|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Отчёт по лабораторной работе №4 по курсу**

**«Операционные системы»**

**Тема**  Файловая система \proc

**Студент**  Якуба Д.В.

**Группа**  ИУ7-63Б

**Оценка (баллы)**

**Преподаватель**  Рязанова Н.Ю.

*Москва, 2021 г*

# /proc/pid/cmdline

|  |
| --- |
| void **printCMDLINE**()  {  char pathToOpen[PATH\_MAX];  snprintf(pathToOpen, PATH\_MAX, "/proc/%d/cmdline", PID);  FILE \*file = fopen(pathToOpen, "r");  char buf[BUFFSIZE];  int len = fread(buf, 1, BUFFSIZE, file);  buf[len - 1] = 0;  printf("\nCMDLINE CONTENT:\n");  printf("pid: %d\ncmdline:%s\n", PID, buf);  fclose(file);  } |

Данный файл содержит полную командную строку процесса, если он полностью не выгружен или убит. В любом из последних двух случаев файл пуст и чтение его приводит к тому же результату, что и чтение пустой строки.

Результат выполнения:

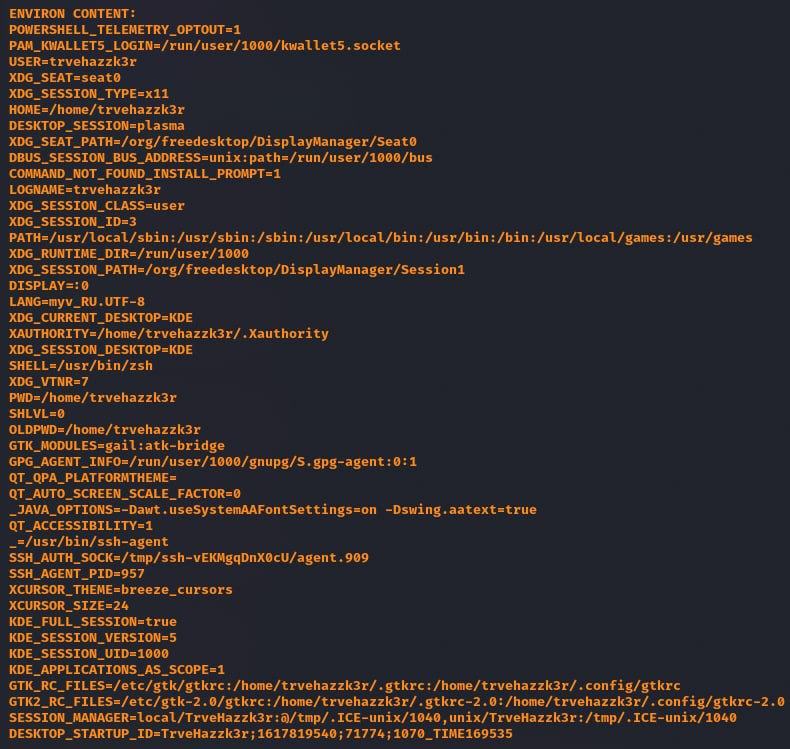


# /proc/pid/environ

|  |
| --- |
| void **printENVIRON**()  {  char pathToOpen[PATH\_MAX];  snprintf(pathToOpen, PATH\_MAX, "/proc/%d/environ", PID);  FILE \*file = fopen(pathToOpen, "r");  int len;  char buf[BUFFSIZE];  printf("\nENVIRON CONTENT:\n");  while ((len = fread(buf, 1, BUFFSIZE, file)) > 0)  {  for (int i = 0; i < len; i++)  if (!buf[i])  buf[i] = '\n';  buf[len - 1] = '\n';  printf("%s", buf);  }  fclose(file);  } |

Файл содержит набор пар “переменная=значение”, доступный каждому пользовательскому процессу. Такой набор называется набором переменных окружения.

Результат выполнения:



Приведённые переменные окружения:

USER – пользователь, запустивший процесс.

HOME – домашний каталог текущего пользователя.

LOGNAME – имя текущего пользователя.

PATH – список каталогов, в которых находятся исполняемые файлы.

LANG – язык и кодировка текущего пользователя.

SHELL – путь к оболочке командной строки.

PWD – путь к рабочей директории.

SHLVL – уровень текущей командной оболочки.

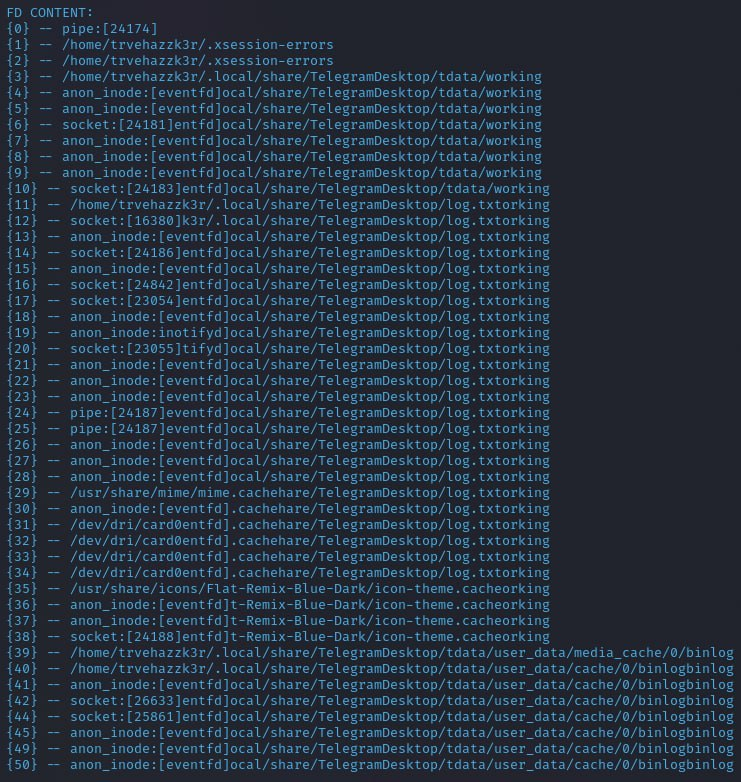
OLDPWD – путь к предыдущему рабочему каталогу.

# /proc/pid/fd

|  |
| --- |
| void **printFD**()  {  char pathToOpen[PATH\_MAX];  snprintf(pathToOpen, PATH\_MAX, "/proc/%d/fd/", PID);  DIR \*dir = opendir(pathToOpen);  printf("\nFD CONTENT:\n");  struct **dirent** \*readDir;  char string[PATH\_MAX];  char path[BUFFSIZE] = {'\0'};  while ((readDir = readdir(dir)) != NULL)  {  if ((strcmp(readDir->d\_name, ".") != 0) && (strcmp(readDir->d\_name, "..") != 0))  {  sprintf(path, "%s%s", pathToOpen, readDir->d\_name);  readlink(path, string, PATH\_MAX);  printf("{%s} -- %s\n", readDir->d\_name, string);  }  }  closedir(dir);  } |

Данная поддиректория содержит одну запись для каждого файла, который открыт процессом. Имя каждой такой записи соответствует номеру файлового дескриптора и является символьной ссылкой на реальный файл.

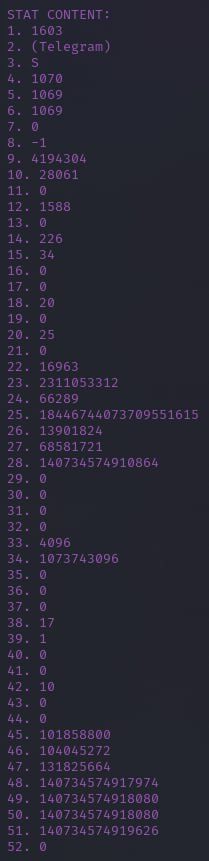
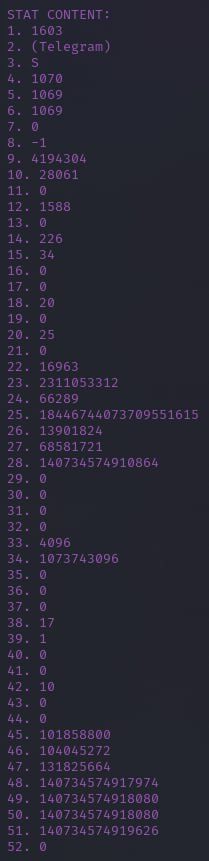
Результат выполнения представлен на следующей странице.



# /proc/pid/stat

|  |
| --- |
| void **printSTAT**()  {  char pathToOpen[PATH\_MAX];  snprintf(pathToOpen, PATH\_MAX, "/proc/%d/stat", PID);  char buf[BUFFSIZE];  FILE \*file = fopen(pathToOpen, "r");  fread(buf, 1, BUFFSIZE, file);  char \*tokens = strtok(buf, " ");  printf("\nSTAT CONTENT: \n");  for (int i = 1; tokens != NULL; i++)  {  printf("%d. %s \n", i, tokens);  tokens = strtok(NULL, " ");  }  fclose(file);  } |

Данный файл содержит в себе статусную информацию о процессе.

Результат выполнения:

Описание содержимого файла:

1) pid - уникальный идентификатор процесса.

2) comm - имя исполняемого файла в круглых скобках.

3) state - состояние процесса.

4) ppid - уникальный идентификатор процесса-предка.

5) pgrp - уникальный идентификатор группы.

6) session - уникальный идентификатор сессии.

7) tty\_nr – управляющий терминал.

8) tpgid – уникальный идентификатор группы управляющего терминала.

9) flags – флаги.

10) minflt - Количество незначительных сбоев, которые возникли при выполнении процесса, и которые не требуют загрузки страницы памяти с диска.

11) cminflt - количество незначительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.

12) majflt - количество значительных сбоев, которые возникли при работе процесса, и которые потребовали загрузки страницы памяти с диска.

13) cmajflt - количество значительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.

14) utime - количество тиков, которые данный процесс провел в режиме пользователя.

15) stime - количество тиков, которые данный процесс провел в режиме ядра.

16) cutime - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме пользователя.

17) cstime - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме ядра.

18) priority – для процессов реального времени это отрицательный приоритет планирования минус один, то есть число в диапазоне от -2 до -100, соответствующее приоритетам в реальном времени от 1 до 99. Для остальных процессов это необработанное значение nice, представленное в ядре. Ядро хранит значения nice в виде чисел в диапазоне от 0 (высокий) до 39 (низкий), соответствующих видимому пользователю диапазону от -20 до 19.

19) nice - значение для nice в диапазоне от 19 (наиболее низкий приоритет) до -20 (наивысший приоритет).

20) num\_threads – число потоков в данном процессе.

21) itrealvalue – количество мигов до того, как следующий SIGALARM будет послан процессу интервальным таймером. С ядра версии 2.6.17 больше не поддерживается и установлено в 0.

22) starttime - время в тиках запуска процесса после начальной загрузки системы.

23) vsize - размер виртуальной памяти в байтах.

24) rss - резидентный размер: количество страниц, которые занимает процесс в памяти. Это те страницы, которые заняты кодом, данными и пространством стека. Сюда не включаются страницы, которые не были загружены по требованию или которые находятся в своппинге.

25) rsslim - текущий лимит в байтах на резидентный размер процесса. 26) startcode - адрес, выше которого может выполняться код программы.

27) endcode - адрес, ниже которого может выполняться код программ.

28) startstack - адрес начала стека.

29) kstkesp - текущее значение ESP (указателя стека).

30) kstkeip - текущее значение EIP (указатель команд).

31) signal - битовая карта ожидающих сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.

32) blocked - битовая карта блокируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.

33) sigignore - битовая карта игнорируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.

34) sigcatch - битовая карта перехватываемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.

35) wchan - "канал", в котором ожидает процесс.

36) nswap - количество страниц на своппинге (не обслуживается).

37) сnswap - суммарное nswap для процессов-потомков (не обслуживается).

38) exit\_signal - сигнал, который будет послан предку, когда процесс завершится.

39) processor - номер процессора, на котором последний раз выполнялся процесс.

40) rt\_priority - приоритет планирования реального времени, число в диапазоне от 1 до 99 для процессов реального времени, 0 для остальных.

41) policy - политика планирования.

42) delayacct\_blkio\_ticks - суммарные задержки ввода/вывода в тиках.

43) guest\_time – гостевое время процесса (время, потраченное на выполнение виртуального процессора на гостевой операционной системе) в тиках.

44) cguest\_time - гостевое время для потомков процесса в тиках.

45) start\_data - адрес, выше которого размещаются инициализированные и неинициализированные (BSS) данные программы.

46) end\_data - адрес, ниже которого размещаются инициализированные и неинициализированные (BSS) данные программы.

47) start\_brk - адрес, выше которого куча программы может быть расширена с использованием brk().

48) arg\_start - адрес, выше которого размещаются аргументы командной строки (argv).

49) arg\_end - адрес, ниже которого размещаются аргументы командной строки (argv).

50) env\_start - адрес, выше которого размещается окружение программы.

51) env\_end - адрес, ниже которого размещается окружение программы.

52) exit\_code – статус завершения потока в форме, возвращаемой waitpid().