Лабораторная работа №3 по курсу Дискретного Анализа: Исследование качества программ

Выполнил студент группы М8О-207Б-21 МАИ - *Кажекин Денис*.

Условие

Для реализации словаря из предыдущей лабораторной работы необходимо провести исследование скорости выполнения и потребления оперативной памяти.

Метод решения

Изучение утилит для исследования качества программ таких как gcov, gprof, valgrind, и их использование для оптимизации программы.

VALGRIND

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

В ходе выполнения лабораторной работы утилита будет использована исключительно для отладки использования памяти.

==5813== HEAP SUMMARY:

==5813== in use at exit: 43,752 bytes in 13 blocks

==5813== total heap usage: 22 allocs, 9 frees, 126,264 bytes allocated

==5813==

==5813== 216 bytes in 1 blocks are definitely lost in loss record 2 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C987: Node::Node() (main.cpp:33)

==5813== by 0x10AF0F: insert(Node**, char*, unsigned long long, bool) (main.cpp:410)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 216 bytes in 1 blocks are definitely lost in loss record 3 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C987: Node::Node() (main.cpp:33)

==5813== by 0x10A73B: splitChild(Node*, Node*) (main.cpp:293)

==5813== by 0x10AE93: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:397)

==5813== by 0x10AFBF: insert(Node**, char*, unsigned long long, bool) (main.cpp:422)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 648 bytes in 3 blocks are definitely lost in loss record 4 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C987: Node::Node() (main.cpp:33)

==5813== by 0x10A73B: splitChild(Node*, Node*) (main.cpp:293)

==5813== by 0x10AEA3: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:400)

==5813== by 0x10AE27: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:385)

==5813== by 0x10AFBF: insert(Node**, char*, unsigned long long, bool) (main.cpp:422)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 7,072 bytes in 1 blocks are definitely lost in loss record 6 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C973: Node::Node() (main.cpp:32)

==5813== by 0x10AF0F: insert(Node**, char*, unsigned long long, bool) (main.cpp:410)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 7,072 bytes in 1 blocks are definitely lost in loss record 7 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C973: Node::Node() (main.cpp:32)

==5813== by 0x10A73B: splitChild(Node*, Node*) (main.cpp:293)

==5813== by 0x10AE93: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:397)

==5813== by 0x10AFBF: insert(Node**, char*, unsigned long long, bool) (main.cpp:422)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 7,312 (24 direct, 7,288 indirect) bytes in 1 blocks are definitely lost in loss record 8 of 9

==5813== at 0x48657B8: operator new(unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10AE53: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:393)

==5813== by 0x10AFBF: insert(Node**, char*, unsigned long long, bool) (main.cpp:422)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== 21,216 bytes in 3 blocks are definitely lost in loss record 9 of 9

==5813== at 0x4866AE8: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)

==5813== by 0x10C973: Node::Node() (main.cpp:32)

==5813== by 0x10A73B: splitChild(Node*, Node*) (main.cpp:293)

==5813== by 0x10AEA3: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:400)

==5813== by 0x10AE27: findNode(Node*, Node*, char*, unsigned long, Node**, bool) (main.cpp:385)

==5813== by 0x10AFBF: insert(Node**, char*, unsigned long long, bool) (main.cpp:422)

==5813== by 0x10B5E7: main (main.cpp:541)

==5813==

==5813== LEAK SUMMARY:

==5813== definitely lost: 36,464 bytes in 11 blocks

==5813== indirectly lost: 7,288 bytes in 2 blocks

==5813== possibly lost: 0 bytes in 0 blocks

==5813== still reachable: 0 bytes in 0 blocks

==5813== suppressed: 0 bytes in 0 blocks

Как можно увидеть в логе, Valgrind обнаружил много утечек памяти (36,464 байта), а также показывает, что это связано с конструктором, который вызывается при вставке узла: Node::Node() на строчке (main.cpp:33) . Для устранения утечек необходимо после завершения работы моего дерева, рекурсивно его обойти и удалить выделенную память под ключи и сыновей узлов. Для этого я написал деструктор структуры ~Node().

==6453==

==6453== HEAP SUMMARY:

==6453== in use at exit: 0 bytes in 0 blocks

==6453== total heap usage: 22 allocs, 22 frees, 126,264 bytes allocated

==6453==

==6453== All heap blocks were freed -- no leaks are possible

==6453==

==6453== For lists of detected and suppressed errors, rerun with: -s

==6453== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Теперь можем увидеть, что все утечки устранены.

GPROF

Gprof - это инструмент для профилирования программы. Мы можем отследить, где и сколько времени проводила программа, тем самым выявляя слабые участки.

Возьмем достаточно большой тест и применим утилиту gprof.

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
50.00	0.02	0.02	290130	68.93	68.93	s_bin_search(int, int, char*, Node*)
25.00	0.03	0.01	100000	100.00	300.00	search(Node*, char*)
25.00	0.04	0.01				_init
0.00	0.04	0.00	500001	0.00	0.00	std::basic_istream<char, std::char_traits<char> >& std::operator>><char, std::char_traits<char> >(std::basic_istream<char, std::char_traits<char> >&, char*)
0.00	0.04	0.00	300000	0.00	0.00	toLower(char*)
0.00	0.04	0.00	289426	0.00	0.00	l_bin_search(int, int, char*, Node*)
0.00	0.04	0.00	201568	0.00	0.00	findKey(Node*, char*)
0.00	0.04	0.00	100585	0.00	0.00	deletion(Node*, char*)
0.00	0.04	0.00	100000	0.00	0.00	m_deletion(Node**, char*)
0.00	0.04	0.00	100000	0.00	0.00	insert(Node**, char*, unsigned long long, bool)
0.00	0.04	0.00	99999	0.00	0.00	findNode(Node*, Node*, char*, unsigned long, Node**, bool)
0.00	0.04	0.00	1049	0.00	0.00	removeFromLeaf(Node*, int)
0.00	0.04	0.00	585	0.00	0.00	removeFromNonLeaf(Node*, int)
0.00	0.04	0.00	536	0.00	0.00	getPredecessor(Node*, int)
0.00	0.04	0.00	491	0.00	0.00	fill_(Node*, int)
0.00	0.04	0.00	290	0.00	0.00	borrowFromPrev(Node*, int)
0.00	0.04	0.00	153	0.00	0.00	borrowFromNext(Node*, int)

0.00	0.04	0.00	72	0.00	0.00	Node::Node()
0.00	0.04	0.00	72	0.00	0.00	Node::~Node()
0.00	0.04	0.00	69	0.00	0.00	splitChild(Node*, Node*)
0.00	0.04	0.00	64	0.00	0.00	mergez(Node*, int)
0.00	0.04	0.00	33	0.00	0.00	getSuccessor(Node*, int)
0.00	0.04	0.00	1	0.00	0.00	deleteTree(Node*)
0.00	0.04	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)

Из лога можем увидеть, что большую часть времени программа проводит в функциях бинпоиска индекса, куда вставить элемент и в функции поиска элемента в дереве, чтобы сказать, есть он там или нет.

Выводы

Я познакомился с очень полезными инструментами:

1. Valgrind позволяет выявлять утечки памяти и профилировать код. Инструмент оказалася довольно не сложным и удобным.

2. gprof позволяет оценить производительность программы, выявляя слабые места в плане производительности.

Инструменты оказались очень полезными. Лабораторная работа помогла исправить утечки в памяти и понять, в какой части программа выполняется больше всего.