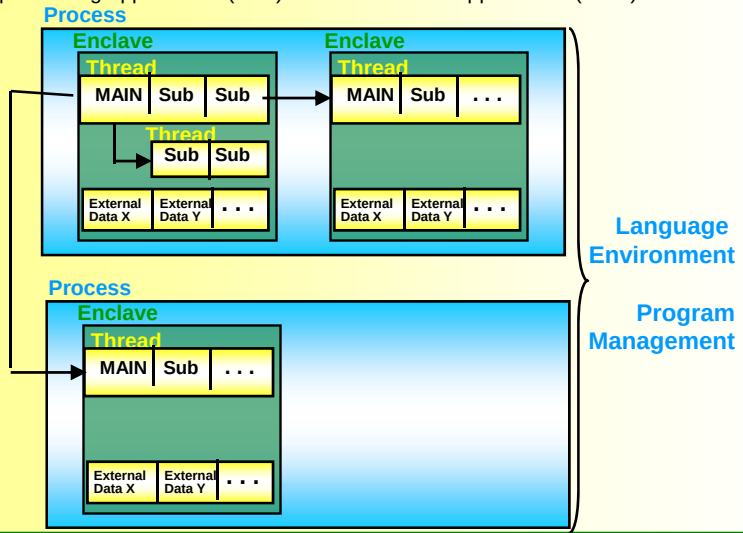
CheatSheet #41 zTidBits zOS' Processes, Enclaves & Threads

•**Program management** defines the program execution constructs of an application, and the semantics associated with the integration of various components management of such constructs.
•Three entities—process, enclave, and thread are core in the program management model.
• The highest level component of the program model is the **process**.
- A process consists of at least one enclave and is logically separate from other processes.
- Processes do not share storage and are independent of and equal to each other
> They are not hierarchically related.

• Processes can create new processes and communicate to each other by using a defined communication, for such things as indicating when a created process has been terminated.
• A key feature of the program management model is the **enclave**, a collection of the routines that make up an application.
• The enclave is the equivalent of any of the following:
- A *run unit*, in COBOL
- A *program*, consisting of a *main C function* and its *subfunctions* in C and C++
- A *main procedure* and all of its *subroutines* in PL/I
- A *program* and its *subroutines* in Fortran

• The enclave consists of one main routine and zero or more subroutines.
NOTE: A POSIX application might not have a main routine active at a given time.
The main routine is the first to execute in an enclave, all subsequent routines are named as subroutines.
• The enclave logically owns resources normally associated with the running of a program.
- As in z/OS virtual storage control information such as Task Control Blocks (TCBs) and Supervisor Request Blocks (SRBs) are used to monitor and account for pgm status.

• Some resources are owned directly, such as heap storage; some are owned indirectly, such as the run-time stack, which is owned by a **thread.**
• Each **enclave** consists of at least one thread, the basic instance of a particular routine.
• A thread is created during enclave initialization with its own run-time stack, which keeps track of the thread's execution, as well as a unique instruction counter, registers, and condition-handling mechanisms.
• Each thread is an independent instance of a routine running under an enclave's resources.
• Threads share all of the resources of an enclave.
• A thread can address all storage within an enclave.
• All threads are equal and independent of one another and are not related hierarchically.
Note: A thread can create a new enclave.
• Because threads operate with unique run-time stacks, they can run concurrently within an enclave and allocate and free their own storage.
- Because they may execute concurrently, threads can be used to implement parallel processing applications (DB2) and event-driven applications (WAS).

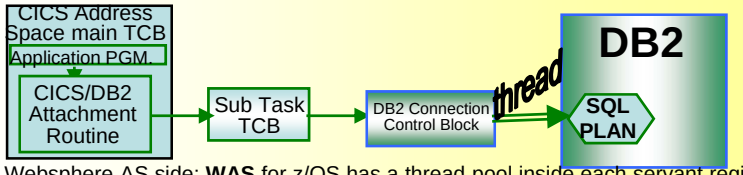
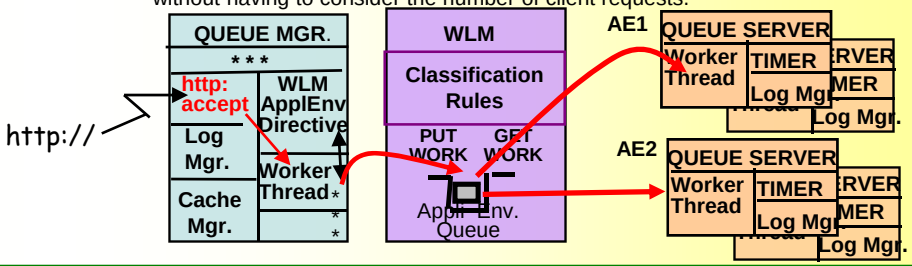


• Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the z operating system.
• To a software developer, the thread concept is a "procedure" that runs independently from its main program may best describe a thread.
NOTE: To go one step further, imagine a main program PGMA that contains a number of procedures. Then imagine all of these procedures being able to be scheduled to run simultaneously and/or independently by the operating system.
This would describe a "multi-threaded" program !

Storage and Threads In Language Environment, a run-time stack, or stack storage, is automatically created when a thread is created, and freed when the thread terminates.
• When a thread is created, Language Environment allocates an initial stack, which can have stack increments added to it as needed. Users can specify the sizes of the initial stack and additional stack increments; they can also tune the stack for performance.
NOTE: For ***AMODE 64*** support, users can specify a stack size above the bar, specifying the maximum stack size. A contiguous block of storage is allocated above the bar.
In ***AMODE 31***, each stack segment is allocated separately.
• Heap storage can be allocated and freed in no particular order.
NOTE: Stack storage, in contrast, is allocated when a routine is entered and freed when the routines ends.
• Heap storage is shared among all program units and all threads in an enclave.
- Allocated heap storage remains allocated until it is explicitly freed by a thread or until the enclave terminates.
• Heap storage is typically controlled by the programmer through Language Environment run-time options and callable services.

You may had heard the term heavy weight thread.
- A heavy weight thread has a one-to-one correspondence with an MVS task control block (TCB) in that the lifetime of the thread is the lifetime of the TCB.
NOTE:A TCB is a small structure in virtual memory containing the current process status used by the address space representing a problem-state program (user program).
It's a common form of managing the "bookkeeping" of units of work in real-time.
• In this case multi-threading and multi-tasking come together.

Who uses heavy weight threads- CICS / DB2 / WAS.
•Within the overall connection between **CICS** and **DB2**, each CICS transaction that accesses DB2 needs a thread, that is, an individual connection into DB2.
•Each thread runs under a thread task control block (thread TCB) that belongs to CICS.
•CICS and DB2 both have connection control blocks linked to the thread TCB.
- They use these connection control blocks to manage the thread into DB2, and to communicate information to each other about the thread.
- The DB2 connection control block controls the thread within DB2.



On the Websphere AS side: **WAS** for z/OS has a thread pool inside each servant region to serve the client requests and is managed differently from its Distributed counterpart.
• When a request comes into WAS for z/OS the control region receives the request and and creates an enclave to represent the transaction.
- Before the request is processed by the servant region, it's put inside a WLM work queue.
- Unlike distributed WAS, WAS for z/OS does not need to use threads to hold requests, instead it uses the WLM queue as the placeholder.
• For a distributed WAS handling relatively high stress workload, you might need a larger number of threads to hold the client requests since the system overhead will be higher because of intensive context switching.
- WAS for z/OS gives you much easier management for the thread number.
- You only need to set a proper number of worker threads for workload execution, without having to consider the number of client requests.