Goals

- New user interface
- Ability to support large, sparse address space
- Unify memory caches
- Efficient memory sharing mechanism
- Better separation of machine dependent/independent layers
- Rethink backing store policies

Why MACH VM?

- Copy-on-write (COW) is fundamental to the design
- Clean separation of machine dependent/independent layers
- Modular design
- Uses ‘‘lazy allocation’’ of resources
- MACH still looks like UNIX
- Has multi-processor support
- Works and the price is right

Concerns about MACH VM

- Contain feature that aren’t needed
- COW may not be so wonderful on all machines
- Different implementation of existing features
- It is inherently page-based

MACH VM with the old user interface

- Throw out /sys/vm_* files
- Replace with MACH /sys/vm directory
- Add some ‘‘glue’’ files to tie together
- Remove external pager support
- *fork*() changed to do vm_map_fork()
- *exec*() maps object back by appropriate vnode (vm_allocate_with_pager)
- *sbrk*() allocates more space in process address map
- User stack pre-allocated at the maximum size
- Implemented a simple pager interface and simple pagers

The Pager Interface

```
struct      pager_struct {
        queue_head_t        pg_list;        /* list management */
        caddr_t             pg_handle;      /* external handle */
        int                 pg_type;        /* type of pager */
        struct pagerops     *pg_ops;        /* pager operations */
        caddr_t             pg_data;        /* private pager data */
};
typedef struct pager_struct *vm_pager_t;

struct      pagerops {
        void            (*pgo_init)(        /* */ );
        vm_pager_t      (*pgo_alloc)(       /* handle, size, prot */ );
        void            (*pgo_dealloc)(     /* pager */ );
        int             (*pgo_getpage)(     /* pager, page, sync */ );
        int             (*pgo_putpage)(     /* pager, page, sync */ );
        boolean_t       (*pgo_haspage)(     /* pager, offset */ );
};

caddr_t         handle;
vm_size_t       size;
vm_prot_t       prot;
vm_pager_t      pager;
vm_page_t       page;
boolean_t       sync;
vm_offset_t     offset;
```

The new user interface

- *mmap*
  check arguments
  lookup (allocate) pager/object of appropriate type
  allocate space in address map
  map in directly (MAP_SHARED) or virtual copy in (MAP_PRIVATE)
- *munmap*
  check arguments
  lookup address map entry or entries
  decrement reference count on object (may destroy)
- *msync*
  check arguments
  lookup address map entry or entries
  synchronously clean dirty pages in affected object range
  flush all pages in affected object range
- *mprotect*
  check arguments
  use existing vm_map_protect primitive to change protection

Problems encountered so far

- Page and buffer cache synchronization
- Copy-on-write semantics are not quite right for MAP_PRIVATE
- MACH assumes sharing will involve a sharing map

What has yet to be done

- Better control over physical memory
- Implementation of remainder of user interface
- Code cleanup
- Conversion of applications
- More kernel use of copy-on-write
- Performance improvements

Unresolved issues

- How to integrate page and buffer caches
- Naming anonymous memory
- Is the pager interface needed?
- New object file format?
- Backing store policies
- Compatibility with existing implementations