МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«**Московский технический университет связи и информатики**»
(**МТУСИ**)

Кафедра «Системное программирование»

ОТЧЕТ
по лабораторной работе № 5
Вариант №21

по дисциплине «**Операционные системы**»

Выполнил:
студент гр. БФИ2202
_____ Сидорук Д. В.
«____» _____ 2024 г.

Проверил:
старший преподаватель
_____ Алексанян Д. А.
«____» _____ 2024 г.

Москва, 2024 г.

Содержание

1 Цель работы

1. Изучение системных функций выделения памяти в ОС GNU/LINUX.
2. Получение практических навыков работы с динамической памятью в ОС GNU/LINUX.

2 Задание

1. Написать программу, работающую в одном из 2-х режимов, задающимся при запуске в качестве аргумента командной строки:
   - Режим выделение памяти malloc() равно освобождению памяти free()
   - Режим выделение памяти больше malloc(), чем освобождение памяти free()

Выделение и освобождение памяти проводить параллельно с задержкой 100–1000 нс в бесконечном цикле. Предусмотреть прерывание процесса по сигналу, направленному асинхронно, с освобождением всей динамически выделенной памяти.

2. Написать программу для создания карты виртуальной памяти процесса, идентификатор которого передается первым аргументом, а полный путь до директории сохранения файла карты памяти – вторым аргументом. Именовать файлы карты по шаблону map_<PID>_<date&time>, где <PID> - идентификатор процесса, переданный в программу и соответствующий процессу, для которого создается карта, <date&time> - дата и время создания карты в формате yyyy-mm-dd_h:m:s. Наполнение карты взять из соответствующего процесса в /proc/ или использовать утилиту pmap.
3. Создать службу, запускающуюся каждые 30 секунд и выполняющую программу из п.2., проверить работу и получить 10-15 файлов с картами памяти для из п.1.
4. Проанализировать полученные результаты, написав программу, строящую график по данным карт виртуальной памяти из п.3 (для всех файлов карт в указанной при запуске директории). Выявить изменение размера кучи, при необходимости провести корректировку размеров выделяемой malloc() памяти для наглядного отображения (для данной программы можно использовать, например, Python).
5. Включить в отчет ответы на вопросы в свободной форме:
   - что такое стек и куча?
   - как происходит статическое и динамическое выделение памяти?
   - каковы плюсы и минусы статического и динамического выделения памяти?

3 Ход работы

В ходе выполнения данной лабораторной работы были созданы следующие файлы:

- install/operating_systems_laboratories_4.service
- install/operating_systems_laboratories_4.timer
- sources/common/globals.hpp
- sources/common/PidFile.cpp
- sources/common/PidFile.hpp

- sources/leaker/main.cpp
- sources/mapper/main.cpp
- sources/monitor/main.cpp
- sources/plotter/main.py
- CMakeLists.txt

В листинге ниже приведено содержание install/operating_systems_laboratories_4.servi
(лист. 1):

Лист. 1 – Содержание install/operating_systems_laboratories_4.service

```
[Unit]
Description=Run monitor

[Service]
Type=oneshot
ExecStart=/usr/share/operating_systems_laboratories_4/bin/monitor
```

В листинге ниже приведено содержание install/operating_systems_laboratories_4.timer
(лист. 2):

Лист. 2 – Содержание install/operating_systems_laboratories_4.timer

```
[Unit]
Description=Run monitor every 30 seconds

[Timer]
OnUnitActiveSec=30s
OnBootSec=30s

[Install]
WantedBy=timers.target
```

В листинге ниже приведено содержание sources/common/globals.hpp (лист. 3):

Лист. 3 – Содержание sources/common/globals.hpp

```
#pragma once

#include <string_view>

constexpr inline std::string_view LEAKER_APPNAME =
    "operating_systems_laboratories_leaker";
constexpr inline std::string_view MONITOR_DUMPS_PATH =
    "/usr/share/operating_systems_laboratories_4/dumps";
constexpr inline std::string_view MAPPER_PATH =
    "/usr/share/operating_systems_laboratories_4/bin/mapper";
```

В листинге ниже приведено содержание sources/common/PidFile.cpp (лист. 4):

Лист. 4 – Содержание sources/common/PidFile.cpp

```cpp
#include "PidFile.hpp"
#include <fcntl.h>
#include <unistd.h>

PidFile::PidFile(const std::string_view &path_, const std::string_view &name)
    : path(std::filesystem::path(path_) / (std::string{name} + ".pid")) {}

PidFile::~PidFile() {
  if (descriptor >= 0) {
    ::close(descriptor);
    std::filesystem::remove(path);
  }
}

bool PidFile::tryLock() {
  descriptor = ::open(path.string().c_str(), O_RDWR | O_CREAT | O_EXCL, 0644);
  if (descriptor >= 0) {
    struct flock lock_info = {};
    lock_info.l_type = F_WRLCK;    /* exclusive write lock */
    lock_info.l_whence = SEEK_SET; /* use start and len */
    if (::fcntl(descriptor, F_SETLK, &lock_info) < 0) {
      ::close(descriptor);
      descriptor = -1;
    }
  }
  return descriptor >= 0;
}

bool PidFile::hasLock() const { return descriptor >= 0; }

bool PidFile::setPID(uint32_t pid) {
  if (descriptor >= 0 && ftruncate(descriptor, 0) >= 0) {
    return dprintf(descriptor, "%u", pid) > 0;
  }
  return false;
}

uint32_t PidFile::getPID() const {
  if (auto file = fopen(path.string().c_str(), "r"); file) {
    if (char pid[16] = {}; fgets(pid, sizeof(pid), file)) {
      return std::stoul(pid);
    }
  }
  return 0;
}

PidFile PidFile::tmp(const std::string_view &name) {
  return PidFile("/tmp", name);
}
```

```
50
51 PidFile PidFile::var(const std::string_view &name) {
52   return PidFile("/var/run", name);
53 }
```

В листинге ниже приведено содержание sources/common/PidFile.hpp (лист. 5):

Лист. 5 – Содержание sources/common/PidFile.hpp

```
1  #pragma once
2
3  #include <filesystem>
4  #include <string>
5
6  class PidFile {
7  public:
8    PidFile(const std::string_view &path, const std::string_view
     ↪  &name);
9    ~PidFile();
10
11   bool tryLock();
12
13   bool hasLock() const;
14   bool setPID(uint32_t);
15   uint32_t getPID() const;
16
17   std::string getPath() const { return path.string(); }
18   int getDescriptor() const { return descriptor; }
19
20   static PidFile tmp(const std::string_view &name);
21   static PidFile var(const std::string_view &name);
22
23 private:
24   std::filesystem::path path;
25   int descriptor = -1;
26 };
```

В листинге ниже приведено содержание sources/leaker/main.cpp (лист. 6):

Лист. 6 – Содержание sources/leaker/main.cpp

```
1  #include <charconv>
2  #include <chrono>
3  #include <csignal>
4  #include <iostream>
5  #include <random>
6  #include <string_view>
7  #include <thread>
8
9  #include "common/PidFile.hpp"
10 #include "common/globals.hpp"
11
```

```cpp
using namespace std::string_view_literals;

std::atomic<bool> is_running{false};

int main(int argc, char *argv[]) {
  constexpr std::string_view USAGE = "Usage: leaker (off|on) [ALLOC_SIZE]";

  if (argc < 2) {
    std::cerr << USAGE << std::endl;
    return 1;
  }

  auto pid_file = PidFile::tmp(LEAKER_APPNAME);
  if (!pid_file.tryLock()) {
    std::cerr << "Can't lock " << pid_file.getPath() << std::endl;
    return 1;
  } else {
    pid_file.setPID(getpid());
  }

  std::signal(SIGINT, [](int signal) { is_running = false; });
  std::signal(SIGUSR1, [](int signal) { is_running = false; });

  double free_percent = (argv[1] == "off"sv) ? 100 : 75;
  std::size_t alloc_size = 1000;
  if (argc >= 3) {
    std::string_view alloc_size_arg = argv[2];
    auto [_, ec] = std::from_chars(
        alloc_size_arg.data(), alloc_size_arg.data() + alloc_size_arg.size(),
        alloc_size);
    if (ec != std::errc{}) {
      std::cerr << USAGE << std::endl;
      return 1;
    }
  }

  std::cout << "Entering allocation loop with ALLOC_SIZE = " << alloc_size
            << std::endl;

  std::vector<unsigned char *> memory_descriptors;
  std::random_device random_device;
  std::mt19937 random_engine(random_device());
  std::uniform_int_distribution free_distribution(0, 100);
  std::uniform_int_distribution sleep_distribution(100, 1000);

  is_running = true;
  while (true) {
    unsigned char *ptr = new unsigned char[alloc_size];
    memory_descriptors.push_back(ptr);
    if (free_distribution(random_engine) <= free_percent) {
```

```
      delete[] ptr;
      memory_descriptors.pop_back();
    }
    std::this_thread::sleep_for(
        std::chrono::nanoseconds(sleep_distribution(random_engine)
        ↪ ));

    if (!is_running) {
      break;
    }
  }

  std::cout << "Leavinig from allocation loop, cleaning up "
            << memory_descriptors.size() << " descriptors" <<
            ↪ std::endl;

  for (unsigned char *ptr : memory_descriptors) {
    delete[] ptr;
  }
}
```

В листинге ниже приведено содержание sources/mapper/main.cpp (лист. 7):

Лист. 7 – Содержание sources/mapper/main.cpp

```
#include <charconv>
#include <csignal>
#include <filesystem>
#include <format>
#include <fstream>
#include <iostream>
#include <string_view>

const std::string currentDateTime() {
  time_t now = time(0);
  struct tm tstruct;
  tstruct = *localtime(&now);

  char buf[80];
  strftime(buf, sizeof(buf), R"(%Y-%m-%d_%H:%M:%S)", &tstruct);
  return buf;
}

std::filesystem::path getMapPath(std::filesystem::path dir,
 ↪ std::size_t PID) {
  auto current_time = currentDateTime();
  return std::format("{}/map_{}_{}", dir.string(), PID,
   ↪ currentDateTime());
}

int main(int argc, char *argv[]) {
  constexpr std::string_view USAGE = "Usage: mapper PID
   ↪ SAVE_DIR_PATH";
```

```
26
27    if (argc < 3) {
28      std::cerr << USAGE << std::endl;
29      return 1;
30    }
31
32    std::size_t PID;
33    std::string_view PID_arg = argv[1];
34    std::string_view save_dir_path_arg = argv[2];
35
36    auto [_, ec] =
37        std::from_chars(PID_arg.data(), PID_arg.data() +
          ↪  PID_arg.size(), PID);
38    if (ec != std::errc{}) {
39      std::cerr << USAGE << std::endl;
40      return 1;
41    }
42    if (kill(PID, 0) != 0) {
43      std::cout << "Process with PID " << PID << " doesn't exist" <<
        ↪  std::endl;
44      return 1;
45    }
46
47    auto map_path = getMapPath(save_dir_path_arg, PID);
48    std::ofstream output_stream{getMapPath(save_dir_path_arg, PID),
49                                std::ios::out};
50    if (!output_stream) {
51      std::cerr << "Can't open " << map_path << std::endl;
52      return 1;
53    }
54
55    auto pmap_cmdline = std::format("pmap -x {}", PID);
56    auto pipe = popen(pmap_cmdline.c_str(), "r");
57    if (!pipe) {
58      std::cerr << "Couldn't execute " << pmap_cmdline << std::endl;
59      return 1;
60    }
61
62    std::array<char, 128> buffer;
63    std::string result;
64    while (fgets(buffer.data(), buffer.size(), pipe) != nullptr) {
65      result += buffer.data();
66    }
67
68    auto pmap_return_code = pclose(pipe);
69    if (pmap_return_code != 0) {
70      std::clog << "pmap return code: " << pmap_return_code <<
        ↪  std::endl;
71    }
72
73    output_stream << result;
```

```
  return 0;
}
```

В листинге ниже приведено содержание sources/monitor/main.cpp (лист. 8):

Лист. 8 – Содержание sources/monitor/main.cpp

```
#include <cstring>
#include <iostream>
#include <string>
#include <sys/wait.h>
#include <unistd.h>

#include "common/PidFile.hpp"
#include "common/globals.hpp"

int main() {
  PidFile pid_file = PidFile::tmp(LEAKER_APPNAME);
  if (pid_file.getPID() == 0) {
    std::cerr << "Leaker is not running. Terminating" <<
     ↪  std::endl;
    return 1;
  }

  int PID = fork();
  if (PID == 0) {
    std::string pid_string = std::to_string(pid_file.getPID());
    const char *argv[] = {"mapper", pid_string.c_str(),
                          MONITOR_DUMPS_PATH.data(), NULL};
    if (execv(MAPPER_PATH.data(), const_cast<char **>(argv)) != 0)
     ↪  {
      std::cerr << "Can't run mapper: " << errno << ": " <<
       ↪  strerror(errno)
                << std::endl;
      return 1;
    }
  }

  int status;
  auto w = waitpid(PID, &status, 0);
  if (w == -1) {
    std::cerr << "waitpid(...) error: " << errno << ": " <<
     ↪  strerror(errno)
              << std::endl;
    return 1;
  }
  if (WEXITSTATUS(status) != 0) {
    std::cerr << "mapper return code: " << WEXITSTATUS(status) <<
     ↪  std::endl;
    return 1;
  }
}
```

В листинге ниже приведено содержание sources/plotter/main.py (лист. 9):

Лист. 9 – Содержание sources/plotter/main.py

```python
import os
import re
import matplotlib.pyplot as plt
import pathlib
import typer

PATTERN = R"([^ ]{16})\s+([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^ ]+)\s+([^ \n]+)"


def main(
    plots_path: pathlib.Path,
    dumps_path: pathlib.Path = "/usr/share/operating_systems_laboratories_4/dumps/",
):
    for filename in os.listdir(dumps_path):
        path = os.path.join(dumps_path, filename)
        f = open(path, "r")
        text = "".join(f.readlines())
        text = text.replace("[ ", "").replace("] ", "")

        addresses = []
        sizes = []

        matches = re.findall(PATTERN, text)
        for match in matches[:-1]:
            addresses.append(match[0])
            sizes.append(int(match[1]))

        fig, ax = plt.subplots()
        ax.bar(addresses, sizes)
        plt.savefig(f"{plots_path}/{filename}.png")


if __name__ == "__main__":
    typer.run(main)
```

В листинге ниже приведено содержание CMakeLists.txt (лист. 10):

Лист. 10 – Содержание CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15...3.30)

project(
        operating_systems_laboratories_4
        VERSION 1.0
        LANGUAGES CXX
)
set(CMAKE_CXX_STANDARD 26)
```

```
9
10 add_library(common STATIC
11         sources/common/PidFile.cpp
12 )
13 target_include_directories(common INTERFACE
    ↪  ${CMAKE_CURRENT_SOURCE_DIR}/sources)
14
15 add_executable(leaker sources/leaker/main.cpp)
16 target_link_libraries(leaker common)
17
18 add_executable(mapper sources/mapper/main.cpp)
19
20 add_executable(monitor sources/monitor/main.cpp)
21 target_link_libraries(monitor common)
22
23 install(
24         DIRECTORY
25         DESTINATION
          ↪  /usr/share/operating_systems_laboratories_4/dumps
26         DIRECTORY_PERMISSIONS
27         OWNER_WRITE OWNER_READ
28 )
29
30 install(
31         TARGETS
32         leaker mapper monitor
33         DESTINATION
          ↪  /usr/share/operating_systems_laboratories_4/bin
34 )
35
36 install(
37         FILES install/operating_systems_laboratories_4.service
          ↪  install/operating_systems_laboratories_4.timer
38         DESTINATION /etc/systemd/system
39 )
```

В листинге ниже приведен пример карты памяти, сформированной программой в 20:27:36 (лист. 11).

Лист. 11 – Содержание /usr/share/operating_systems_laboratories_4/dumps/map_7748_2
11-27_20:27:36

```
1  7748:   ./leaker on 1
2  Адрес           Кб     RSS   Dirty Mode  Mapping
3  000057eb05f14000      12      12       0 r---- leaker
4  000057eb05f17000      20      20       0 r-x-- leaker
5  000057eb05f1c000      12      12       0 r---- leaker
6  000057eb05f1f000       4       4       4 r---- leaker
7  000057eb05f20000       4       4       4 rw--- leaker
8  000057eb06ed6000    4788    4676    4676 rw---   [ anon ]
9  00007171c6fff000    2052    1168    1168 rw---   [ anon ]
10 00007171c7200000     152     148       0 r---- libc.so.6
```

```
00007171c7226000    1532    1056       0 r-x-- libc.so.6
00007171c73a5000     340     144       0 r---- libc.so.6
00007171c73fa000      16      16      16 r---- libc.so.6
00007171c73fe000       8       8       8 rw--- libc.so.6
00007171c7400000      52      20      20 rw---   [ anon ]
00007171c7600000     624     624       0 r---- libstdc++.so.6.0.32
00007171c769c000    1220     884       0 r-x-- libstdc++.so.6.0.32
00007171c77cd000     564     136       0 r---- libstdc++.so.6.0.32
00007171c785a000      44      44      44 r---- libstdc++.so.6.0.32
00007171c7865000      12      12      12 rw--- libstdc++.so.6.0.32
00007171c7868000      16      12      12 rw---   [ anon ]
00007171c78ea000      20      20      20 rw---   [ anon ]
00007171c78ef000      12      12       0 r---- libgcc_s.so.1
00007171c78f2000     108      64       0 r-x-- libgcc_s.so.1
00007171c790d000      16      16       0 r---- libgcc_s.so.1
00007171c7911000       4       4       4 r---- libgcc_s.so.1
00007171c7912000       4       4       4 rw--- libgcc_s.so.1
00007171c7913000      64      60       0 r---- libm.so.6
00007171c7923000     512     240       0 r-x-- libm.so.6
00007171c79a3000     356       0       0 r---- libm.so.6
00007171c79fc000       4       4       4 r---- libm.so.6
00007171c79fd000       4       4       4 rw--- libm.so.6
00007171c7a27000       8       8       8 rw---   [ anon ]
00007171c7a29000       4       4       0 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a2a000     168     168       0 r-x--
 ↪  ld-linux-x86-64.so.2
00007171c7a54000      40      40       0 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a5e000       8       8       8 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a60000       8       8       8 rw---
 ↪  ld-linux-x86-64.so.2
00007fff652a0000     136      28      28 rw---   [ stack ]
00007fff652f8000      16       0       0 r----   [ anon ]
00007fff652fc000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
всего Кб    12976    9696    6052
```

В листинге ниже приведен пример карты памяти, сформированной программой в 20:42:55 (лист. **??**).

Лист. 12 – Содержание /usr/share/operating_systems_laboratories_4/dumps/map_7748_2
11-27_20:42:55

```
7748:   ./leaker on 1
Адрес           Кб     RSS   Dirty Mode  Mapping
000057eb05f14000      12      12       0 r---- leaker
000057eb05f17000      20      20       0 r-x-- leaker
000057eb05f1c000      12      12       0 r---- leaker
000057eb05f1f000       4       4       4 r---- leaker
```

```
000057eb05f20000       4       4       4 rw--- leaker
000057eb06ed6000  125700  125572  125572 rw---   [ anon ]
00007171c33fb000   32772   31392   31392 rw---   [ anon ]
00007171c7200000     152     148       0 r---- libc.so.6
00007171c7226000    1532    1056       0 r-x-- libc.so.6
00007171c73a5000     340     144       0 r---- libc.so.6
00007171c73fa000      16      16      16 r---- libc.so.6
00007171c73fe000       8       8       8 rw--- libc.so.6
00007171c7400000      52      20      20 rw---   [ anon ]
00007171c7600000     624     624       0 r---- libstdc++.so.6.0.32
00007171c769c000    1220     884       0 r-x-- libstdc++.so.6.0.32
00007171c77cd000     564     136       0 r---- libstdc++.so.6.0.32
00007171c785a000      44      44      44 r---- libstdc++.so.6.0.32
00007171c7865000      12      12      12 rw--- libstdc++.so.6.0.32
00007171c7868000      16      12      12 rw---   [ anon ]
00007171c78ea000      20      20      20 rw---   [ anon ]
00007171c78ef000      12      12       0 r---- libgcc_s.so.1
00007171c78f2000     108      64       0 r-x-- libgcc_s.so.1
00007171c790d000      16      16       0 r---- libgcc_s.so.1
00007171c7911000       4       4       4 r---- libgcc_s.so.1
00007171c7912000       4       4       4 rw--- libgcc_s.so.1
00007171c7913000      64      60       0 r---- libm.so.6
00007171c7923000     512     240       0 r-x-- libm.so.6
00007171c79a3000     356       0       0 r---- libm.so.6
00007171c79fc000       4       4       4 r---- libm.so.6
00007171c79fd000       4       4       4 rw--- libm.so.6
00007171c7a27000       8       8       8 rw---   [ anon ]
00007171c7a29000       4       4       0 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a2a000     168     168       0 r-x--
 ↪  ld-linux-x86-64.so.2
00007171c7a54000      40      40       0 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a5e000       8       8       8 r----
 ↪  ld-linux-x86-64.so.2
00007171c7a60000       8       8       8 rw---
 ↪  ld-linux-x86-64.so.2
00007fff652a0000     136      28      28 rw---   [ stack ]
00007fff652f8000      16       0       0 r----   [ anon ]
00007fff652fc000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
всего Кб   164608  160816  157172
```

На рисунке ниже приведен график, сформированный для карты памяти, сформированной программой в 20:27:36 (рис. 1):

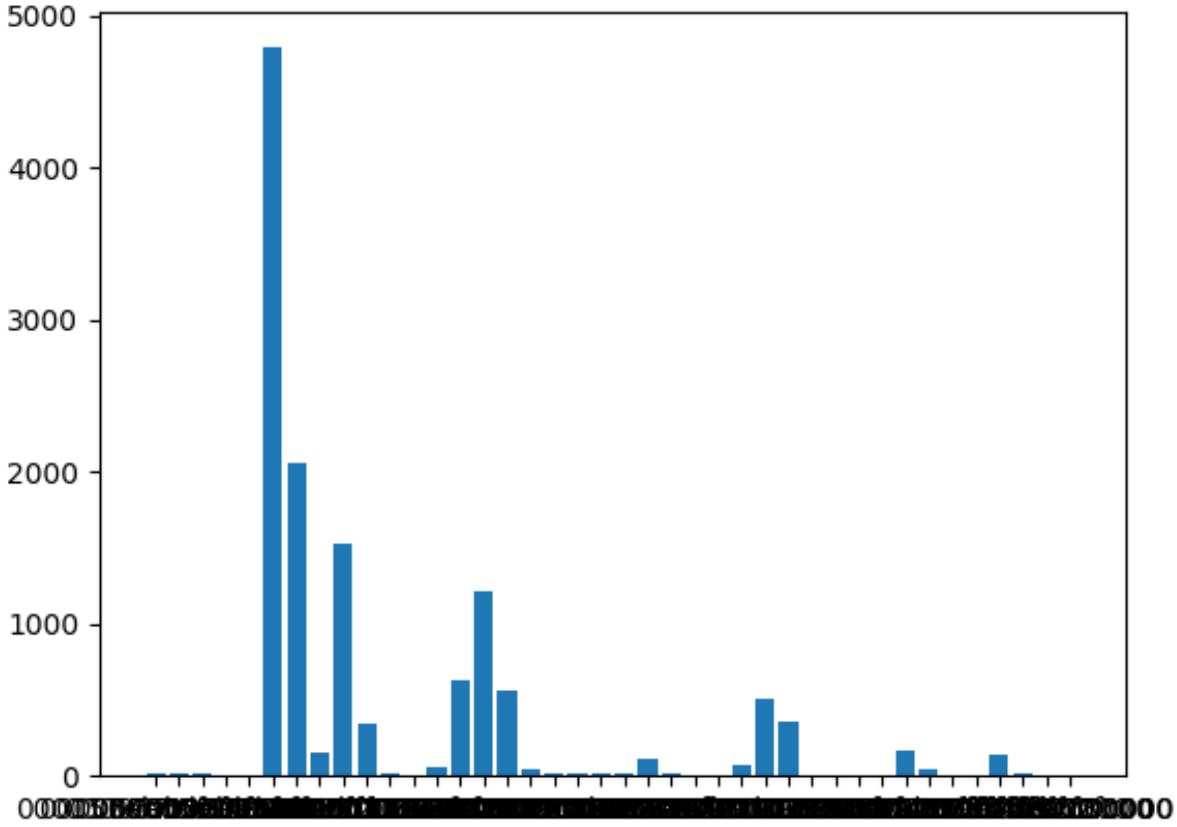

Рис. 1 – График, сформированный для карты памяти, сформированной программой в 20:27:36

На рисунке ниже приведен график, сформированный для карты памяти, сформированной программой в 20:42:55 (рис. 2):

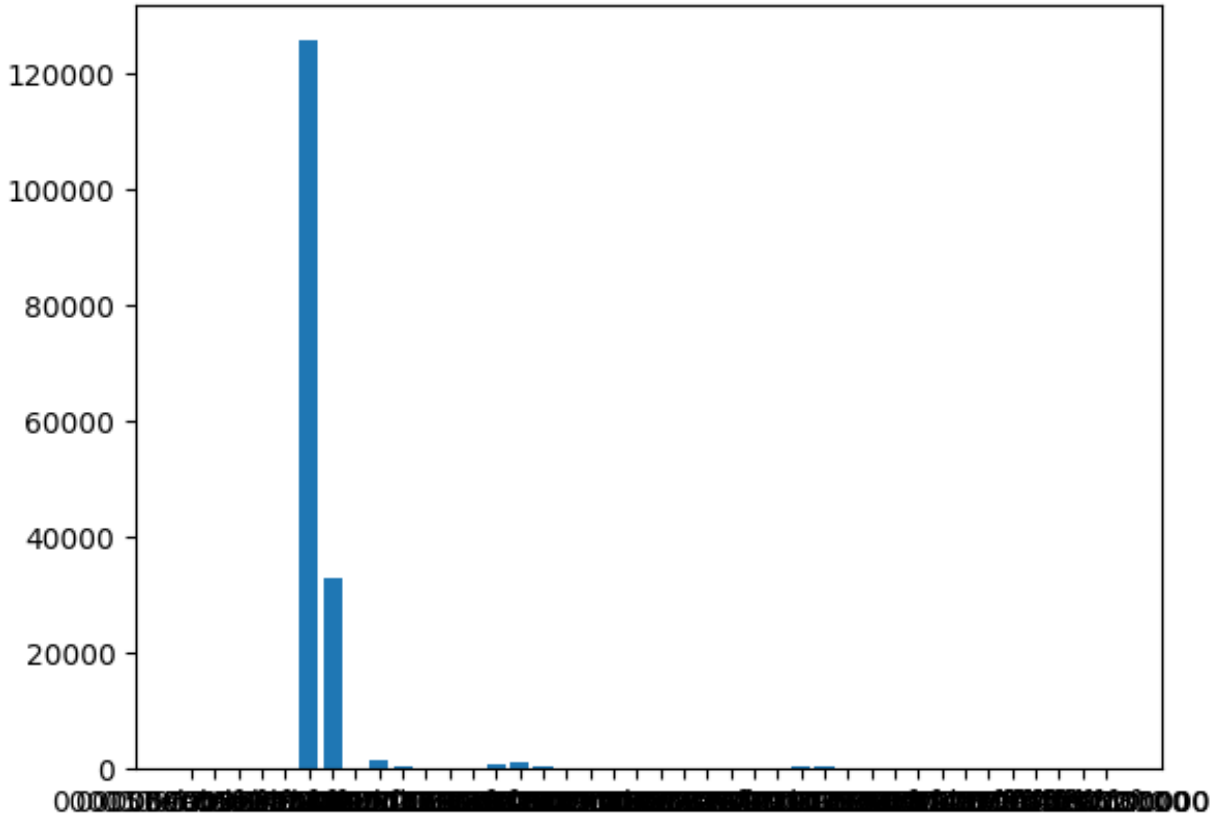

Рис. 2 – График, сформированный для карты памяти, сформированной программой в 20:42:55

4 Ответы на контрольные вопросы

1. Стек — структура данных, представляющая собой список элементов, организованных по принципу LIFO. В контексте архитектуры ПК стек (аппаратный) — непрерывная область памяти, адресуемая специальными регистрами ESP и SS.

Куча — структура данных, поддерживающая операции добавления и удаления элемента, а также нахождения минимального элемента. В контексте архитектуры ПК — условное название области динамической памяти.

2. Статическое выделение памяти происходит благодаря вычитанию из ESP необходимого значения байт. Динамическое выделения памяти происходит благодаря обращению к malloc, который аллоцирует необходимое количество памяти и возвращает пользователю. На Linux динамическое выделение памяти реализовано с помощью системных вызовов sbrk или mmap. С помощью них malloc выделяет память, а потом управляет ей согласно соображениям оптимальности и прочим для уменьшения фрагментации и так далее.

3. Вопрос некорректен. Статическое выделение памяти используется для одних вещей, динамическое выделение памяти используется для других вещей. Если объем необходимой памяти известен статически и нет требований к тому, чтобы вре-

мя жизни выделенной памяти распространялось за пределы вызова данной функции, разумно и целесообразно использовать статическое выделение памяти. Динамическое выделение памяти используется в иных случаях, когда невозможно или нецелесообразно использовать статическое выделение памяти. Статическое выделение и освобождение памяти происходит быстрее, чем динамическое.

Заключение

Таким образом, в ходе выполнения данной работы были успешно изучены системные функции выделения памяти в ОС GNU/LINUX, получены практические навыки работы с динамической памятью в ОС GNU/LINUX.