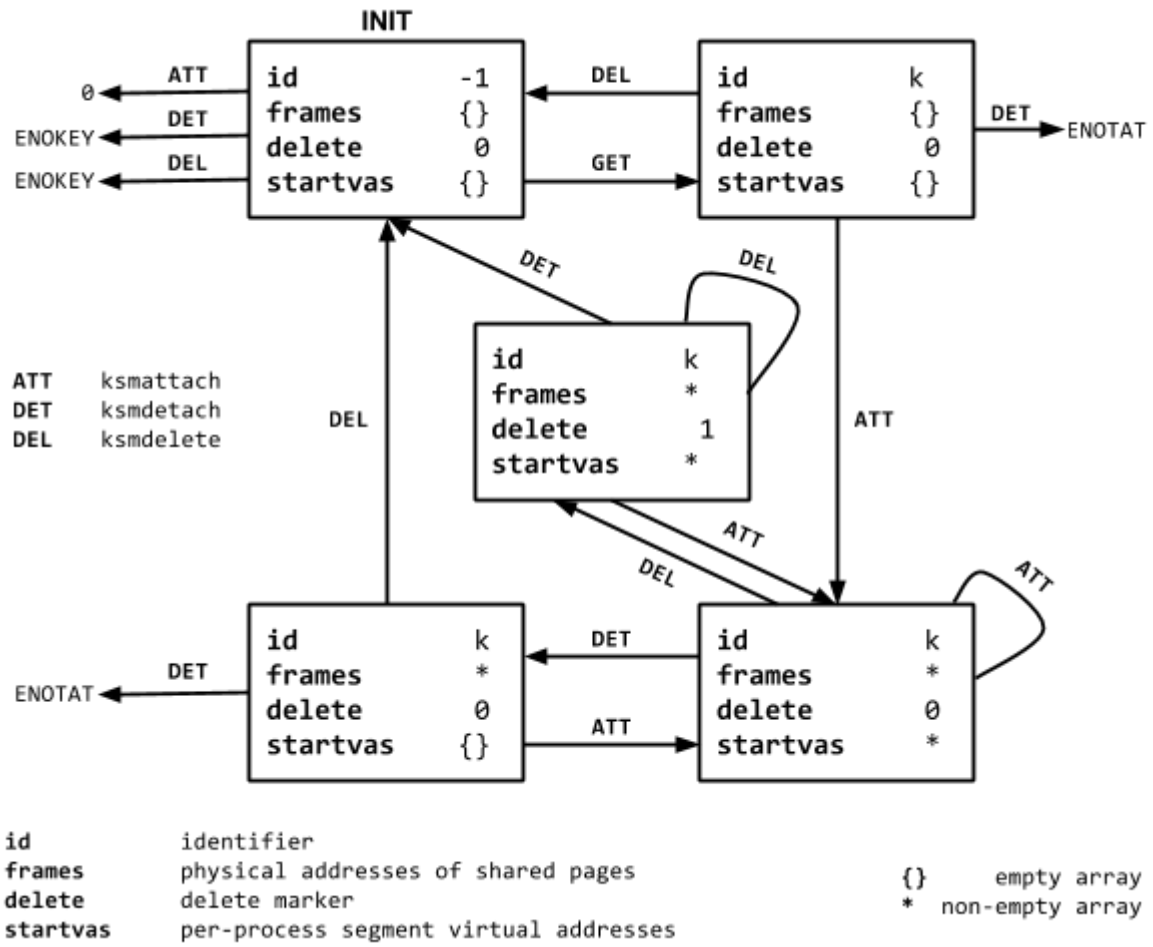
CS240 Assignment 3

Fall 2013, Prof. Hany Ramadan

Emaad Ahmed Manzoor (129110) November 21, 2013

Design. The following state diagram summarizes all the transitions and reachable global (`id`, `frames`, `delete`) and per-process (`startvas`) states of a single shared memory segment.



Limits. The following parameters are assumed to be fixed at compile time.

`MAXKSMIDS`	The maximum number of shared memory segments in the system.
`MAXKSMSZ`	The maximum size of a shared memory segment.
`MINKSMSZ`	The minimum size of a shared memory segment.

A larger `MAXKSMIDS` results in a larger kernel. `MAXKSMSZ` and `MINKSMSZ` are only a matter of policy and do not affect the kernel size when the kernel is compiled with the `LARGESEGS` mode (discussed in the additional features section). When the `LARGESEGS` mode is disabled, a larger `MAXKSMSZ` increases the kernel size drastically, since each `struct ksminfo_t` that consumed just 40 bytes in `LARGESEGS` mode will now consume the order of `MAXKSMSZ` * 4 bytes.

Files Modified or Created.

`Makefile`	Entries for new source files, `CFLAGS` for the two shared memory modes.
`defs.h syscall.c syscall.h sysproc.c user.h usys.S`	Stub code and function prototypes for the system calls.
`kalloc.c`	Implementation of the `pgused` system call.
`main.c ksm.c ksm.h ksmtest.c mmu.h param.h`	Shared memory implementation, parameters, page table entry bit and user test program .
`proc.c proc.h exec.c vm.c`	Modified `fork`, `exit` and `exec` behaviour.

Interface. The details below also address many design questions posed in the assignment handout.

`ksmget`

Returns:

`handle`	On success.
`ENOIDS`	If there are no free handles.
`EINVAL`	If the requested size is outside the `[MINKSMSZ,MAXKSMSZ]` bounds.
`EEXISTS`	If the `MUSTNOTEXIST` flag is set and the handle for the identifier exists.
`ENOTEXIST`	If the `MUSTEXIST` flag is set and the handle for the identifier does not exist.

`ksmget` only reserves a handle for the given identifier and records the expected size rounded up to the nearest page. Free handles are found by a linear search in a kernel array starting from the last returned handle.

`ksmget` accepts two flags as bits in the flags argument:

`MUSTEXISTS`	Causing failure if a handle does not exist for the given identifier.
`MUSTNOTEXIST`	Causing failure if a handle exists for the given identifier.

If both bits are set, the return value can be used to determine if a handle exists for an identifier without modifying any shared memory state.

`ksmattach, ksmattach_proc`

Returns:

The start virtual address of the shared memory segment in the process, if successful.

`0`, if:

- The handle is invalid.
- The handle has no associated identifier.
- The there is not enough virtual address space to map into the process.
- There is not enough physical memory to allocate the segment.

ksmattach calls ksmattach_proc with the currently active struct proc. The separation is so that ksmattach_proc can be called from proc.c to attach shared memory to a forked process starting at a specific virtual address.

ksmattach accepts a flag indicating a read-only attachment if set. This attachment maps pages into the process' page table with PTE_W cleared on the page table entries. All the shared memory pages are mapped in with a newly introduced PTE_S bit (defined in mmu.h) set in the page table entry. This is used to track and prevent freeing shared memory pages on process exit.

If no page frames have already been allocated for the given handle, the allocation is carried out and the frame addresses are recorded. If page frames have already been allocated, they are simply attached to the given process' page table. Multiple attachments to a segment just return the start virtual address of the segment in that process without making any changes.

Shared memory attachments in the process address space grow from KERNBASE downwards, until proc->sz. Free slots in the virtual address space are found using first-fit. Holes are ignored, so the virtual address space may become fragmented.

ksmdetach, ksmdetach_proc

Returns:

0	If successful.
EINVAL	If the handle is outside the [0,MAXKSMSZ] bounds.
ENOKEY	If there is no identifier that has been associated with this handle with ksmget.
ENOTAT	If the segment is not currently attached to the calling process.

ksmdetach calls ksmdetach_proc with the currently active struct proc. The separation is so that ksmattach_proc can be called from proc.c when exec or kill is called on a process. ksmdetach removes the shared memory mapping of the given handle from the process' page table. If marked_for_delete has been set on a segment, it also calls ksmdelete on that segment.

ksmdelete

Returns:

0	If successful.
EINVAL	If the handle is outside the [0,MAXKSMSZ] bounds.
ENOKEY	If there is no identifier that has been associated with this handle with ksmget.

If the given segment has processes attached to it, ksmdelete sets marked_for_delete on that segment and returns. If it has no processes attached, its physical pages are freed and reserved handle is released (set to -1).

Segments marked for deletion may be deleted once the last attached process detaches from it, or may be unmarked if a process attaches to it. Segments not marked for deletion are not deleted when the last process detaches; i.e. they are persistent by default. Attaching to a segment marked for deletion succeeds and clears the marked_for_delete flag.

`ksminfo`
Returns: Nothing.

`ksminfo` just copies the required information for the handle to the user-provided address of a `struct ksminfo_t`. If the handle provided is `0`, only the global shared memory information is copied.

Process Behaviour. Descriptions of shared memory states when a process `fork`s, `exit`s or `exec`s.

fork. When a process forks, it calls `ksmattach_proc` to attach its shared memory segments to its child. It also calls `copyksmperms` to copy the read-only permissions of shared memory segments from the parent to the child.

exit, exec. When a process exits or execs, it calls `ksmdetach_proc` on itself. While `deallocuvm` frees a process' page table, it checks for the `PTE_S` bit in every page table entry and avoids freeing the mapped physical pages for these page table entries.

sbrk. A record of the lowest attached shared memory segment address is maintained in each process as `p->ksmstart`. `sbrk` checks this when growing the process and fails if the new process size will overlap shared memory. On adding a new segment, `ksmstart` can be updated with a single comparison and assignment. Only on deleting the lowest attached segment will a linear scan of all the attached segment addresses be required to update the lowest segment address.

Additional Features.

Two shared memory modes. By default, the kernel is compiled in the `LARGESEGS` mode. In this mode, the physical addresses of the shared memory segments are tracked dynamically with a linked list. Each node is page-sized, and stores `1023` segment frame addresses and one pointer to the next node. Since memory is allocated at the page granularity, this wastes space if the segments are not several pages large.

Without `LARGESEGS`, physical page addresses of a shared memory segment are tracked in a static array in the kernel. This consumes a fixed `4 * MAXKSMSZ` bytes per `struct ksminfo_t`. When segment sizes are small, this is efficient.

Dynamic segment tracking can be disabled conditionally by recompiling the kernel after uncommenting **line 79** in the `Makefile`.

State modifications are isolated to few functions. For the two classes of shared memory state (per-process and global), only four functions in `shm.c` (`ksmget`, `ksmattach_proc`, `ksmdetach_proc` and `ksmdelete`) cause state transitions. They perform heavy parameter validation and can be easily debugged. `proc.c` is restricted to initializing the per-process state and delegates all state updates to these four functions.