Mach's VM System

- Idea: use memory objects and message passing mechanisms to allow user programs to manage their memory
- A brief history of Mach:
  - Accent, Mach 2.5, Mach 3.0 and OSF/1
  - Early work: Accent
    - use memory-mapping to transfer large amount of data
    - a "single level store"

Message Passing in Mach

Use message-based IPC to support OS:

- secure communication (only authorized threads can send/receive messages)
- task_self, task_notify, thread_self, reply, exception ports; inherited registered ports
- support client-server model; allow kernel services be constructed as user level servers
- interface transparently extensible to a distributed environment

Mach IPC: Ports

- Port: mailbox with authorized senders only
  - anyone who has a send/recv right can send/recv
  - only one receiver at a time
  - task/thread has default ports; ports can be inherited
- Capability: send rights and receive rights
  - the user task uses a local name to refer to a port and its capabilities on that port
  - kernel maintains the correspondence

Mach IPC: Messages

- simple messages: uninterpreted, simply copied
- complex messages: interpreted by kernel
  - out-of-line memory
  - send/recv rights to port
- Interface:
  - header: (type, size, destination port, reply port, message_ID), followed by data components
  - msg_send/rcv/rpc (header, option, timeout)

Coupling between VM and IPC

Part 1: copy-on-write for large messages

- out-of-line memory: special type descriptor
- msg_copyin (called from msg_send)
  - modify sender task's mapping to be read-only and COW
  - create temporary holding map for the pages
- msg_copyout(called from msg_rcv)
  - allocate an address range in the receiving task
  - copy the entries from holding map; deallocate holding map
- kernel handles mappings when sender or receiver modifies a page

Mach's VM System

- a task's address space can contain variable number of memory regions
  - vm_allocate(task, address, size, anywhere)
  - vm_allocate_with_pager
  - vm_deallocate
  - vm_regions, vm_statistics
  - vm_protect
  - vm_inherit (NONE, SHARE, COPY)
  - vm_read, vm_write

Memory Objects

- Represented by a port
  - an abstract object representing a collection of data bytes on which several operations (e.g. read and write) are defined
  - Initial values and permanent storage are provided by the ***data manager (or pager)***
  - kernel acts as a cache manager for mem objects
- Interface
  - vm_allocate_with_pager(task, address, size, anywhere, memory_object, offset)

Memory Object Manager

- Kernel to data manager interface
  - pager_init(mem_obj, pager_req_port, pager_name)
  - pager_data_request(obj, req_port, offset, len, access)
  - pager_data_write(obj, offset, data, count)
  - pager_data_unlock(obj, req_port, offset, len, access)
  - pager_create(old_obj, new_obj, new_req_port, new_name)

Data Manager

- Data manager to kernel interface
  - pager_data_provided(req_port, offset, data, data_count, lock_value)
  - pager_flush_request(req_port, offset, len)
  - pager_clean_request(req_port, offset, len)
  - pager_data_lock(req_port, offset, len, lock)
  - pager_cache(req_port, may_cache_object)
  - pager_data_unavailable(req_port, offset, size)

Example Using External Pager

- Consistent network shared memory

Implementation Details

- Address maps
- Internal memory object structures
- Resident memory Structures
  - virtual to physical hash table
- Page Replacement Queues
- Fault Handling
  - validity and protection, page lookup, copy-on-write, hardware validation

Memory Failures

- Types of memory failures
  - fails to return data; fails to free flushed data; floods the cache; changes data; blocks on its own data
- Handling memory failures
  - timeout, notification, wait; abort; use default pager

Applications

- Emulating UNIX environments
- Copy-on-reference task migration
- Database Management: Camelot
  - kernel is the cache manager
  - data manager guarantees the write-ahead-log protocal
- Speech research: Agora
  - use both shared memory and message passing