

Московский авиационный институт
(национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: К.С. Саженов
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19
Дата:
Оценка:
Подпись:

Москва 2020

Лабораторная работа №3

Задача: Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

1. Введение

Разработка современного ПО включает в себя большое количество различных процессов. Один из таких процессов, который помогает понять, какое конкретное место является, т. н., `bottleneck` (англ. горлышко бутылки) — узким местом программы, является анализ производительности программ, а именно `profiling` (англ. Профилирование) — позволяет узнать, какие участки кода занимают больше всего процессорного/пользовательского времени, `coverage` (англ. покрытие) — позволяет узнать, какие части кода сколько раз выполняются, а также `memcheck` (англ. Проверка памяти) — позволяет определить утечки памяти. На примере программы из предыдущей лабораторной работы — реализация словаря с использованием `PATRICIA Trie` — я проведу детальный анализ производительности данной программы.

2. Анализ времени исполнения

Valgrind(callgrind) & kcachegrind

При выборе утилиты я хотел найти наиболее многофункциональную и удобную для пользования новичкам программу. Эти два критерия совместили в себе две программы соответственно — *valgrind* и *kcachegrind*(содержится в среде рабочего стола KDE). Valgrind включает в себя несколько утилит(некоторые из них будут рассмотрены далее), одной из которых является *callgrind*.

Используя утилиту *kcachegrind*, данный симбиоз программ позволяет крайне детально оценивать скорость выполнения тех или иных мест в программе. Одними из основных полезных функций для визуализации вызовов в данной графической утилите является построение графов вызовов и карты вызовов.

Первая позволяет проследить вызовы, а вторая оценить время выполнения каждого вызова.

Однако стоит понимать, что указанную программу следует компилировать

с флагом компиляции(`-g`)

Запустим профайлер следующим образом:

```
└─sakost@sakost-pc ~/university/2 course/diskran/lab2
```

```
└─$ valgrind --tool=callgrind --dump-instr=yes --simulate-cache=yes --collect-jumps=yes cmake-build-debug/lab2 < data.txt
```

Сгенерировался файл `callgrind.out.XXX` — откроем его:

```
└─sakost@sakost-pc ~/university/2 course/diskran/lab2
```

```
└─$ kcachegrind callgrind.out.115127
```

Наблюдаем интерфейс программы:

Файл

Вид

Переход

Настройка

Справка

Открыть...

Назад

Вперед

Вверх

% Относительно

Определять циклы

Относительно родителей

Сокращать шаблоны

Instruction Fetch

Плюсик.профиль

Search: Search Query

(No Grouping)

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000002090	ld-2.32.so
78.86	0.03	1	_dl_start	ld-2.32.so
78.84	0.02	1	_dl_symdep_start	ld-2.32.so
78.19	0.04	1	_dl_main	ld-2.32.so
76.73	11.76	7	_dl_relocate_object	ld-2.32.so
65.57	36.32	3 218	_dl_lookup_symbol_x	ld-2.32.so
29.26	22.21	3 218	_do_lookup_x	ld-2.32.so
19.11	0.00	1	_start	lab2
19.11	0.00	1	(below main)	libc-2.32.so
17.79	0.27	1	_main	lab2: main.cp
6.96	4.50	3 182	check_match	ld-2.32.so
3.04	0.98	308	std::basic_istream<char, st...	libstdc++ + so.6.0.28
2.78	2.78	3 770	strcmp	ld-2.32.so
2.78	0.18	15	TPatricia::SerializeStd::bas...	lab2: patricia.h
2.70	0.37	323	malloc	libc-2.32.so
2.55	0.28	14	TPatricia::DeserializeStd::b...	lab2: patricia.h
2.32	0.03	29	_dl_runtime_resolve_xsave	ld-2.32.so
2.23	0.53	518	std::ostream::write(char co...	libstdc++ + so.6.0.28
2.03	0.00	1	_dl_init	ld-2.32.so
2.02	0.01	7	call_init.part.0	ld-2.32.so
1.93	0.00	1	_GLOBAL_sub_1_eh_alloc.cc	libstdc++ + so.6.0.28
1.93	0.00	1	malloc_hook_ini	libc-2.32.so
1.91	0.02	1	_ptmalloc_init.part.0	libc-2.32.so
1.89	1.88	1	_dl_addr	libc-2.32.so
1.88	0.46	479	std::istream::read(char, lo...	libstdc++ + so.6.0.28
1.75	0.80	686	std::basic_filebuf<char, st...	libstdc++ + so.6.0.28
1.38	0.05	39	std::istream::std::istream(...)	libstdc++ + so.6.0.28
1.36	0.16	316	std::ctype<char>::const&...	libstdc++ + so.6.0.28
1.35	0.54	396	_dynamic_cast	libstdc++ + so.6.0.28
1.34	0.00	39	std::istream::operator<<(u_...	libstdc++ + so.6.0.28
1.27	0.00	39	std::num_get<char, std::ist...	libstdc++ + so.6.0.28
1.26	0.18	39	std::istreambuf_iterator<u_...	libstdc++ + so.6.0.28
1.26	0.15	39	TPatricia::AddItem(char, u_...	lab2: patricia.h
1.24	0.00	1	_libc_csu_init	lab2
1.24	0.00	1	_GLOBAL_sub_1_Z4copylib	lab2: main.cp
1.24	0.00	1	_static_initialization_and_f...	lab2: main.cp
1.10	0.69	164	std::basic_ofstream<char, s...	libstdc++ + so.6.0.28
1.08	0.01	163	std::istream::operator<<(c...	libstdc++ + so.6.0.28
1.07	0.41	480	std::basic_filebuf<char, s...	libstdc++ + so.6.0.28
1.00	0.02	36	std::locale::localize()	libstdc++ + so.6.0.28
0.99	0.01	36	std::locale::_S_initialize()	libstdc++ + so.6.0.28
0.98	0.00	1	std::locale::_S_initialize_1	libstdc++ + so.6.0.28
0.98	0.02	1	std::locale::impl::implfun...	libstdc++ + so.6.0.28
0.91	0.08	163	std::basic_ofstream<char, s...	libstdc++ + so.6.0.28
0.91	0.91	828	std::istream::sentry::sentr...	libstdc++ + so.6.0.28
0.89	0.62	686	std::basic_streambuf<char, ...	libstdc++ + so.6.0.28
0.81	0.08	292	operator new(unsigned long)	libstdc++ + so.6.0.28
0.78	0.01	15	TPatricia::~Patricia()	lab2: patricia.h
0.78	0.02	15	TPatricia::Destruct(TNode*)	lab2: patricia.h
0.76	0.22	164	std::basic_ofstream<char, s...	libstdc++ + so.6.0.28

calling out: 115127 [1] - Bcero Instruction Fetch CPMOMCITY 3 846 768

Types

Callers

All Callers

Callee Map

Source Code

lr	lr per call	Count	Caller
17.79	684	471	1 (below main) (libc-2.32.so)
3.04	379	308	15 std::basic_istream<char, std::char_traits<char>>, >6, char*> (>std::basic_istream<char, std::char_traits<char>>, >6, char*) (libstdc++ + so.6.0.28: istream.cc, ...)
2.78	128	15	TPatricia::SerializeStd::basic_ofstream<char, std::char_traits<char>>, >6) (lab2: patricia.h)
2.55	7 000	14	TPatricia::DeserializeStd::basic_ofstream<char, std::char_traits<char>>, >6) (lab2: patricia.h)
1.34	1 360	39	std::istream::operator<<(unsigned long)& (<libstdc++ + so.6.0.28: istream)
1.26	1 243	39	std::AddItem(char*, unsigned long) (<libstdc++ + so.6.0.28: istream)
1.08	254	163	std::ostream::operator<<(std::ostream&) (<libstdc++ + so.6.0.28: ostream)
0.91	215	163	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (<libstdc++ + so.6.0.28: ostream)
0.89	1 893	18	_dl_runtime_resolve_xsave (ld-2.32.so)
0.78	2 002	15	TPatricia::~Patricia() (lab2: patricia.h)
0.50	142	135	KeyToLower(char*) (lab2: main.cp)
0.44	364	47	TPatricia::Erase(char*) (lab2: patricia.h)
0.44	3	49	TPatricia::Find(char const*) const (lab2: patricia.h)
0.31	856	14	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>open(char const*, std::ios_OpenMode) (libstdc++ + so.6.0.28: ostream, ...)
0.29	804	14	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>close() (libstdc++ + so.6.0.28: ostream)
0.13	645	39	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>open(char const*, std::ios_OpenMode) (libstdc++ + so.6.0.28: ostream, ...)
0.23	695	13	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>close() (libstdc++ + so.6.0.28: ostream)
0.22	21	409	_strcmp_avx2 (libc-2.32.so)
0.04	111	15	std::Patricia() (lab2: patricia.h)
0.04	90	15	operator delete(void*, unsigned long) (libstdc++ + so.6.0.28: del_ops.cc)
0.03	570	2	std::ostream::operator<<(unsigned long) (libstdc++ + so.6.0.28: ostream)
0.02	55	14	operator new(unsigned long) (libstdc++ + so.6.0.28: new_op.cc)
0.02	4	164	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>operator bool() (libstdc++ + so.6.0.28: basic_ios.h)
0.01	0	44	std::operator(std::ios_OpenMode, std::ios_OpenMode) (lab2: ios_base.h)
n/no	n	4	std::basic_ofstream<char, std::char_traits<char>>, >6, char*> (>close() (libstdc++ + so.6.0.28: basic_ios.h)

Exit

Call Graph

All Callees

Caller Map

Machine Code

Слева отображается таблица функций, отранжированных по умолчанию по времени выполнения по убыванию. Нас интересуют функции, написанные нами — возможно, анализ данных функций, позволит нам понять, каким образом мы можем оптимизировать нашу программу. Выбрав одну из функций, например `TPatricia::Serialize`, справа можно наблюдать целый набор инструментов, предоставляющий подробную информацию о ней:

Файл Вид Переход Настройка Справка

Открыть... < Назад > Вперед > ^ Верх % Отображение Определять циклы Относительно родителей Сокращать шаблоны Instruction Fetch

Плюсик профиля

Search: Search Query (No Grouping)

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000209	ld-2.32.so
78.86	0.03	1	_dl_start	ld-2.32.so
78.84	0.02	1	_dl_sysdep_start	ld-2.32.so
78.19	0.04	1	_dl_main	ld-2.32.so
76.73	11.76	7	_dl_relocate_object	ld-2.32.so
65.57	36.32	3 218	_dl_lookup_symbol_x	ld-2.32.so
29.26	22.21	3 218	_do_lookup_x	ld-2.32.so
19.11	0.00	1	_start	lab2
19.11	0.00	1	[below main]	libc-2.32.so
17.79	0.27	1	main	lab2: main.cp
6.96	4.50	3 182	check_match	ld-2.32.so
3.04	0.98	308	std::basic_ostream<char, st...	libstdc++ so
2.78	3.770	1	strcmp	ld-2.32.so
2.78	0.18	15	T Patricia::Serialize(std::bas...	lab2: patricia
2.70	0.37	323	malloc	libc-2.32.so
2.55	0.28	14	T Patricia::Deserialize(std:b...	lab2: patricia
2.32	0.03	29	_dl_runtime_resolve_xsave	ld-2.32.so
2.23	0.53	518	std::ostream::write(char co...	libstdc++ so
2.03	0.00	1	_dl_init	ld-2.32.so
2.02	0.01	7	call_init.part.0	ld-2.32.so
1.93	0.00	1	_GLOBAL__sub_i eh_alloc.cc	libstdc++ so
1.93	0.00	1	malloc_hook_ini	libc-2.32.so
1.91	0.02	1	ptmalloc_init.part.0	libc-2.32.so
1.89	1.88	1	_dl_addr	libc-2.32.so
1.88	0.46	479	std::istream::read(char*, lo...	libstdc++ so
1.75	0.80	686	std::basic_filebuf<char, std...	libstdc++ so
1.38	0.05	39	std::istream::std::istrea...	libstdc++ so
1.36	0.16	316	std::ctype<char>.const&...	libstdc++ so
1.35	0.54	396	__dynamic_cast	libstdc++ so
1.34	0.00	38	std::istream::operator<<(...	libstdc++ so
1.27	0.00	39	std::num_get<char, std::is...	libstdc++ so
1.26	1.18	39	std::istreambuf_iterator<...	libstdc++ so
1.26	0.15	39	T Patricia::AddItem(char*, ...	lab2: patricia
1.24	0.00	1	_libc_sos_init	lab2
1.24	0.00	1	_GLOBAL__sub_i _Z4CopyCp...	lab2: main.cp
1.24	0.00	1	__static_initialization_and...	lab2: main.cp
1.10	0.09	164	std::basic_ostream<char, s...	libstdc++ so
1.08	0.01	163	std::ostream::operator<<(...	libstdc++ so
1.07	0.41	480	std::basic_filebuf<char, std...	libstdc++ so
1.00	0.02	36	std::locale::locale()	libstdc++ so
0.99	0.01	36	std::locale::_S_initialize()	libstdc++ so
0.98	0.00	1	std::locale::_S_initialize_on...	libstdc++ so
0.98	0.02	1	std::locale::_impl::_implfun...	libstdc++ so
0.91	0.08	163	std::basic_ostream<char, s...	libstdc++ so
0.91	0.91	828	std::istream::sentry::strea...	libstdc++ so
0.89	0.62	686	std::basic_streambuf<char,...	libstdc++ so
0.81	0.08	292	operator new(unsigned long)	libstdc++ so
0.78	0.01	15	T Patricia::~TPatricial()	lab2: patricia
0.78	0.02	15	T Patricia::Destruct(TNode*)	lab2: patricia
0.76	0.22	164	std::basic_ostream<char, s...	libstdc++ so

Types Callers All Callers Callee Map Source Code

lr	lr per call	Count	Caller
2.78	7 128	15	main (lab2: main.cpp)
2.23	165	518	std::ostream::write(char const*, long) (libstdc++ so 6.0.28: ostream.tcc, ...)
0.21	541	15	T Patricia::GenerateId(TNode*, TNode**, int&) (lab2: patricia.h)
0.05	120	15	operator new[](unsigned long) (libstdc++ so 6.0.28: new_opvc.c)
0.04	858	2	_dl_runtime_resolve_xsave (ld-2.32.so)
0.03	17	69	_strlen_avx2 (libc-2.32.so)
0.03	85	14	operator delete[](void*) (libstdc++ so 6.0.28: del_opvc.c)

Parts Callees Call Graph All Callees Caller Map Machine Code

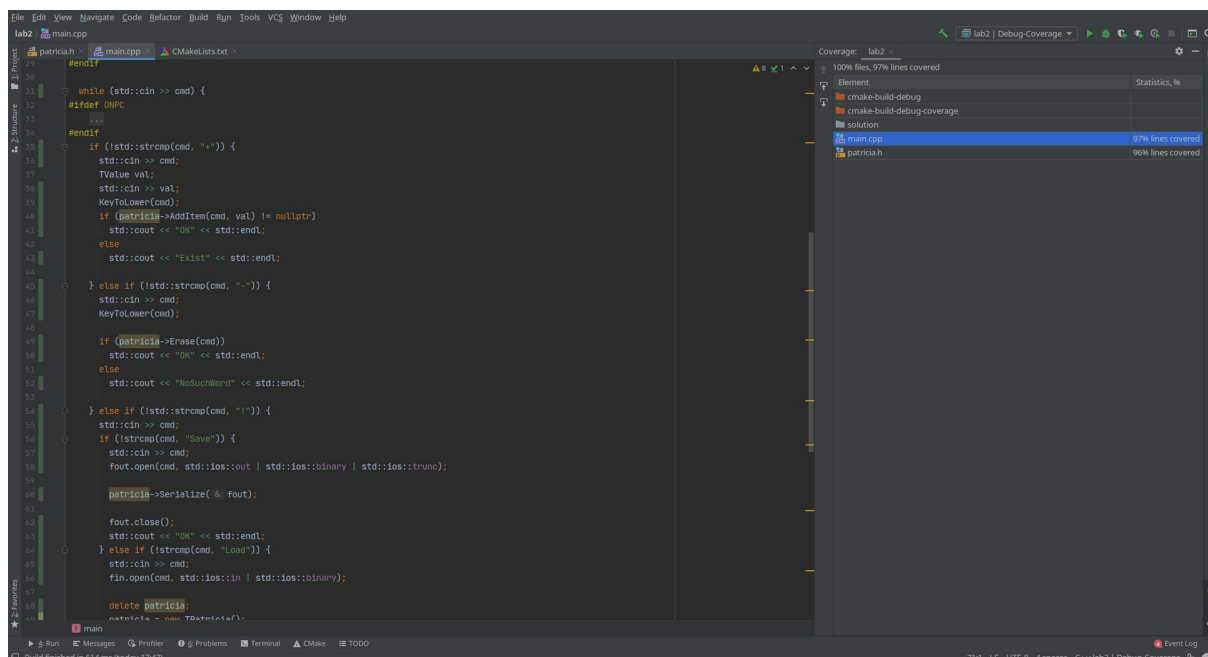
called out 115127 [1] - Born Instruction Fetch Crossover, 3 846 768

Одни из самых хорошо интерпретируемых полей здесь, как и упоминалось выше, callee map и call graph. Судя по данным графикам, наиболее «долгая» функция — встроенная функция записи в файл. Её оптимизировать, к сожалению, на данном этапе моего обучения не предоставляется возможным.

3. Анализ покрытия кода

1. gcov

Одной из основной программой для проверки кода на покрытие в IDE CLion является утилита gcov. При помощи среды разработки можно в режиме написания кода просмотреть, какие строки сколько раз исполнялись (красные — 0, желтые — частично вызывается, зеленые — исполняется постоянно):

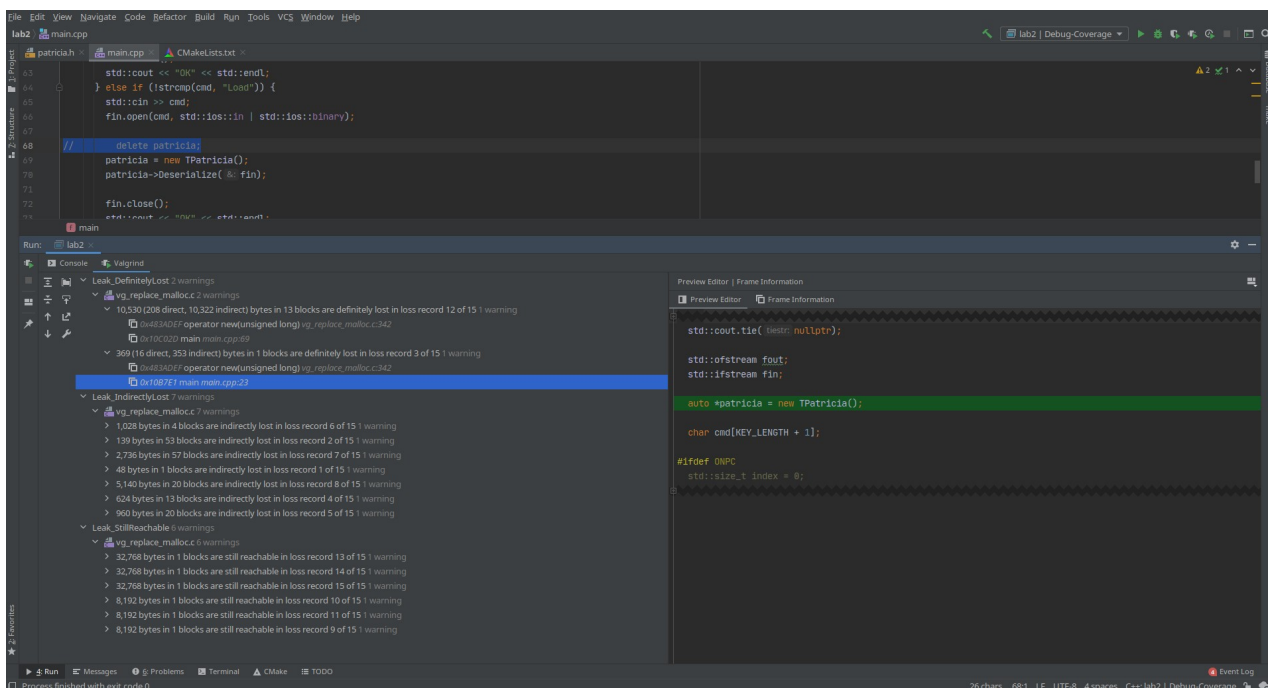


Делаем вывод, что в нашем тесте обрабатываются все возможные случаи, которые возможны на чекере. Следовательно, если ошибок нет, значит наша программа работает корректно для всех случаев входных данных.

4. Анализ потребления памяти

1. Valgrind

В основном мире C/C++ для отслеживания утечек памяти используется такая утилита, как Valgrind(memcheck). Она содержит в себе, как и упоминалось выше, несколько инструментов для анализа программ и memcheck — ещё один из них. Valgrind(memcheck) является довольно надёжным инструментом, указывающий иногда на, порой, незначительные ошибки, которые даже не приводят к утечкам(но потенциально могли бы). Интерфейс CLion, который использует valgrind memcheck в качестве инструмента для анализа утечек памяти(я закомментировал выделенный код — будет явная утечка):



Данная утилита верно определила какая часть памяти не освободилась(а за ней ещё несколько, т. к. при удалении данного объекта освобождаются и другие области памяти)

2. heaptrack

Имеет схожий с valgrind memcheck функционал и интуитивно понятный интерфейс. Чтобы просмотреть данные по программе, нужно её сначала отдать консольной версии программы:

```
~sakost@sakost-pc ~/university/2 course/diskran/lab2
```

```
$ heaptrack cmake-build-debug/lab2 < data.txt
```

...

/build/heaptrack/src/heaptrack-1.2.0/src/interpret/heaptrack_interpret.cpp:351

ERROR:Failed to create backtrace state for module /home/sakost/university/2:

/home/sakost/university/2 / No such file or directory (error code 2)

heaptrack stats:

allocations: 323

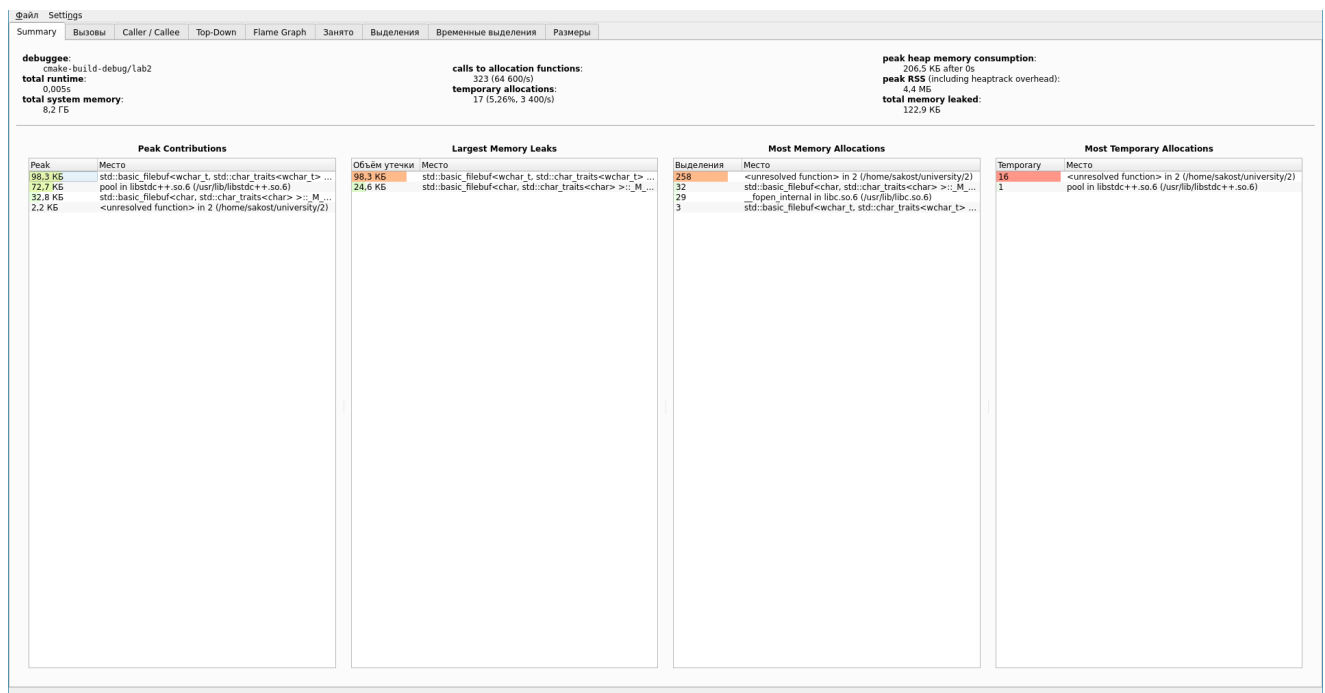
leaked allocations: 6

temporary allocations: 15

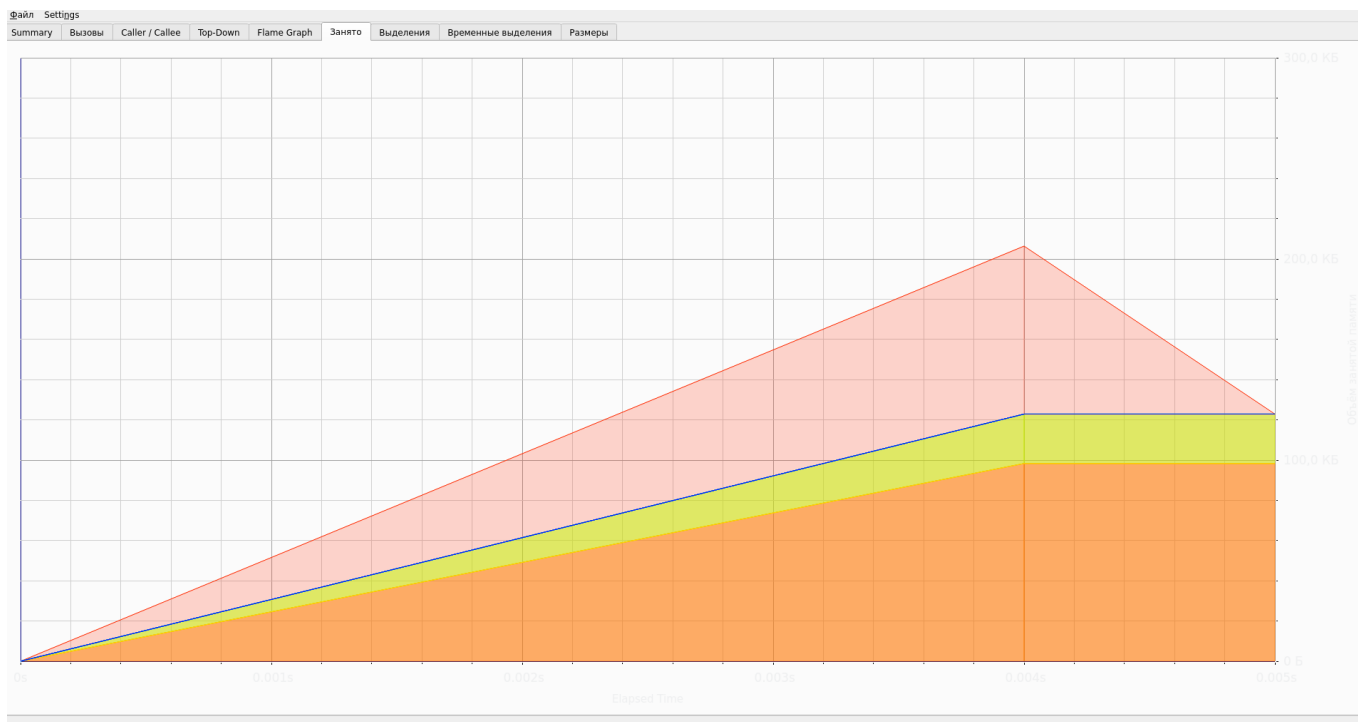
Heaptrack finished! Now run the following to investigate the data:

```
heaptrack --analyze "/home/sakost/university/2  
course/diskran/lab2/heaptrack.lab2.177357.zst"
```

И запускаем то, что нам выдала консольная утилита. Запустилась графическая версия программы, вот её интерфейс:



Тут также можно посмотреть красивые графики:



5. Вывод

В этой лабораторной работе я проанализировал код программы и саму программу, которую я написал в прошлой лабораторной работе. Я изучил несколько способов анализа поведения программы, несколько утилит, помогающих выловить те или иные ошибки. Как мне показалось, самым функциональным и одним из самых простых в использовании инструментов оказался `valgrind`. Также, попробовав поизучать `heaptrack`, я понял, что этот инструмент также заслуживает внимания программистов, хоть он и не является тем инструментом, которым в основном привыкли использовать программисты.

Борьба с ошибками, утечками и багами — это неотъемлемая часть разработки практически любого ПО. Ни одна сложная в написании программа, используемая в повседневной жизни людьми, не была написана сразу без ошибок и для решения проблем использовались, в частности, инструменты, изученные мной. Данные утилиты помогут мне в будущем более оперативно и с меньшими усилиями находить критические части программ.