» для процесса и потока однократно, а также многократно с восстановлением исходного обработчика после нескольких раз срабатывания. Прodelайте аналогичную работу для переназначения другой комбинации клавиш.**

Перехватим сигнал `Ctrl C` для процесса и потока однократно, а также многократно с восстановлением исходного обработчика после нескольких раз срабатывания.

*ctrlc.c*

```
#include <stdio.h>
#include <signal.h>
void handler()
{
    puts("^C - signal received");
    signal(SIGINT, SIG_DFL); //восстановление диспозиции по умолчанию
}
int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("Current pid = %d and ppid = %d\n", pid, ppid);
    signal(SIGINT, handler);
    while(1);
    return 0;
}
```



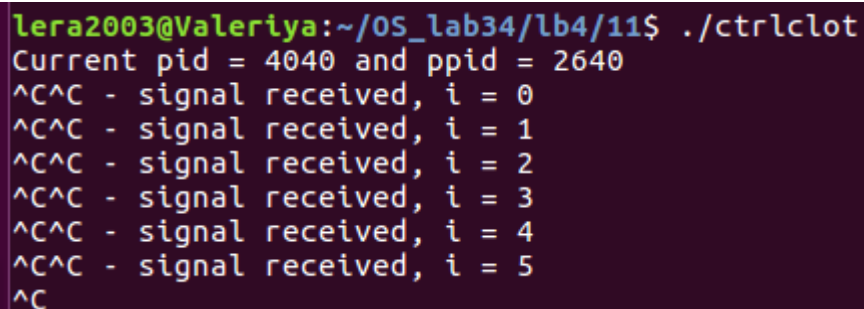
```
lera2003@Valeriya:~/OS_lab34/lb4/11$ ./ctrlc
Current pid = 4028 and ppid = 2640
^C^C - signal received
^C
```

Сигнал `^C` перехватился и однократно вызвался обработчик `handler`, который вывел строку, оповещающую о получении сигнала, после чего

обработчик SIGINT возвращается по умолчанию, результат которого – принудительное завершение программы. Скорректируем программу для многократного нажатия.

*ctrlclot.c*

```
#include <stdio.h>
#include <signal.h>
void handler()
{
    static int i = 0;
    printf("^C - signal received, i = %d\n", i);
    if (i++ == 5) //количество срабатываний текущего обработчика
        signal(SIGINT, SIG_DFL); //восстановление стандартного
//обработчика
}
int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("Current pid = %d and ppid = %d\n", pid, ppid);
    signal(SIGINT, handler);
    while(1);
    return 0;
}
```



```
lera2003@Valeriya:~/OS_lab34/lb4/11$ ./ctrlclot
Current pid = 4040 and ppid = 2640
^C^C - signal received, i = 0
^C^C - signal received, i = 1
^C^C - signal received, i = 2
^C^C - signal received, i = 3
^C^C - signal received, i = 4
^C^C - signal received, i = 5
^C
```

Прделаем аналогичную работу для другой комбинации клавиш (Ctrl + Z).

*ctrlz.c*

```
#include <stdio.h>
#include <signal.h>
void handler()
{
    static int i = 0;
```

```

    printf("^Z - signal received, i = %d\n", i);
    if (i++ == 5) //количество срабатываний текущего обработчика
        signal(SIGTSTP, SIG_DFL); //восстановление стандартного
//обработчика
}
int main()
{
    int pid, ppid;
    pid = getpid();
    ppid = getppid();
    printf("Current pid = %d and ppid = %d\n", pid, ppid);
    signal(SIGTSTP, handler);
    while(1);
    return 0;
}

```

```

lera2003@Valeriya:~/OS_lab34/lb4/11$ ./ctrlz
Current pid = 4075 and ppid = 2640
^Z^Z - signal received, i = 0
^Z^Z - signal received, i = 1
^Z^Z - signal received, i = 2
^Z^Z - signal received, i = 3
^Z^Z - signal received, i = 4
^Z^Z - signal received, i = 5

```

**12. С помощью утилиты kill выведите список всех сигналов и дайте их краткую характеристику на основе документации ОС. Для чего предназначены сигналы с 32 по 64-й. Приведите пример их применения.**

С помощью утилиты kill выведем список сигналов и дадим краткую характеристику.

```

lera2003@Valeriya:~/OS_lab34/lb4/12$ 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
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX

```

Команда выводит на экран список всех сигналов, которые могут быть отправлены процессу с помощью команды kill или других инструментов управления процессами. Процессы от 1 до 31 заданы стандартом POSIX и

выполняют определенные функции. Например, завершение процесса (SIGTERM), неожиданное прерывание процесса (SIGINT), остановка процесса (SIGSTOP). Сигналы 32 и 33 не отображаются, так как они заняты реализацией POSIX-ядер. Сигналы с номерами от 34 и выше – переменные. SIGRTMIN и SIGRTMAX используются как минимальное и максимальное значение для этих сигналов (эти значения могут быть изменены), которые могут быть определены динамически в программах. Из мануала, эти сигналы называются сигналами реального времени. Их предназначение не обязательно определено, как говорилось ранее, они могут быть использованы при разработке. По умолчанию сигналы от 34 и выше завершают работу программы. Приведём пример программы, использующей сигналы от 34 до 40. Результат работы программы и устройство на рисунке 38.

*signals\_time.c*

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

void sig_handler(int sig) {
    printf("Signal %d ignored.\n", sig);
}

int main() {

    for (int i = 34; i <= 40; i++) {
        signal(i, sig_handler);
    }

    while (1) {
        sleep(1);
    }

    return 0;
}
```



```

[3] 4109
lera2003@Valeriya:~/OS_lab34/lb4/12$ ./signals_time &
[3] 4109
lera2003@Valeriya:~/OS_lab34/lb4/12$ ps
  PID TTY          TIME CMD
 2640 pts/11        00:00:00 bash
 3276 pts/11        01:56:24 nohup1
 4075 pts/11        00:00:04 ctrlz
 4109 pts/11        00:00:00 signals_time
 4110 pts/11        00:00:00 ps
lera2003@Valeriya:~/OS_lab34/lb4/12$ kill -37 4109
Signal 37 ignored.
lera2003@Valeriya:~/OS_lab34/lb4/12$ kill -40 4109
Signal 40 ignored.
lera2003@Valeriya:~/OS_lab34/lb4/12$ kill -34 4109
Signal 34 ignored.
lera2003@Valeriya:~/OS_lab34/lb4/12$ kill -41 4109
lera2003@Valeriya:~/OS_lab34/lb4/12$ ps
  PID TTY          TIME CMD
 2640 pts/11        00:00:00 bash
 3276 pts/11        01:56:47 nohup1
 4075 pts/11        00:00:04 ctrlz
 4113 pts/11        00:00:00 ps
[3]-  Сигнал реального времени 7                               ./signals_time

```

Как видно, при посылке сигналов от 34 до 40 сигналы игнорируются нашим обработчиком, а при посылке сигнала 41 программа завершается (т.к. по умолчанию, как и ожидалось, сигнал 41 завершает работу).

### 13. Проанализируйте процедуру планирования для процессов и потоков одного процесса.

Исследуем борьбу за ресурс для процессов с одинаковой и разной политикой планирования, внутри которых будут запущены циклы.

*RR.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>

#define PROC_AMOUNT 10

int main() {
    int i;
    char* program = "son1";
    struct sched_param params;
    params.sched_priority = PROC_AMOUNT;
    int sched;
    for (i = 0; i < PROC_AMOUNT; i++) {
        pid_t pid = fork();
    }
}

```

```

        if (pid < 0) {
            printf("Ошибка при создании дочернего процесса\n");
            return 1;
        }
        else if (pid == 0) {
            sched = SCHED_RR;
            if (sched_setscheduler(0, sched, &params) < 0) {
                printf("Ошибка при установке политики планирования\n");
                return 1;
            }
            execl(program, program, NULL);
            printf("Ошибка при вызове функции execl\n");
            return 1;
        }
    }
    for (int i = 0; i < PROC_AMOUNT; i++){
        wait(NULL);
    }
    return 0;
}

```

*FIFO.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>

#define PROC_AMOUNT 10

int main() {
    int i;
    char* program = "son1";
    struct sched_param params;
    params.sched_priority = PROC_AMOUNT;
    int sched;
    for (i = 0; i < PROC_AMOUNT; i++) {
        pid_t pid = fork();
        if (pid < 0) {
            printf("Ошибка при создании дочернего процесса\n");
            return 1;
        }
        else if (pid == 0) {
            sched = SCHED_FIFO;
            if (sched_setscheduler(0, sched, &params) < 0) {
                printf("Ошибка при установке политики планирования\n");
                return 1;
            }
            execl(program, program, NULL);
            printf("Ошибка при вызове функции execl\n");
        }
    }
}

```

```

        return 1;
    }
}
for (int i = 0; i<PROC_AMOUNT;i++){
    wait(NULL);
}
return 0;
}

```

### *RRFIFO.c*

```

#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>

#define PROC_AMOUNT 10

int main() {
    int i;
    char* program = "son1";
    struct sched_param params;
    params.sched_priority = PROC_AMOUNT;
    int sched;
    for (i = 0; i < PROC_AMOUNT; i++) {
        pid_t pid = fork();
        if (pid < 0) {
            printf("Ошибка при создании дочернего процесса\n");
            return 1;
        }
        else if (pid == 0) {
            if (i % 2 == 0){
                sched = SCHED_FIFO;
            }
            else{
                sched = SCHED_RR;
            }
            if (sched_setscheduler(0, sched, &params) < 0) {
                printf("Ошибка при установке политики планирования\n");
                return 1;
            }
            execl(program, program, NULL);
            printf("Ошибка при вызове функции execl\n");
            return 1;
        }
    }
    for (int i = 0; i<PROC_AMOUNT;i++){
        wait(NULL);
    }
    return 0;
}

```

```
}
```

*son1.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <time.h>
#include <unistd.h>
int main()
{
    struct sched_param shdprm; // Значения параметров планирования
    int pid;
    pid = getpid();
    int k=0;
    int schedule = sched_getscheduler(0);
    switch (schedule)
    {
        case SCHED_FIFO:
            for (int i =0;i<100000000;i++){
                k+=1;
                printf("%iF\n", pid);
            }
            break;
        case SCHED_RR:
            for (int i =0;i<100000000;i++){
                k+=1;
                printf("%iR\n", pid);
            }
            break;
        case SCHED_OTHER:
            printf("SCHED_OTHER\n");
            break;
        case -1:
            perror("SCHED_GETSCHEDULER");
            break;
        default:
            printf("Неизвестная политика планирования\n");
    }
    return 0;
}
```

Результат работ программ следующий:

[illegible]

[illegible]



атрибутов потока управления командой `pthread_attr_init`. Иначе получим `SEGFAULT`. Поменяем стратегию планирования потоков. Первому потоку зададим стратегию планирования `SCHED_RR`, а второму – `SCHED_FIFO`. Самому же процессу зададим стратегию планирования `FIFO`.

### *1\_thread\_plan.c*

```
#include <signal.h>
#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sched.h>
#include <unistd.h>

pthread_t t1, t2;
void *thread1();
void *thread2();
void switch_policy(int policy);

void main()
{
    int policy1;
    int policy2;
    struct sched_param param;
    struct sched_param shdprm;
    pid_t pid = getpid();
    pthread_attr_t attr_1, attr_2;
    param.sched_priority = 5;
    shdprm.sched_priority = 10;
    if (sched_setscheduler(0, SCHED_FIFO, &shdprm) == -1)
    {
        perror("SCHED_SETSCHEDULER");
    }

    printf("Политика процесса: ");
    switch_policy(sched_getscheduler(pid));
    printf("Приоритет процесса: %d\n", shdprm.sched_priority);

    pthread_attr_init(&attr_1);
    pthread_attr_init(&attr_2);
    pthread_attr_setschedparam(&attr_1, &shdprm);
    pthread_attr_setschedparam(&attr_2, &param);
    pthread_attr_setschedpolicy(&attr_1, SCHED_RR);
    pthread_attr_setschedpolicy(&attr_2, SCHED_FIFO);
```



```

pthread_attr_getschedparam(&attr_1, &shdprm);
pthread_attr_getschedpolicy(&attr_1, &policy1);

pthread_attr_getschedparam(&attr_2, &param);
pthread_attr_getschedpolicy(&attr_2, &policy2);

printf("Thread 1 policy: ");
switch_policy(policy1);
printf("Thread 2 policy: ");
switch_policy(policy2);
pthread_create(&t1, &attr_1, thread1, NULL);
pthread_create(&t2, &attr_2, thread2, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
pthread_attr_destroy(&attr_1);
pthread_attr_destroy(&attr_2);
}

void switch_policy(int policy)
{
    switch (policy)
    {
        case SCHED_FIFO:
            printf("policy SCHED_FIFO\n");
            break;
        case SCHED_RR:
            printf("policy SCHED_RR\n");
            break;
        case SCHED_OTHER:
            printf("policy SCHED_OTHER\n");
            break;
        case -1:
            perror("policy SCHED_GETSCHEDULER");
            break;
        default:
            printf ("policy Неизвестная политика планирования\n");
    }
}

void *thread2()
{
    int i, count = 0;
    int tid, pid;
    tid = syscall(SYS_gettid);
    pid = getpid();
    printf("Thread_2 with thread id = %d and pid = %d is started\n", tid,
pid);
    for (i = 0; i < 10; i++)
    {

```

```

        count += 1;
    }
}
void *thread1()
{
    int i, count = 0;
    int tid, pid;
    tid = syscall(SYS_gettid);
    pid = getpid();
    printf("Thread_1 with thread id = %d and pid = %d is started\n", tid,
pid);
    for (i = 0; i < 10; i++)
    {
        count += 1;
    }
}

```

```

Thread_2 with thread id = 4448 and pid = 4448 is started
lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./1_thread_plan
Политика процесса: policy SCHED_FIFO
Приоритет процесса: 10
Thread 1 policy: policy SCHED_RR
Thread 2 policy: policy SCHED_FIFO
Thread_1 with thread id = 4451 and pid = 4450 is started
Thread_2 with thread id = 4452 and pid = 4450 is started

```

Как видно, процессу 1 задана процедура планирования RR, а процессу 2 – FIFO. Разница между `pthread_attr_setschedpolicy()` и `sched_setscheduler()` в том, что первая команда используется для установки стратегии планирования новому потоку, а вторая – для установки планирования уже существующему потоку или процессу.

Модифицируем программы так, чтобы создавалось несколько потоков, и этим потокам выдавались как одинаковые, так и разные политики планирования. Внутри потоков вставим некоторые вычисления. Рассмотрим содержание программы.

### RR.c

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <errno.h>

long int OPS_AMOUNT = 1000000000;

```

```

void* func_fifo(void* arg) {
    int marker = *((int*) arg);
    int dummy = 0;
    for (int i = 0; i < 100; i++) {
        dummy++;
        printf("%dF\n", marker);
    }
    //printf("Thread with SCHED_FIFO END\n");
    pthread_exit(NULL);
}

void* func_rr(void* arg) {
    int dummy = 0;
    int marker = *((int*) arg);
    for (int i = 0; i < OPS_AMOUNT; i++) {
        dummy++;
        printf("%dR\n", marker);
    }
    //printf("Thread with SCHED_RR END\n");
    pthread_exit(NULL);
}

int main() {
    const int NUM_THREADS = 10;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attrs[NUM_THREADS];
    struct sched_param params[NUM_THREADS];
    int priorityFIFO = 50;
    int priorityRR = 50;
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_attr_init(&attrs[i]);
        pthread_attr_setinheritsched(&attrs[i], PTHREAD_EXPLICIT_SCHED);
        params[i].sched_priority = priorityRR;
        pthread_attr_setschedpolicy(&attrs[i], SCHED_RR);
        pthread_attr_setschedparam(&attrs[i], &params[i]);
        int err = pthread_create(&threads[i], &attrs[i], func_rr, &i);
        if (err != 0) {
            printf("Error creating thread %d", i);
        }
    }
    puts("");
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("All threads finished\n");
    return 0;
}

```

## *FIFO.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <errno.h>

void* func_fifo(void* arg) {
    int marker = *((int*) arg);
    int dummy = 0;
    for (int i = 0; i < 1000000000; i++) {
        dummy++;
        printf("%dF\n", marker);
    }
    //printf("Thread with SCHED_FIFO END\n");
    pthread_exit(NULL);
}

void* func_rr(void* arg) {
    int dummy = 0;
    int marker = *((int*) arg);
    for (int i = 0; i < 1000000000; i++) {
        dummy++;
        printf("%dR\n", marker);
    }
    //printf("Thread with SCHED_RR END\n");
    pthread_exit(NULL);
}

int main() {
    const int NUM_THREADS = 10;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attrs[NUM_THREADS];
    struct sched_param params[NUM_THREADS];
    int priorityFIFO = 50;
    int priorityRR = 50;
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_attr_init(&attrs[i]);
        pthread_attr_setinheritsched(&attrs[i], PTHREAD_EXPLICIT_SCHED);
        params[i].sched_priority = priorityRR;
        pthread_attr_setschedpolicy(&attrs[i], SCHED_FIFO);
        pthread_attr_setschedparam(&attrs[i], &params[i]);
        int err = pthread_create(&threads[i], &attrs[i], func_fifo, &i);
        if (err != 0) {
            printf("Error creating thread %d", i);
        }
    }
    puts("");
}
```

```
for (int i = 0; i < NUM_THREADS; i++) {  
    pthread_join(threads[i], NULL);  
}  
printf("All threads finished\n");  
return 0;  
}
```

```
1R  
8R  
8R  
8R  
8R  
8R  
8R  
8R  
1R  
1R  
1R  
1R  
1R  
1R  
1R  
1R  
7R  
3R  
3R  
3R  
3R  
3R  
3R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
7R  
4R  
1R  
6R  
6R  
6R  
6R  
6R  
3R  
3R  
3R  
3R  
3R  
3R  
8R
```



Можно сделать вывод, что борьба за ресурс у нитей происходит более равномерно.

Модифицируем программу. Теперь уменьшим количество подсчетов для политики RR.

*RRsmall.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <errno.h>

long long int OPS_AMOUNT = 1000000000;

void* func_fifo(void* arg) {
    int marker = *((int*) arg);
    int dummy = 0;
    for (int i = 0; i < OPS_AMOUNT; i++) {
        dummy++;
    }
}
```

```

    }
    printf("Thread with SCHED_FIFO END: %d\n", marker);
    pthread_exit(NULL);
}

void* func_rr(void* arg) {
    int marker = *((int*) arg);
    int dummy = 0;
    for (int i = 0; i < 10; i++) {
        dummy++;
    }
    printf("Thread with SCHED_RR END: %d\n", marker);
    pthread_exit(NULL);
}

int main() {
    const int NUM_THREADS = 5;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attrs[NUM_THREADS];
    struct sched_param params[NUM_THREADS];
    int priorityFIFO = 50;
    int priorityRR = 50;
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_attr_init(&attrs[i]);
        pthread_attr_setinheritsched(&attrs[i], PTHREAD_EXPLICIT_SCHED);
        if (i % 2 == 0)
            params[i].sched_priority = priorityRR;
        else
            params[i].sched_priority = priorityFIFO;
        // pthread_attr_setschedpolicy(&attrs[i], (i % 2) ? SCHED_RR :
        SCHED_FIFO);
        pthread_attr_setschedpolicy(&attrs[i], SCHED_RR);
        pthread_attr_setschedparam(&attrs[i], &params[i]);
        // int err = pthread_create(&threads[i], &attrs[i], (i % 2) ? func_rr
        : func_fifo, NULL);
        int err = pthread_create(&threads[i], &attrs[i], func_rr, &i);
        if (err != 0) {
            printf("Error creating thread %d", i);
        }
    }
    // puts("");
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("All threads finished\n");

    return 0;
}

```

```
}
```

Результат работы программы с большим количеством вычислений.

```
lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./plan_sched_thread_RR
Thread with SCHED_RR END: 1
Thread with SCHED_RR END: 2
Thread with SCHED_RR END: 1
Thread with SCHED_RR END: 5
Thread with SCHED_RR END: 5
All threads finished
```

Результат работы программы с малым количеством вычислений.

```
lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./plan_sched_thread_RR
Thread with SCHED_RR END: 1
Thread with SCHED_RR END: 1
Thread with SCHED_RR END: 2
Thread with SCHED_RR END: 3
Thread with SCHED_RR END: 4
All threads finished
```

Как видно, при большом количестве вычислений величина кванта превышаетя, потоки передают управление друг другу, следовательно, порядок завершения произволен. При малом количестве вычислений потоки завершаются друг за другом.

Рассмотрим команду `pthread_attr_setinheritsched()` сначала с `PTHREAD_INHERIT_SCHED` (т. е. наследуемся от родительского), а потом – `PTHREAD_EXPLICIT_SCHED` (устанавливаем явно отдельно).

*dummy.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int i = 0;

void switch_policy(int policy)
{
    switch (policy)
    {
        case SCHED_FIFO:
            printf("policy SCHED_FIFO\n");
            break;
        case SCHED_RR:
            printf("policy SCHED_RR\n");
            break;
        case SCHED_OTHER:
            break;
```



```

        printf("policy SCHED_OTHER\n");
        break;
    case -1:
        perror("policy SCHED_GETSCHEDULER");
        break;
    default:
        printf ("policy Неизвестная политика планирования\n");
    }
}

void *thread_func(void *arg) {
    printf("Child thread started.\n");
    /* Do some work here */
    printf("Thread's ");
    switch_policy(sched_getscheduler(0));
    while (i<10){
        i++;
    }
    printf("Child thread finished.\n");
    pthread_exit(NULL);
}

int main() {
    pthread_t child_thread;
    pthread_attr_t attr;
    struct sched_param shdprm;
    int ret;

    pid_t pid = getpid();
    shdprm.sched_priority = 1;

    if (sched_setscheduler(pid, SCHED_FIFO, &shdprm) == -1)
    {
        perror("SCHED_SETSCHEDULER");
    }

    printf("Политика процесса: ");
    switch_policy(sched_getscheduler(pid));

    /* Initialize thread attributes */
    ret = pthread_attr_init(&attr);
    if (ret != 0) {
        fprintf(stderr, "pthread_attr_init() failed.\n");
        exit(EXIT_FAILURE);
    }

    /* Set thread scheduling to inherit from parent */
    //ret = pthread_attr_setinheritsched(&attr, PTHREAD_INHERIT_SCHED);
    ret = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);

```

```

if (ret != 0) {
    fprintf(stderr, "pthread_attr_setinheritsched() failed.\n");
    exit(EXIT_FAILURE);
}

/* Create child thread with inherited scheduling */
ret = pthread_create(&child_thread, &attr, thread_func, NULL);
if (ret != 0) {
    fprintf(stderr, "pthread_create() failed.\n");
    exit(EXIT_FAILURE);
}

int thread_policy;
pthread_attr_getschedpolicy(&attr, &thread_policy);

/* Wait for child thread to finish */
pthread_join(child_thread, NULL);

/* Cleanup thread attributes */
pthread_attr_destroy(&attr);

printf("Parent thread exiting.\n");
return 0;
}

```

Результат с INHERIT.

```

lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy
Политика процесса: policy SCHED_FIFO
Child thread started.
Thread's policy SCHED_FIFO
Child thread finished.
Parent thread exiting.

```

Результат с EXPLICIT.

```

lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy
Политика процесса: policy SCHED_FIFO
Child thread started.
Thread's policy SCHED_OTHER
Child thread finished.
Parent thread exiting.

```

Как видно, если применять EXPLICIT политика потока — SCHED\_OTHER, поскольку не была задана явно. Если же INHERIT — политика наследуется от политики процесса (который его породил).

Теперь не будем задавать процессу процедуру планирования. Зададим её потоку 1, который, в свою очередь, породит поток 2. Рассмотрим наследование при таком варианте.

*dummy1.c*

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

void *thread_function1(void *arg);
void *thread_function2(void *arg);
void sigusr1_handler(int signal_number);
pthread_t thread2;

void switch_policy(int policy)
{
    switch (policy)
    {
        case SCHED_FIFO:
            printf("policy SCHED_FIFO\n");
            break;
        case SCHED_RR:
            printf("policy SCHED_RR\n");
            break;
        case SCHED_OTHER:
            printf("policy SCHED_OTHER\n");
            break;
        case -1:
            perror("policy SCHED_GETSCHEDULER");
            break;
        default:
            printf("policy Неизвестная политика планирования\n");
    }
}

int main(int argc, char *argv[])
{
    pthread_t thread1;
    int result1;
    pthread_attr_t attr1;

    // Установка обработчика сигнала SIGUSR1
    signal(SIGUSR1, sigusr1_handler);

    pthread_attr_init(&attr1);
```

```

pthread_attr_setinheritsched(&attr1, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr1, SCHED_FIFO);
// Установка параметров планирования для первого потока
struct sched_param shdprm;
shdprm.sched_priority = 1;
pthread_attr_setschedparam(&attr1, &shdprm);

// Создание первого потока
result1 = pthread_create(&thread1, &attr1, thread_function1, NULL);

// Ожидание завершения первого потока
pthread_join(thread1, NULL);
printf("Главный поток завершен\n");
exit(EXIT_SUCCESS);
}

void *thread_function1(void *arg)
{

    printf("Поток 1 создан\n");
    printf("Политика потока 1: ");
    switch_policy(sched_getscheduler(0));
    //printf("Приоритет потока 1: %d", );

    pthread_attr_t attr2;

    pthread_attr_init(&attr2);
    pthread_attr_setinheritsched(&attr2, PTHREAD_EXPLICIT_SCHED);
    //pthread_attr_setinheritsched(&attr2, PTHREAD_INHERIT_SCHED);

    pthread_attr_setschedpolicy(&attr2, SCHED_RR);

    // Установка параметров планирования для второго потока
    struct sched_param param;
    param.sched_priority = 2;
    pthread_attr_setschedparam(&attr2, &param);

    pthread_create(&thread2, &attr2, thread_function2, NULL);

    // Ждем 5 секунд и затем останавливаем второй поток
    sleep(5);
    pthread_kill(thread2, SIGUSR1);

    // Ожидание завершения второго потока
    pthread_join(thread2, NULL);
    printf("Поток 1 завершен\n");
}

```

```

    pthread_attr_destroy(&attr2);
    return NULL;
}

void *thread_function2(void *arg)
{
    printf("Поток 2 создан\n");
    printf("Политика потока 2: ");
    switch_policy(sched_getscheduler(0));
    while (1)
    {
        sleep(1);
    }
    return NULL;
}

void sigusr1_handler(int signal_number)
{
    printf("Поток 2 завершен\n");
    pthread_exit(NULL);
}

```

Результат с INHERIT.

```

lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy1
Поток 1 создан
Политика потока 1: policy SCHED_FIFO
Поток 2 создан
Политика потока 2: policy SCHED_FIFO
Поток 2 завершен
Поток 1 завершен
Главный поток завершен

```

Результат с EXPLICIT.

```

lera2003@Valeriya:~/OS_lab34/lb4/13$ cc -pthread -o dummy1 dummy1.c
lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy1
Поток 1 создан
Политика потока 1: policy SCHED_FIFO
Поток 2 создан
Политика потока 2: policy SCHED_RR
Поток 2 завершен
Поток 1 завершен
Главный поток завершен

```

Программа работает как ожидалось, поведение наследования от процесса и наследования от потока не различаются.

Напишем программы, задающие приоритет извне и внутри программы для двух разных потоков.

*dummy2.c*

Приоритет извне.

```
#include <pthread.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void *thread_function(void *arg) {
    int prio = *((int*) arg);
    printf("Thread priority: %d\n", prio);
    // sleep(1);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t thread1, thread2;
    pthread_attr_t attr;
    struct sched_param param;

    if (argc < 2) {
        printf("Usage: %s <priority1> <priority2>\n", argv[0]);
        return 1;
    }

    int priority1 = atoi(argv[1]);
    int priority2 = atoi(argv[2]);

    // Set the scheduling policy and priority for the newly created pthread
    pthread_attr_init(&attr);
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
    param.sched_priority = priority1;
    pthread_attr_setschedparam(&attr, &param);

    pthread_attr_init(&attr);
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
    param.sched_priority = priority2;
    pthread_attr_setschedparam(&attr, &param);

    // Create the pthread with the specified SCHED_FIFO policy and priority
```

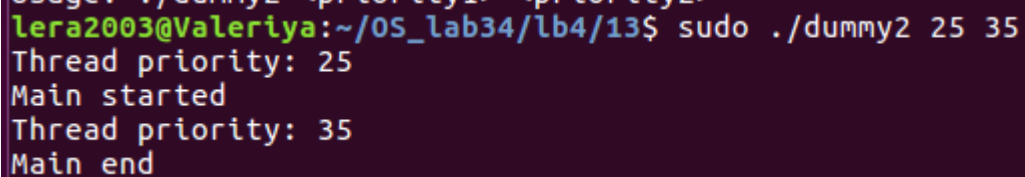
```

if (pthread_create(&thread1, &attr, thread_function, &priority1) != 0) {
    perror("pthread_create");
    return 1;
}

if (pthread_create(&thread2, &attr, thread_function, &priority2) != 0) {
    perror("pthread_create");
    return 1;
}

printf("Main started\n");
// Now join the pthread
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
printf("Main end\n");
return 0;
}

```



```

lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy2 25 35
Thread priority: 25
Main started
Thread priority: 35
Main end

```

### *dummy3.c*

программа, задающая приоритет программно

```

#include <pthread.h>
#include <stdio.h>

void* thread_func(void* arg) {
    int prio = *((int*) arg);
    printf("Thread priority: %d\n", prio);
    return NULL;
}

int main(void) {
    pthread_t thread1, thread2;
    int prio1 = 1, prio2 = 99;

    pthread_create(&thread1, NULL, thread_func, &prio1);
    pthread_create(&thread2, NULL, thread_func, &prio2);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
}

```

```
return 0;  
}
```

```
lera2003@Valeriya:~/OS_lab34/lb4/13$ sudo ./dummy3  
Thread priority: 1  
Thread priority: 99
```

В первой программе задаются приоритеты 25 и 35, а во второй – 1 и 99 для потока 1 и 2 соответственно. Как видно, принципиальных различий в задаче приоритетов нет.

Подытожив, можно сказать, что на данной системе (WSL, Ubuntu) нет ограничений на изменение процедур планирования, приоритета для потоков одного процесса.

**14. Создайте командный файл (скрипт), выполняющий вашу лабораторную работу автоматически при наличии необходимых С-файлов.**

Создадим скрипт, выполняющий нашу работу автоматически. Для этого воспользуемся утилитой Linux script. С помощью команды `script --timing=time_log script_lab` все действия в терминале (с соблюдением временных промежутков) будет записан в `script_lab` (временные промежутки в `time_log`). Можем его воспроизвести с помощью команды `scriptreplay --timing=time_log script_lab` варьируя скорость воспроизведения. Напишем небольшой bash скрипт, воспроизводящий лабораторную работу с задаваемой скоростью.

*lb\_script.sh*

```
#!/bin/bash  
scriptreplay --timing=time_log script_lab $1
```



```

lera2003@Valeriya:~/OS_lab34/lb4$ script --timing=time_log script_lab
Скрипт запущен, файл – script_lab
lera2003@Valeriya:~/OS_lab34/lb4$ ls
1  11 13 3 5 7 9      session2.log  time_log
10 12 2  4 6 8  script_lab session.log  write.sh
lera2003@Valeriya:~/OS_lab34/lb4$ touch lb_script.sh
lera2003@Valeriya:~/OS_lab34/lb4$ nano lb_script.sh
lera2003@Valeriya:~/OS_lab34/lb4$ nano lb_script.sh
lera2003@Valeriya:~/OS_lab34/lb4$ ls
1  11 13 3 5 7 9      script_lab  session.log  write.sh
10 12 2  4 6 8  lb_script.sh session2.log time_log
lera2003@Valeriya:~/OS_lab34/lb4$ nano lb_script.sh

```

### Вывод.

В ходе выполнения лабораторной работы ознакомились с возможностью управлением процессов посредством сигналов, многопоточным функционированием. Рассмотрен перечень сигналов, поддерживаемых процессами. Организованы различные посылки сигналов, их фиксация, обработка, перехватывание, переопределение. Ознакомились с выполнением команды `nice()` и `getpriority()`, `nohup()`. Определили некоторые системные параметры (максимальные, минимальные `uid`, `pid`). Рассмотрено многопоточное программирование (управление потоками из процесса, из другого потока, создание обработчиков сигнала, перехват сигналов). Проанализированы процедуры планирования для процессов и потоков одного процесса. Создан скрипт, выполняющий работу автоматически.

### Список источников.

1. «Системное программное обеспечение. Практические вопросы разработки системных приложений. Учебное пособие» Душутина Е.В.
2. Сайт [fork\(2\) - Справочная страница Linux \(man7.org\)](http://man7.org)
3. Сайт [Создание процессов с помощью вызова fork\(\). \(opennet.ru\)](http://opennet.ru)
4. Сайт [Ubuntu Manpage: Welcome](http://ubuntu-manpage.org)
5. Сайт [https://www.opennet.ru/docs/RUS/linux\\_parallel/node7.html](https://www.opennet.ru/docs/RUS/linux_parallel/node7.html)