**10.Реализация стека вызовов, реализация локальных данных нити.**

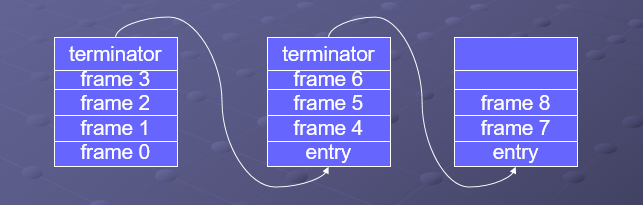
**Реализация стека вызовов нити**:

[**Про нити**](http://citforum.ru/operating_systems/sos/glava_14.shtml)

[**Стек вызовов(вики)**](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2%D0%BE%D0%B2)

Из презентации:

* Релокация стека
  + избыточное копирование памяти
  + требует отдельного стека для вызова через внешний интерфейс
* Спагетти стек



* сложность реализации
* требует отдельного стека для вызова через внешний интерфейс
* Выделение виртуальной памяти
* перерасход виртуальной памяти
* ограничение максимального размера стека

**Локальные данные нити (Thread Local Storage)**

[**Thread local storage**](https://en.wikipedia.org/wiki/Thread-local_storage)

[**ThreadLocal Class in Java**](http://netjs.blogspot.ru/2015/07/when-and-how-to-use-thread-local-class-in-java.html)**,** [**javadoc**](https://docs.oracle.com/javase/8/docs/api/java/lang/ThreadLocal.html)

**private** **static** **final** ThreadLocal<Integer> myThreadLocalInteger = **new** ThreadLocal<Integer>();

Различные нити работают параллельно над общими статическими данными, совершая таким образом некоторую совместную работу. При этом одна и та же подпрограмма может вызываться из разных нитей. В отличие от статических переменных, которые являются общими для всех нитей, для каждой нити выделяется свой отдельный стек. При использовании нитей очень важно, чтобы локальные переменные подпрограммы располагались в стеке. Иначе было бы невозможно параллельно вызывать одну и ту же подпрограмму из разных нитей: повторный вызов подпрограммы, уже работающей в рамках другой нити, разрушил бы статический набор локальных переменных этой подпрограммы. А при использовании стека наборы локальных данных одной и той же подпрограммы, вызываемой из разных нитей, различны, поскольку они располагаются в разных стеках. Таким образом, разные нити работают с разными наборами локальных переменных, не мешая друг другу.

*Системный вызов* — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

*Через регистр процессора:*

Во всех современных архитектурах точка возврата сохраняется в аппаратном стеке, что делает возможным рекурсию, а также параллельное выполнение нескольких легковесных процессов (нитей). Для вызова подпрограммы f служит команда call, которая осуществляет переход к подпрограмме f (т.е. присваивает регистру PC адрес f ) и одновременно помещает старое содержимое регистра PC в стек:

call f ~ push PC;

PC:= f;

В момент выполнения любой команды регистр PC содержит адрес следующей команды, т.е. фактически адрес возврата из подпрограммы f. Таким образом, команда call сохраняет в стеке точку возврата и осуществляет переход к подпрограмме f.

Для возврата из подпрограммы используется команда return. Она извлекает из стека адрес возврата и помещает его в регистр PC:

return ~ pop PC;

Из презентации:

* Через системный вызов
  + очень медленно
  + только для нитей OS
* Через регистр процессора
  + очень быстро
  + занимает регистр процессора
  + требует поддержки Исполнителя
* Часть стека нити
  + быстро
  + требует выделения стека блоками фиксированного размера с известным выравниванием
* Отображение блока стека
  + медленно
  + использование самосинхронизированной структуры данных