Gem #79: Куда делась моя память? (Часть 3)

Автор: Emmanuel Briot, AdaCore

Краткое содержание: Gem Ada #79 - Для контроля за использованием памяти, обнаружением утечек памяти и разрешения проблем общего характера, связанных с памятью, существует множество инструментов и библиотек. Эта серия из трёх публикаций (Gem #77,#78,#79) содержит обзор этой тематики и описание работы с подобными инструментами.

Давайте начнём…

Если ваш стандарт кодирования не запрещает динамическое выделение, управление памятью является постоянной проблемой при разработке системы. Может потребоваться ограничить объем памяти, необходимый приложению, или могут возникнуть утечки памяти (блоки выделения, которые никогда не возвращаются в систему). Последнее является критической проблемой для долговременных приложений.

Часть III: Инструментарий поиска утечки памяти других производителей программных средств

В предыдущих двух публикациях Gems (#77, #78), посвящённых памяти, объяснялось использование `GNAT.Debug_Pools` и `GNATCOLL.Memory` для контроля доступа к памяти и обнаружения утечек памяти.

В некоторых системах существуют очень мощные внешние инструменты, которые можно использовать для достижения аналогичных целей. Очень часто эти инструменты просто заменяют системные вызовы `malloc` и бесплатны, хотя в некоторых случаях они фактически будут эмулировать виртуальную машину, на которой выполняется ваше приложение.

Один из распространённых видов инструментов, доступных в большинстве систем, показывает, сколько памяти использует ваше приложение (например, диспетчер процессов (`Process Manager`) в Windows или «`top`» в системах Unix). Это, однако, очень ограниченный инструмент, поскольку память, которая должным образом освобождена вашим приложением, фактически не может быть возвращена системе (по соображениям производительности `malloc` сохраняет её и повторно использует для следующего выделения). Поэтому очень трудно обнаружить утечки памяти, таким образом, и, конечно, даже если вы увидите, что есть утечка, вы не можете узнать, где находится ваш код, который вызывает утечку памяти.

Одним из очень полезных приложений для Linux является `valgrind`. Это виртуальная машина, которую вы запускаете с различных инструментов. Один из них - это средство проверки памяти (memory checker), которое может обнаруживать недопустимые обращения к памяти, двойное освобождение, использование неинициализированных переменных и утечки памяти. Вам не нужно ничего делать при компиляции приложения, и вы можете просто запустить его следующим образом:

```
valgrind --tool=memcheck your_app app_arguments
```

Если Вы также хотите обнаружить утечки памяти, запустите Ваше приложение следующим образом:

```
valgrind --tool=memcheck --leak-check=full --leak-resolution=med your_app
```

По сравнению с методами, которые мы обсуждали в предыдущих Gems, `valgrind` намного медленнее (поскольку весь код работает на виртуальной машине), но более точный. Например, когда он сообщает об утечке памяти, он будет сообщать только те блоки памяти, которые больше не ссылаются нигде в вашем приложении. Например, если у вас есть глобальная константа,

инициализированная один раз, вызывая «new ...», пакеты Ada сообщают об этом, идентифицируя это как утечку, тогда как valgrind по умолчанию знает, что она по-прежнему ссылается и не будет вас беспокоить (хотя, конечно, у вас есть возможность увидеть эти ссылки, а именно --show-reachable). Так как часто бывает очень трудно освободить такую память (вы должны сделать это во время финализации – finalization), и, как правило, это не стоит того, так как память все равно будет исправлена системой, вы, как правило, захотите игнорировать эти куски.

Аналогично, если у вас есть выделенный объект, который сам содержит обращения к динамически распределённой памяти, valgrind по умолчанию будет сообщать только о корневом объекте как утечке, так как другие куски доступны из первого. Когда вы исправляете утечку для первого, вполне вероятно, что вы одновременно устраните другие утечки.

Другими источниками утечек памяти, которые намного сложнее обнаружить, являются те, которые встречаются в графической части вашего приложения. В большинстве систем графическое рендеринг - это процесс клиент-сервер, а память выделяется на сервере, а не на клиенте. Поэтому инструменты, такие как valgrind или пакеты GNAT, не смогут сообщить об этом, идентифицировав в этом процессе утечку памяти (например, если вы выделили большую pixmap, но никогда не освобождали её). В Windows они называются «GID» и видны в диспетчере процессов, поэтому вы можете реально контролировать, появляются ли у вашего приложения такие утечки. Ещё раз обратим Ваше внимание, что это, однако не даёт никаких подробностей о происхождении утечки.

В Unix вы можете использовать приложение «xrestop», которое может указать вам количество окон, курсоров, графического контекста и т. д., которые вы выделили/захватили Вашей программой.

Существует множество таких инструментов, как коммерческих, так и бесплатных. Сообщите нам, если вы регулярно используете такие инструменты, поскольку они могут быть полезны другим читателям этого Gem.

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Сведений об авторе нет.

Last Updated: 10/13/2017
Posted on: 2/8/2010

Обсуждение…