Structuring Of Systems Using Upcalls

- By David D. Clark

Presented by :-
Varsha Radhakrishnan
Overview

- Layered Approach
- Need for new layered approach
- Multitask Modules
- Advantages
- Disadvantages
- Swift system
Layered architecture

- Breaks up the operating system into different layers.
- Well-defined interfaces between the layers.
- Service invocation has a top down approach
- Each layer provides services to the layer above or client layer.
- Provides modularity and has implicit hierarchy of trust.
- Simplicity of construction and debugging
But the problems are...

- Each layer corresponds to a particular process (as in an event based model) with separate address space per layer. This means interlayer communication becomes expensive cross-domain communication (IPC, RPC, LRPC, URPC etc).

- The natural flow of control is not always downwards!! Source of an activity can be from a lower to a higher layer also. Such upcalls are usually implemented as asynchronous interprocess communications signal which is very inefficient.
Solution is ..

- A new layered approach in which procedure calls are used to communicate up or down between layers.

- Instead of mapping a layer to a process, a layer is organised as subroutines which live in a number of tasks, callable as appropriate from layer above or below.

- There would be a single shared memory, with access controlled by a monitor lock.
Multi-task modules
Multi-task modules

- A layer is organized as a group of collaborating subroutines from different tasks.
- Flow of control between layers is achieved through subroutine calls (synchronous), both up and down.
- Individual tasks move up and down between the layers according to the natural control flow.
- A single shared memory
- The term "task" is used in more or less the same way we have been using the term "thread". So, the multi-task modules are multi-threaded modules. So, this is a procedural model, i.e., multi-threaded programming in a shared address space.
Example: Three layer protocol package (Create and receive task)
Arming call

- It is a downcall telling lower layers that an action should be taken.

- Called "arming call," because it arms the lower layer for action.

- The arming downcall does no serious processing, and always returns immediately, never blocking.

- The resulting upcall is executed whenever the flow control would permit.
Advantages

❖ The use of subroutine calls increases the efficiency of the system.

❖ Procedure flow maps on to the natural flow of control in the program, whether that is up, down, or sideways.

❖ Lower layer can “ask advice” to the upper layer.

❖ “Piggybacking” occurs naturally- Method in which information from various layers are combined into a single outgoing packet as an efficiency enhancement.
Disadvantages

➢ Trusted layers call into untrusted layers.

With this upcalling approach the lower layer must trust the higher layers and the activities in higher layers must trust each other as well.

Propagation of failures:-
If the module implementing the upper layer fails, the lower layer would be left in an inconsistent state, which would destroy the lower layer and every other client layer of the lower layer.
Methods to overcome this:

- Shared data of clients must be consistent and unlocked before any upcall is made.

- We can destroy the task executing the failed upcall provided it is isolated from resources owned by other clients.

- Separate expendable tasks for upcalling each client.
Disadvantages (Cont..)

➢ Indirect recursive calls
   When control returns back from the upcall, lower layer may find that its state has changed.

   - Before making upcalls all variables of the lower layer must be in consistent state, and then re-evaluated on return.(clumsy).

   - Prohibit recursive downcalls

   - Downcall requests are put in work queue of the task holding the lock for later execution.(complexity of work queue).

   - Restrict the semantics of down call.(just set flags etc).

   - In special cases downcall call can be replaced as return arguments to upcalls or as an upcall to query
Disadvantages (Cont...)

- Difficult to find all pieces of a module if it is necessary to change global state of a module, for ex, to shut it down.

  - A global cleanup signal can be sent out to all relevant tasks, defining which module is shutting down.
Swift OS

Task scheduling

- Priority scheduling which is dynamic (measured in terms of deadline)
- Queue of tasks associated with monitor.
- ”Deadline” promotion - if a task with a short deadline encounters a monitor held by a task with longer deadline, it can temporarily change the deadline of the other task to a shorter value.

Address Space Management

- Single shared address space for efficient passing of data from one task to another (may cause corruption due to program bugs)
- Program system in high level language CLU which provides strong checking at compile and run time to ensure that the address space is not corrupted.
- Garbage collection used to prevent dangling pointer (pointers of destroyed objects).
Conclusion

➔ Structuring systems using upcall leads to efficient programming. Improves performance and code size by a factor of five to ten.

➔ It could help systems to have the natural flow of control more easily.

➔ Proved that mapping processes to layers can be a bad idea.

➔ Efficiency Vs Trust issue

➔ Overuse of upcalls is dangerous - we may not be able to separate two modules anymore.
Reference


- Images from slides of Peter Banda, Winter 2012
Thank you..