OS MP2 Implementation

B08902055 李英華

April 19, 2021

1 How To Manage VMA

- Each process has its *vma_list*, which is an array of VMA of size $nVMA$(=16).
- The struct contains:
  - *vm_valid*
    Indicate whether the VMA is used.
  - vm_start, vm_end
    Address range.
  - *vm_page_prot*
    Access permission.
  - vm_flags
    Flags of mapping.
  - vm_file, vm_fd
    Mapping files and fd.
- Each process also has the pointer *curr_va* pointing to the current max virtual memory address which could be assigned to the next VMA.

2 mmap

1. Implementation
   Find the unused region in the process address space so that the file could be mapped to. The corresponding data (address, length, flags, etc.) retrieved from trapframe is then be stored in the unused VMA.
2. Assumption
   - *length* will be the multiple of $PGSIZE$.
   - *addr* is assumed to be 0, so the *start* and *end* address will be assigned as page-aligned.

3 mmap with Lazy Allocation

1. Implementation
   Modify the *trap.c* in response to a page fault. First check whether the faulting address is within the range of any VMA possessed by process. Then map a new physical page at the faulting address.

2. Assumption

   - No specific assumption in this section.

4 munmap

1. Implementation
   Find the VMA within the range and unmap corresponding pages. If all pages are removed, reference count of the corresponding struct file will be decreased. If the mapping flag is MAP_SHARED, write the modified page back to the file if any.

2. Assumption

   - *length* and *addr* will be the multiple of $PGSIZE$ and *length* will not cross different VMA. That is, if address matches the range of one VMA, we don't need to check other VMAs.
   - Assume unmap only at start toward middle or at middle toward end or maybe unmap the whole VMA.
   - Assume the memory is unlimited. That is, we don't need to consider the previous unmapped area and could simply keep moving the *curr_va* pointer downward for the next VMA.
     But, if we find that none of the VMAs is invalid, which is, all the VMA have been unmapped, the pointer *curr_va* will be reset to the original start point, VMA_strt.

5 Reclaim mmap-ed Files

1. Implementation
   Find the invalid VMA and check whether it has physical address. If true, *uvmunmap*, and write back the modified pages to the file if the flag is MAP_SHARED. $fileclose$ after write back to file.

2. Assumption

   - Inherit the previous assumption, no new assumption needed in this section.

6 Shared Virtual Memory

1. Implementation
   Allocate new physical pages for the child if the pages in parent have been allocated physically. If not, simply copy the virtual memory (VMAs).

2. Assumption

   - Assume that after fork, each time parent or child modify the shared files, they will write back to file right after, and they won't write to the shared file simultaneously. This way, file change is shared between parent and child.