**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);  } |

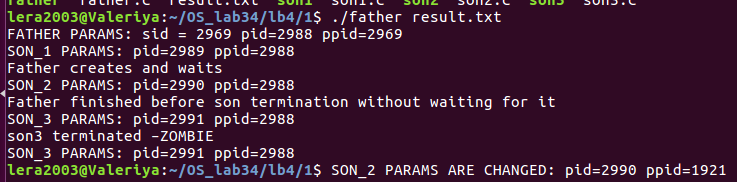
*son2.c (father завершает свое выполнение раньше son2)*

|  |
| --- |
| #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)*

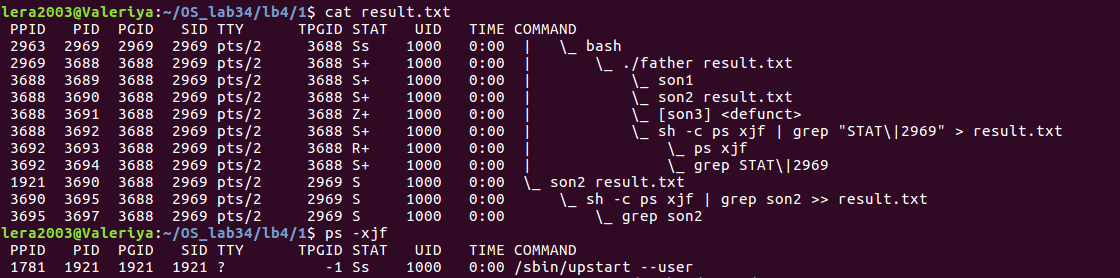
|  |
| --- |
| #include <stdio.h>  void main()  {      int pid, ppid;      pid = getpid();      ppid = getppid();      printf("SON\_3 PARAMS: pid=%i ppid=%i\nson3 terminated –ZOMBIE\n",pid,ppid);      ppid=getppid();      printf("SON\_3 PARAMS: pid=%i ppid=%i\n",pid,ppid);  } |

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



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, появлется уже после приглашения.



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

В выводе зафиксированы: нормальное выполнение потомка son1.

Cмена 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.**

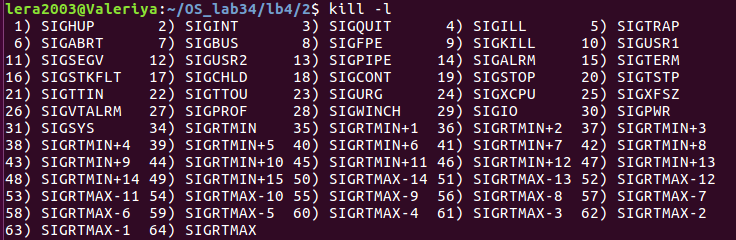
Для передачи сигналов процессам в Linux используется утилита kill. Ее синтаксис очень прост:

$ 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.



Следует заметить, что именование 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 процесc, 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 представленоэкспериментально.

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 sucessfuly\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 sucessfuly\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 sucessfuly\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*

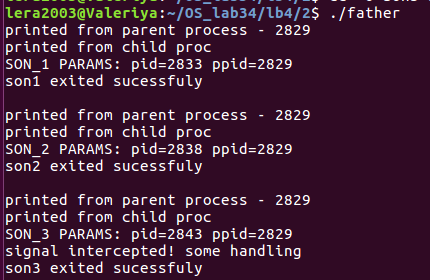
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/wait.h>  #include <signal.h>  int main()  {      int pid, ppid;      pid = getpid();      ppid = getppid();      printf("SON\_1 PARAMS: pid=%i ppid=%i\n", pid, ppid);    //  while (1);      sleep(3);      return 0;  } |

*son2.c*

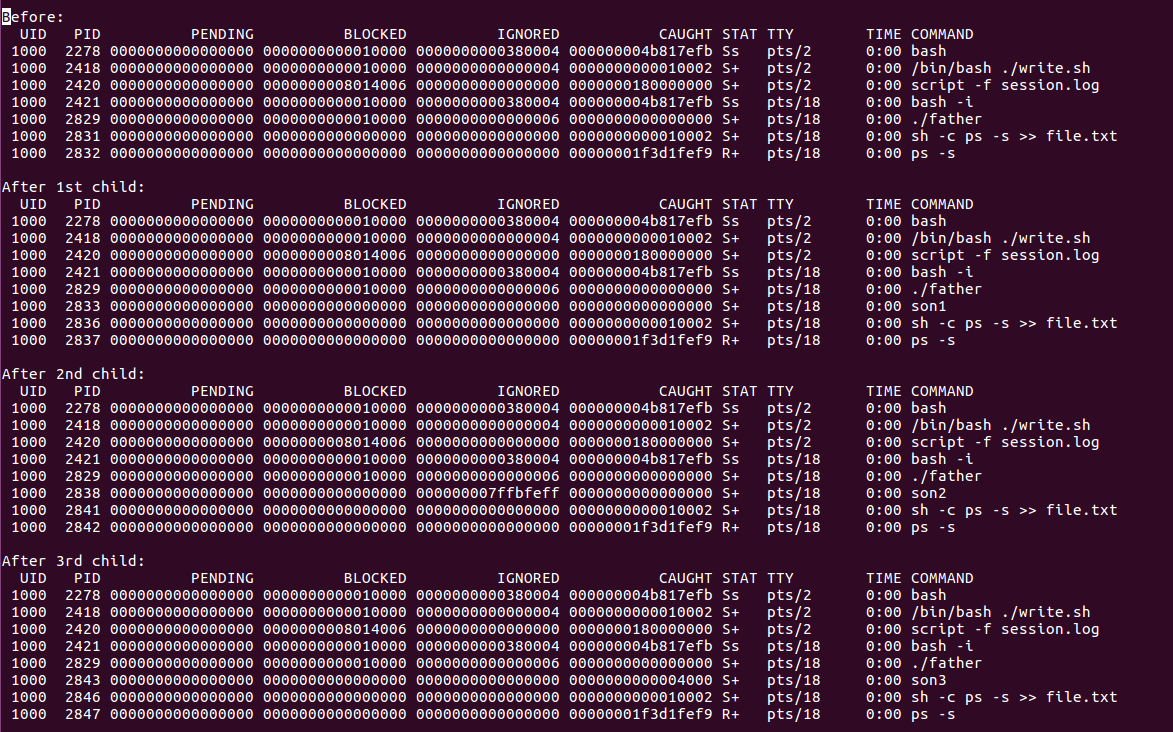
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/wait.h>  #include <signal.h>  int main()  {      int pid, ppid;      pid = getpid();      ppid = getppid();      printf("SON\_2 PARAMS: pid=%i ppid=%i\n", pid, ppid);      for(int signum = 1; signum <=31 ; signum++){          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;  } |



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.



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).

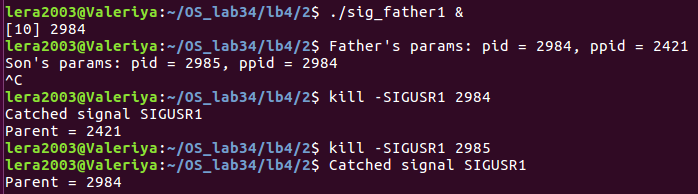
Проанализируем наследование диспозиции сигналов при создании процессов на этапе fork() и exec(). Напишем программу sig\_father.c, которая меняет диспозицию сигналов. Она задаёт обработчик многопользовательских сигналов SIGUSR1 и SIGUSR2. Родительский процесс порождает процесс-копию с помощью fork() и 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("Catched 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;  } |

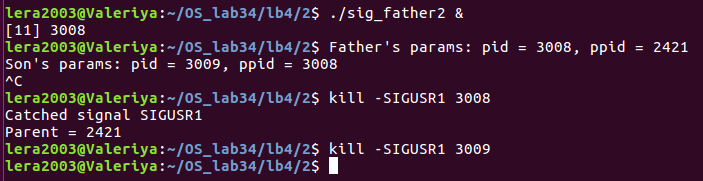


Как видно, при отправке сигнала разным процессам результат совпадает, что говорит о том, что потомок использует один и тот же обработчик, что и родитель. Это свидетельствует о наследовании диспозиции при порождении потомка на этапе fork(). Диспозиция сигналов для дочернего процесса, созданного с помощью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

*sig\_father2.c*

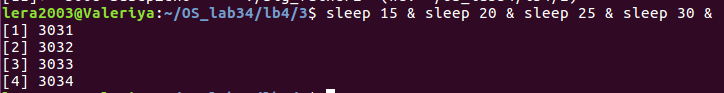
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <signal.h>  #include <unistd.h>  static void handler(int sig)  {      printf("Catched signal %s\n", sig == SIGUSR1 ? "SIGUSR1" : "SIGUSR2");      printf("Parent = %d\n", getppid());      signal(sig, SIG\_DFL);  }  int main()  {      printf("Father's 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;  } |



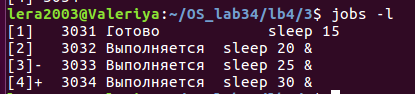
При отправке сигнала процессу-отцу срабатывает его обработчик, а при отправке процессу-потомку диспозиция этого сигнала не сохраняется и срабатывает обработчик по умолчанию (ничего не происходит). Из этого можно сделать вывод, что при создании процесса с помощью fork() и exec() диспозиция сигналов не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

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

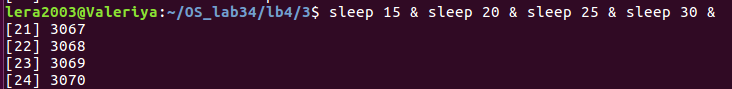
Запустим в фоновом режиме несколько утилит:



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

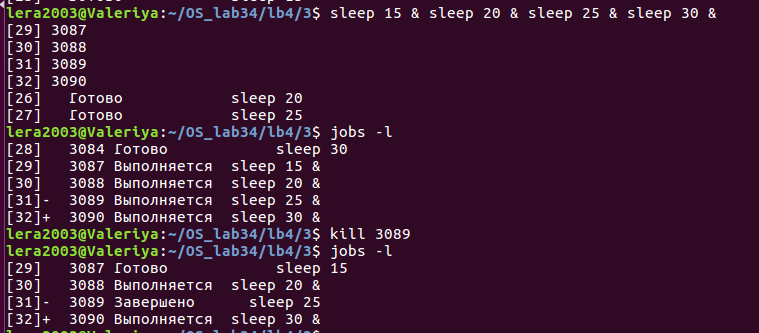


Выполнение задач начинается с начала. С помощью утилиты fg повысим приоритет задачи 23. В результате сразу начинается выполняться задача 23, не в фоновом режиме.





С помощью команды kill отменим одно из невыполненных заданий



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

nice1) — утилита, запускающая программу с измененным приоритетом. Если не указано ни одного аргументы, 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);

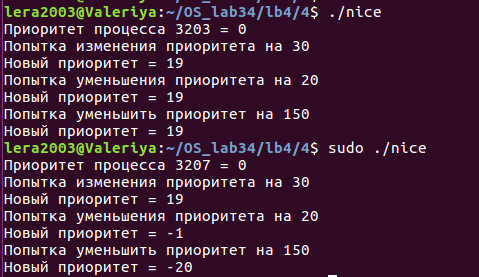
Функциями getpriority() и setpriority() можно получить и установить приоритет для процесса, группы, и пользователя, в зависимости от заданных значений which и who:

Which: PRIO\_PROCESS, PRIO\_USER, PRIO\_PGRP,

Who: идентификатор PID, Prio: приоритет.

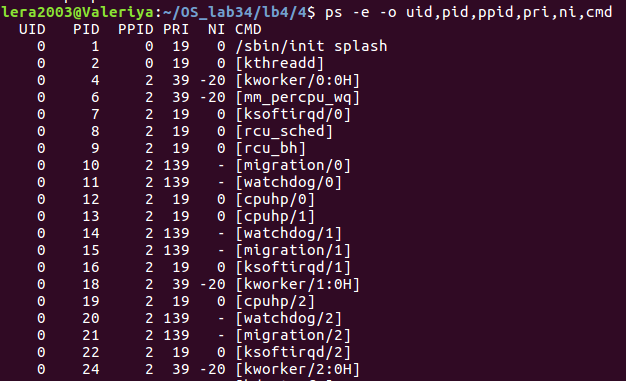
*nice.c*

|  |
| --- |
| #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;  } |



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.



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



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



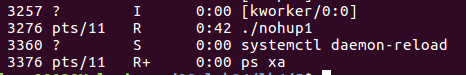
**5. Ознакомьтесь с командой nohup(1).**

nohup(1) — утилита, позволяющая запустить команду, невосприимчивую к сигналам потери связи (hungup), 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++);  } |

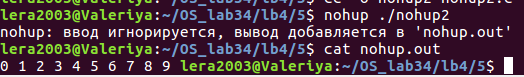




Таким образом, при выходе из системы процесс nohup1 не завершился, так как команда nohup позволила этому процессу игнорировать сигнал SIGHUP, который высылается при выходе из системы. Рассмотрим ещё один пример с командой nohup, выводящей на экран строку.

*nohup2.c*

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

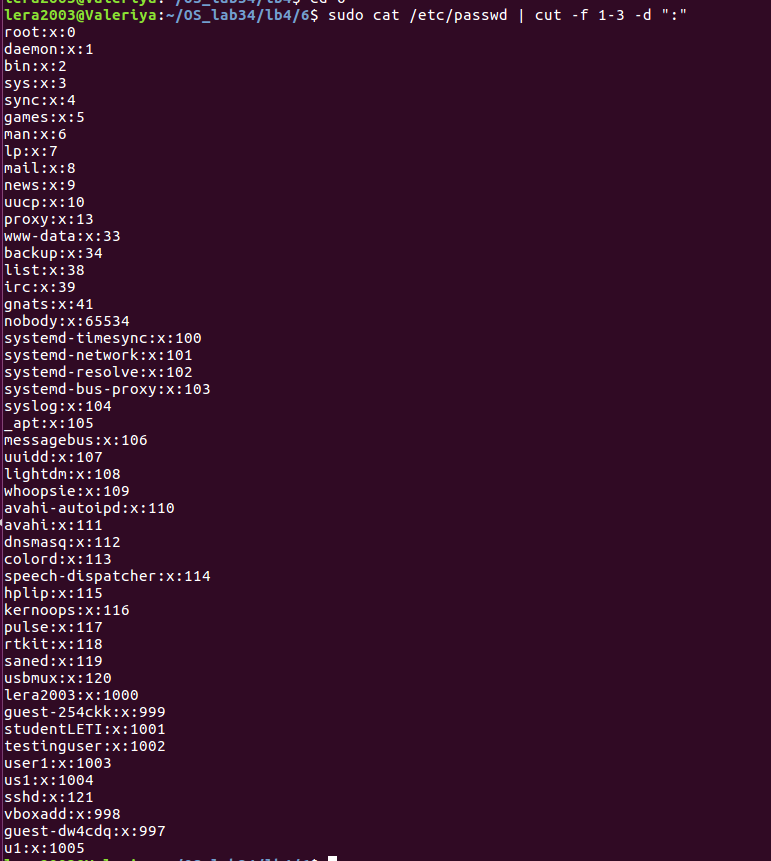


Как видно, в результате работы команды nohup запись производилась не в консоль, а в файл nohup.out.

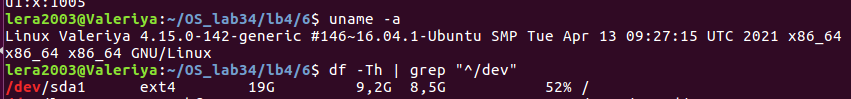
**6. Определите uid процесса, каково минимальное значение и кому оно принадлежит. Каково минимальное и максимальное значение pid, каким процессам принадлежат?**

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

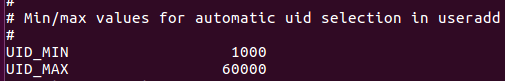
Минимальное значение uid равно нулю и принадлежит root. В файле /etc/passwd на 1-м месте пользователь, а на третьем – его uid.



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

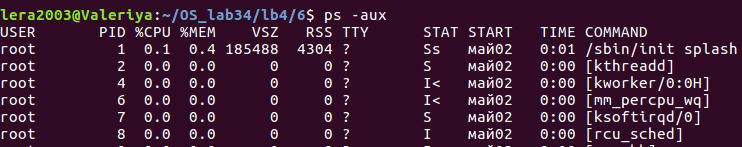
****

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



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

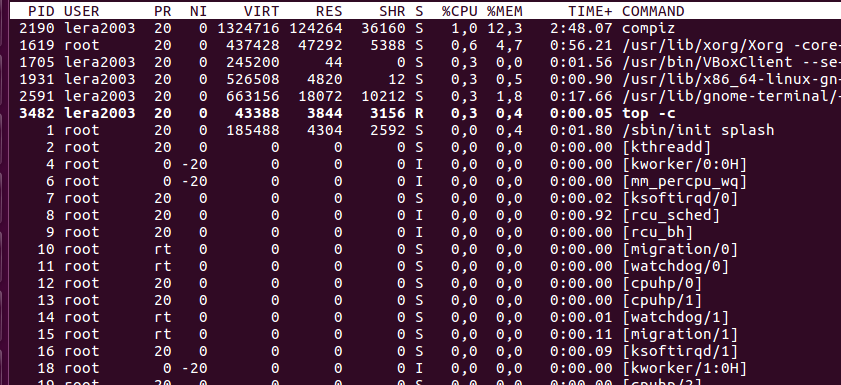
Минимальное значение PID равно одному и принадлежит процессу init. Этот процесс запускается после загрузки системы и является родительским процессом для всех остальных. PID 0 зарезервирован ядром. Таким образом, любой новый процесс будет иметь pid больше либо равный двум.



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



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



**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.

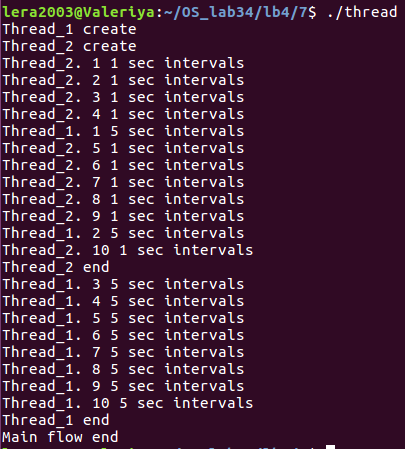
pthread\_t — идентификатор потока.

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



**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).

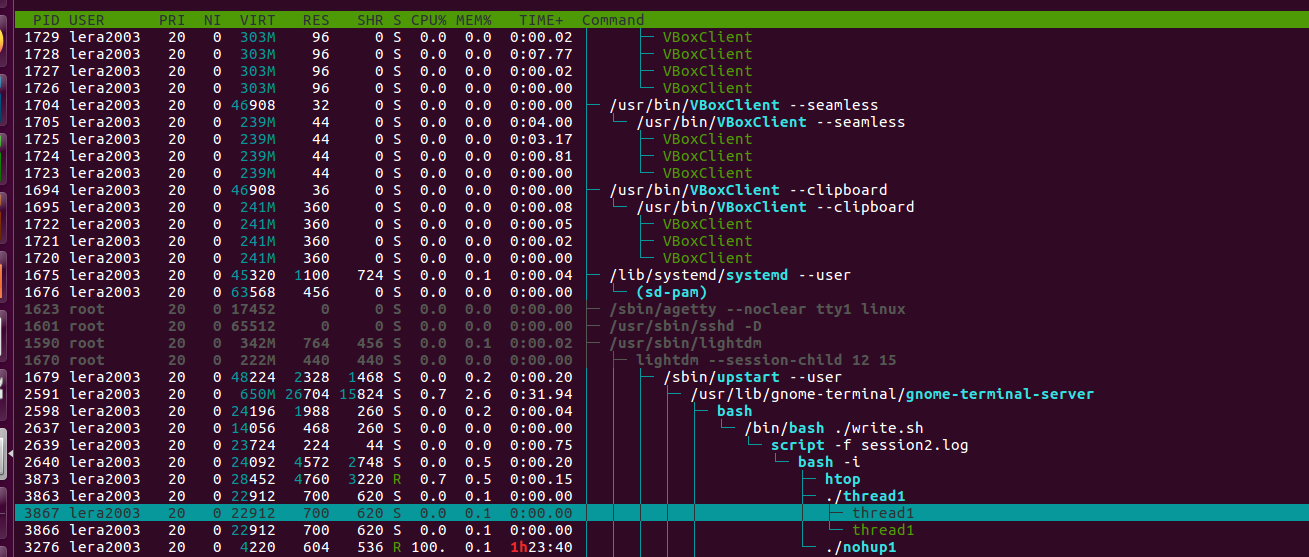
*thread1.c*

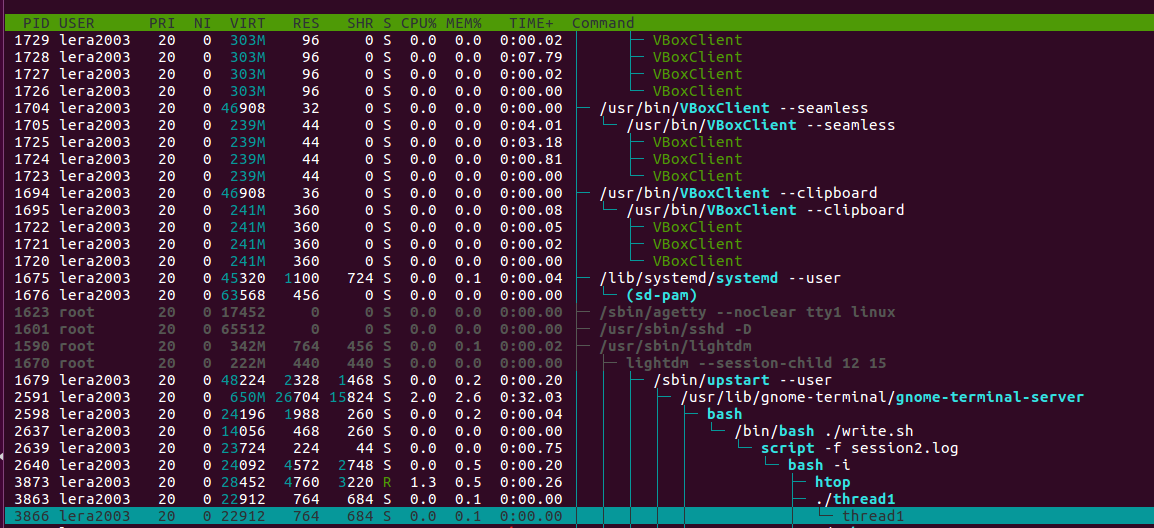
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <unistd.h>  #include <time.h>  #include <string.h>  void\* thread\_function1(void \*arg);  void\* thread\_function2(void \*arg);  char\* get\_cur\_time();  void \*thread\_function1(void \*arg)  {      int i;      system("echo 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);  } |

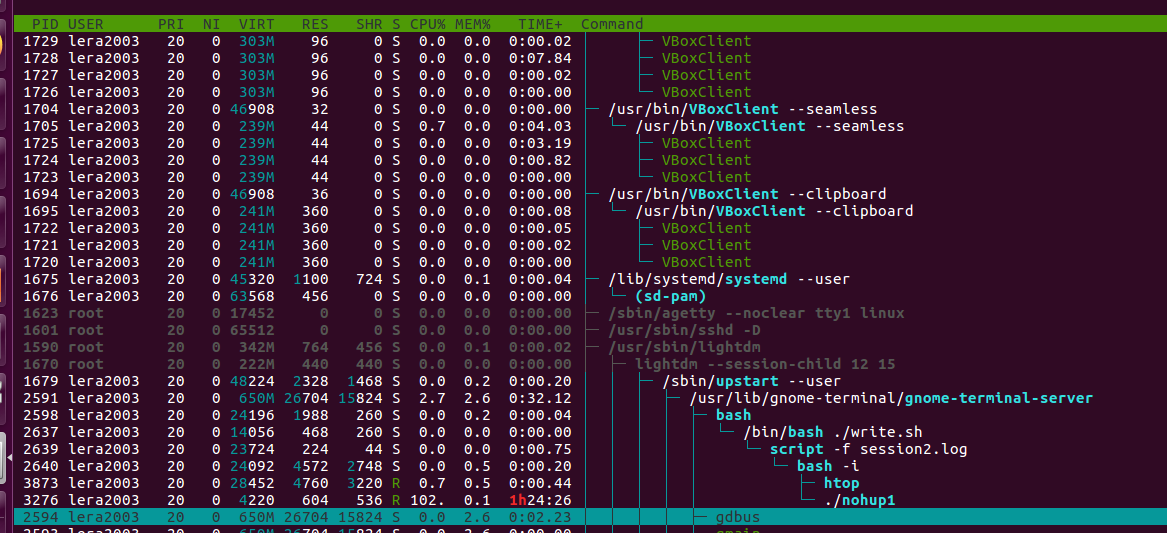
*outfile.txt*

|  |
| --- |
| 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 \\_ [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 \\_ [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-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-udevd  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/gvfsd  1912 ? SLl 0:00 \\_ gnome-keyring-daemon --start --components pkcs11,secrets  1914 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-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/gvfsd-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/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec\_spaw/2  2574 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-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 \\_ [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-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-udevd  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+ 1:24 \\_ /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/gvfsd  1912 ? SLl 0:00 \\_ gnome-keyring-daemon --start --components pkcs11,secrets  1914 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-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 4:18 \\_ 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 /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/gvfsd-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/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec\_spaw/2  2574 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-dnssd --spawner :1.3 /org/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/colord/colord  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 \\_ [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-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-udevd  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+ 1:24 \\_ /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/gvfsd  1912 ? SLl 0:00 \\_ gnome-keyring-daemon --start --components pkcs11,secrets  1914 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-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 4:19 \\_ 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 /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/gvfsd-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/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec\_spaw/2  2574 ? Sl 0:00 \\_ /usr/lib/gvfs/gvfsd-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 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 ? 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 1 завершен  Главный поток завершен |

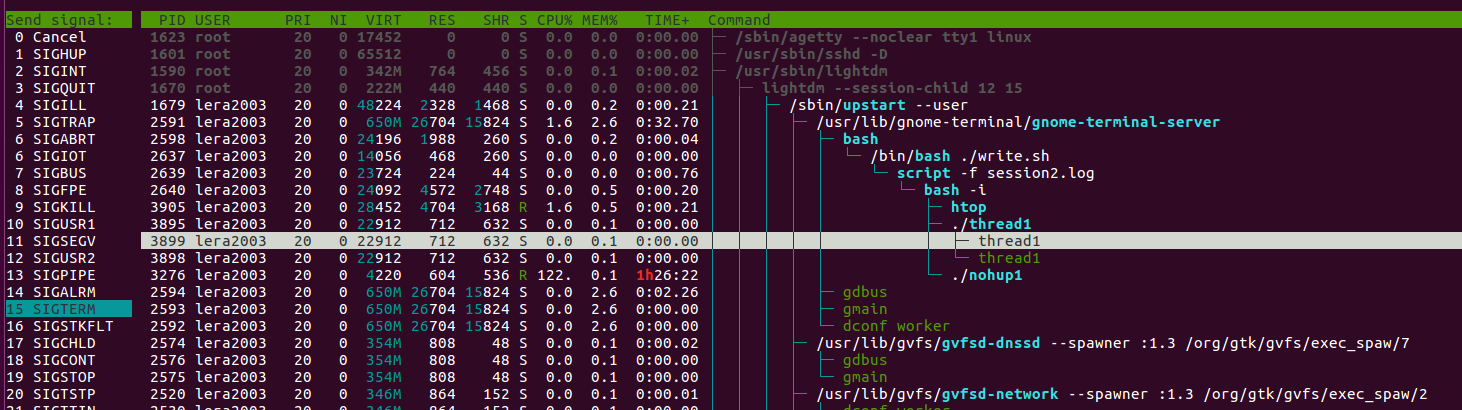
Для наглядности изменим программу так: все выводы перенаправим в файл (чтобы можно было без проблем запустить её в фон), а анализировать работу будем через утилиту htop.







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.





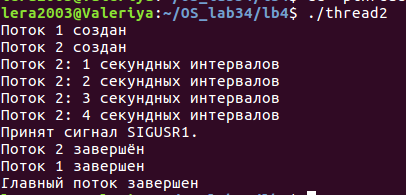
Как видно, после послания одному потоку SIGTERM завершилась вся программа, что подтверждает информацию из теоретических сведений.

**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);  } |

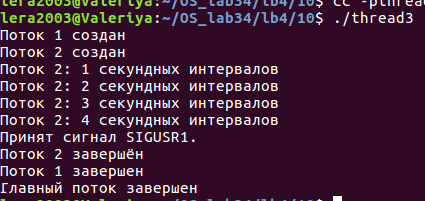


Как и ожидалось, поток 2 завершён из потока 1 в течение 5 секундных интервалов.

**10. Последняя модификация предполагает создание собственного обработчика сигнала, содержащего уведомление о начале его работы и возврат посредством функцииpthread\_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 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);      pthread\_exit(NULL);  } |



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).

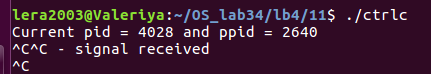
**11. Перехватите сигнал «CTRL C» для процесса и потока однократно, а также многократно с восстановлением исходного обработчика после нескольких раз срабатывания. Проделайте аналогичную работу для переназначения другой комбинации клавиш.**

Перехватим сигнал Ctrl C для процесса и потока однократно, а также

многократно с восстановлением исходного обработчика после нескольких раз срабатывания.

*ctrlc.c*

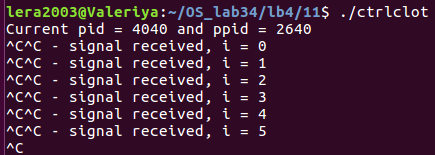
|  |
| --- |
| #include <stdio.h>  #include <signal.h>  void handler()  {      puts("^C - signal received");      signal(SIGINT, SIG\_DFL); //восстановление диспозиции по умолчанию  }  int main()  {      int pid, ppid;      pid = getpid();      ppid = getppid();      printf("Current pid = %d and ppid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



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*

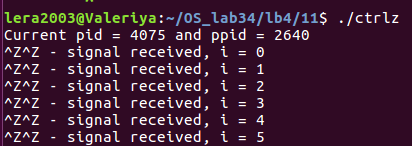
|  |
| --- |
| #include <stdio.h>  #include <signal.h>  void handler()  {      static int i = 0;      printf("^C - signal received, i = %d\n", i);      if (i++ == 5) //количество срабатываний текущего обработчика          signal(SIGINT, SIG\_DFL); //восстановление стандартного  //обработчика  }  int 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;  } |



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

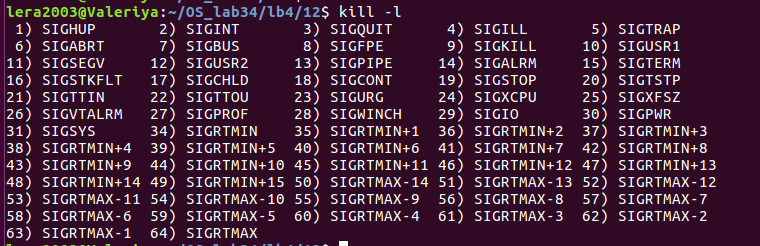
*ctrlz.c*

|  |
| --- |
| #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;  } |



**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.**

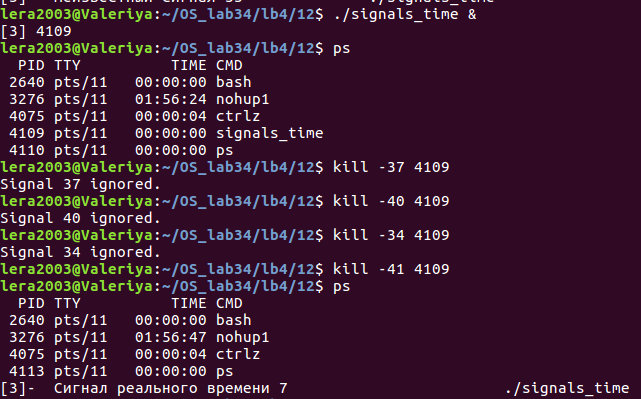
С помощью утилиты kill выведем список сигналов и дадим краткую характеристику.



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 раннее, 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;  } |



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 планирвоания, 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;  } |

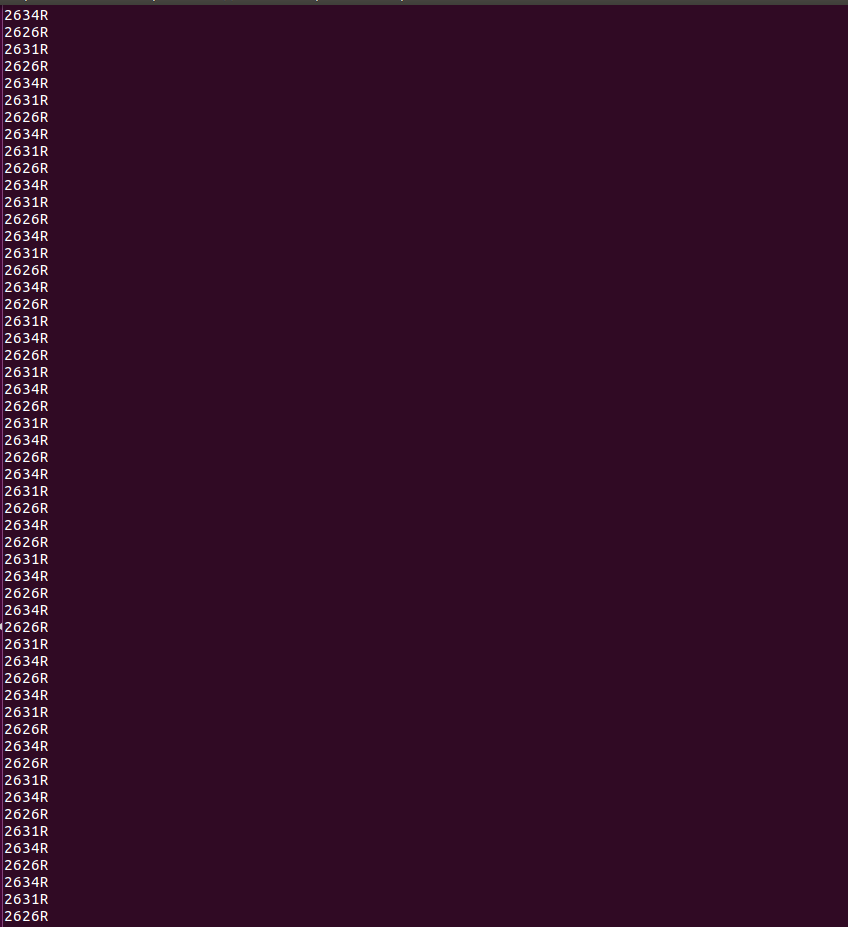
*RRFIFO.c*

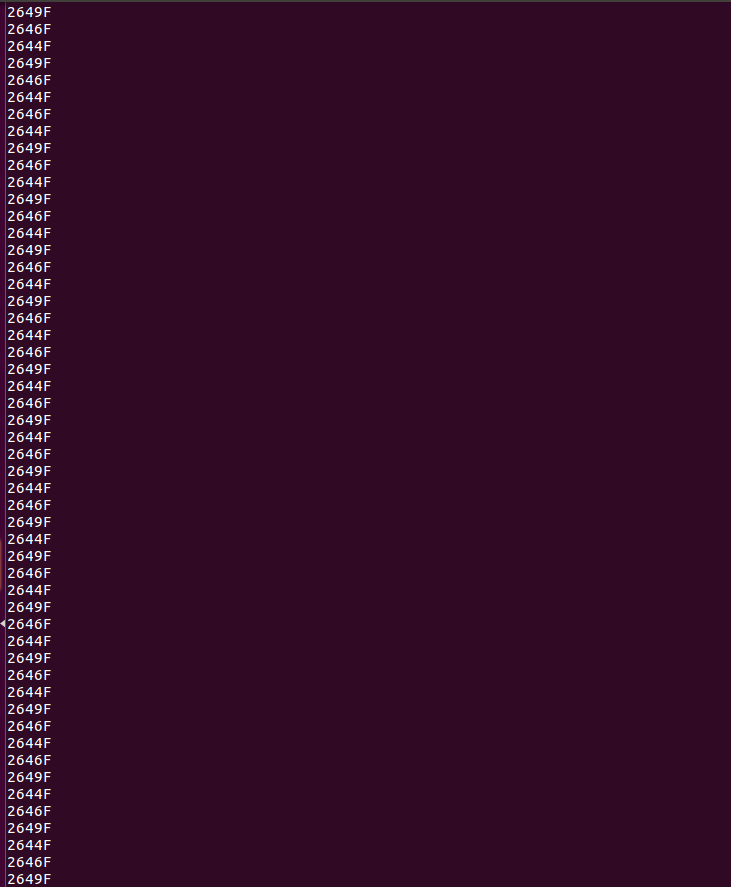
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sched.h>  #include <unistd.h>  #define PROC\_AMOUNT 10  int main() {      int i;      char\* program = "son1";      struct sched\_param params;      params.sched\_priority = PROC\_AMOUNT;      int sched;      for (i = 0; i < PROC\_AMOUNT; 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;  } |

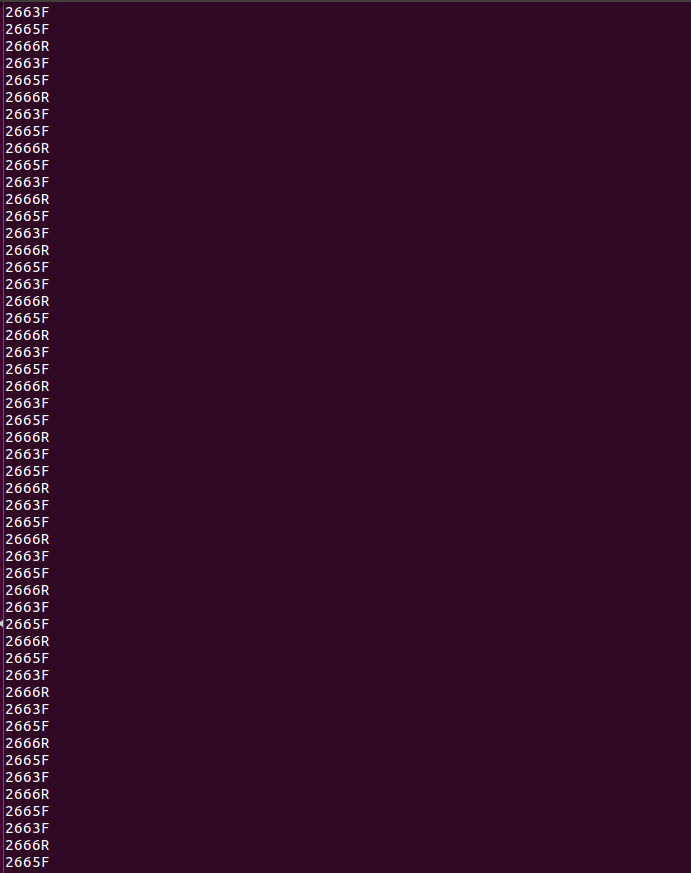
*son1.c*

|  |
| --- |
| #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:





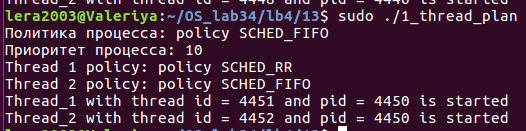


Таким образом, всегда происходит борьба за ресурс. Как с одинаковыми, так и различными политиками планирования.

Исследуем характеристики планирования потоков. Изменять стратегию планирования потока можно с помощью команды pthread\_attr\_setschedpolicy. Перед этим нужно инициализировать описатель 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;      }  } |



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;  } |





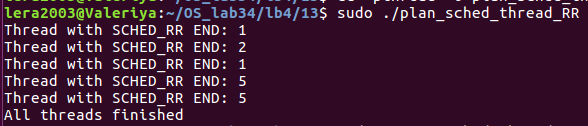
Можно сделат вывод, что борьба за ресурс у нитей происходит более равномерно.

Модифицируем программу. Теперь уменьшим количество подсчетов для политики RR.

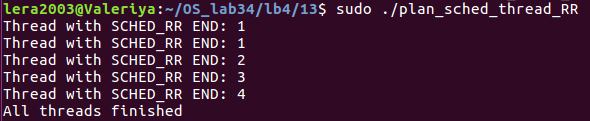
*RRsmall.c*

|  |
| --- |
| #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.



Результат работы программы с малым количеством вычислений.



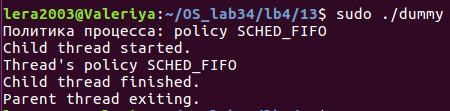
Как видно, при большом количестве вычислений величина кванта превышается, потоки передают управление друг другу, следовательно, порядок завершение произволен. При малом количестве вычислений потоки завершаются друг за другом.

Рассмотрим команду pthread\_attr\_setinheritsched() сначала с PTHREAD\_INHERIT\_SCHED (т. е. наследуемся от родительского), а потом – PTHREAD\_EXPLICIT\_SCHED (устанавливаем явно отдельно).

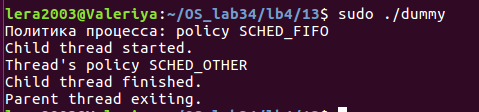
*dummy.c*

|  |
| --- |
| #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:              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.



Результат с EXPLICIT.



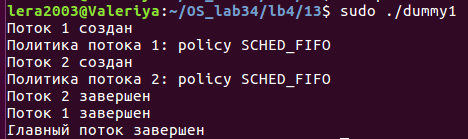
Как видно, если применять EXPLICIT политика потока – SCHED\_OTHER, поскольку не была задана явно. Если же INHERIT – политика наследуется от политики процесса (который его породил).

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

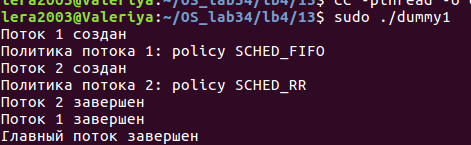
*dummy1.c*

|  |
| --- |
| #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.



Результат с EXPLICIT.



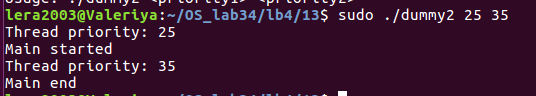
Программа работает как ожидалось, поведение наследования от процесса и наследования от потока не различаются.

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;  } |



*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;  } |



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.

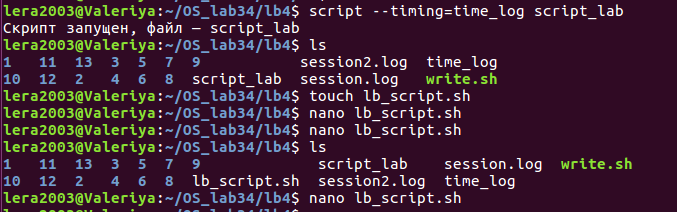
Подытожив, можно сказать, что на данной системе (WSL, Ubuntu) нет ограничений на изменение процедур планирования, приоритета для потоков одного процесса.

**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 |



**Вывод.**

Проанализировано порождение процессов, и разница в зависимости от того, что отец ожидает завершение потомка, завершается до и не ождиает завершения. Таким образом, у последнего получается новый родитель, а именно /sbin/upstart –user.

В ходе лабораторной работы ознакомились с сигналами посредством утилиты kill. Созданы программы, позволяющие совершать следующие действия над сигналами: игнорирование, перехват и обработка, действие по умолчанию, то есть диспозиции. Проведен анализ диспозиции сигналов на этапе fork() и exec(). На этапе fork() диспозиция наследовалась, на втором же нет.

Написаны программы, в которых использовались nice и getpriority для установки приоритета, в программах произведена попытка изменения приоритетов, что было возможно только с правами суперпользователя.

Изучена утилита nohup(), которая позволяет запустить команду, невосприимчивую к сигналам потери связи. То есть при выходе из системы процесс запущенный с nohup не завершается.

Определено максимальное значение UID. В моей файловой системе ext4 это 4294967295. Максимальное значение PID может варьироваться, значенеи можно найти в файле /poc/sys/kernel/pid\_max. Минимальное значение 0 зарезервировано ядром.

Проведена работа с нитями. Создавалось несколько нитей, затем, одна из них уничтожалась, при этом общая программа не завершалась, другая нить продолжала работу. Написана программа, в которой управление второй нитью осуществляется из первой посредством сигнала SIGUSR1. Также был написан собственный обработчик сигнала.

Осуществлен перехват сигнала ctrl+c однократно и многократно, результат работы продемоснтрирован в отчете. Эксперимент был проведён повторно для игнала ctrl+z.

Проанализирована процедура планирования и борьба за ресурс для нитей и процессов с политиками планирования SCHED\_RR и SCHED\_FIFO. Эксперементально изучено наследование политики планирования для нитей при использовании PTHREAD\_INHERIT\_SCHED и PTHREAD\_EXPLICIT\_SCHED. В первом случае наследование от родительского происходит, во втором политика устанавливается явно отдельно.

Написан скрипт, запускающий выполнение данной лабораторной работы.

**Список литературы.**

1. «Системное программное обеспечение. Практические вопросы разработки системных приложений. Учебное пособие» Душутина Е.В.
2. Сайт [fork(2) - Справочная страница Linux (man7.org)](https://man7.org/linux/man-pages/man2/fork.2.html)
3. Сайт [Создание процессов с помощью вызова fork(). (opennet.ru)](https://www.opennet.ru/docs/RUS/linux_parallel/node7.html)
4. Сайт [Ubuntu Manpage: Welcome](https://manpages.ubuntu.com/)
5. Сайт <https://www.opennet.ru/docs/RUS/linux_parallel/node7.html>