Embedded Operating Systems

Paging

mmap() Syscall

- Allocate a segment in virtual address space of the process.
  - Create a new VAD to keep information of segment (start, end, prot, flags, ...) and add it into process's VAD list (i.e. memory map of the process -> mm_struct -> mmap).
  - Create new page table entries for this segment.
- ptr = mmap(addr, length, prot, flags, file_descriptor, file_offset)
  - arg1: Start virtual addr of process's VAS to which new segment is to be mapped. NULL means any available address.
  - arg2: Length of virtual address segment
  - arg3: Segment protection flags: PROT_READ, PROT_WRITE, PROT_EXEC, or PROT_NONE.
  - arg4: Segment flags: MAP_PRIVATE, MAP_SHARED, MAP_ANONYMOUS, or ...
  - arg5: File descriptor of file to be mapped.
  - arg6: File offset of the file to be mapped.
  - returns: virtual base address of the process at which segment is allocated.
- ptr = mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, fd, 0);
  - When read operation is performed on the segment, page fault will occur.
  - It will allocate an empty frame and load data from the file in it.
  - Now we can access data from the file.
- ptr = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
  - Creates anonymous segment -- not backed by any file on disk.
  - e.g. heap segment, stack segment, ...
- munmap(ptr, size)
  - remove segment from process's VAS.
    - delete its VAD from VAD list.
    - delete (or mark invalid) its page table entries.

brk() and sbrk() Syscall

- brk() change program break (in task_struct) to the given virtual address and the end virtual address will be updated in VAD (of heap).
- If new program break is higher than the current, new page table entries will be created. If new program break is lower than the current, corresponding page table entries will be invalidated.
- ret = brk(addr);
  - arg1: New virtual address of program break.
  - returns: 0 on success.
- sbrk() increment/decrement program break by given value/delta. It change program break (in task_struct) and the end virtual address will be updated in VAD.
- If given value is +ve, program break is incremented. The new page table entries will be created. If given value is -ve, program break is decremented. Corresponding page table entries will be invalidated.
- ptr = sbrk(delta);
  - arg1: int by which program break is to be incremented (+ve) or decremented (-ve).
  - returns: new program break.

malloc()

- C library function that dynamically allocate contiguous memory in the heap section of the process.
- The malloc() internally calls brk()/sbrk() or mmap() syscall to allocate contiguous address range into virtual address space of the process.
- When malloc() allocating small segments, it will return base address from existing heap address range. If exceeds beyond heap address range, then it internally calls brk() syscall to increase the heap address range.
- When malloc() allocating larger segments, it will invoke mmap() syscall to allocate a new segment into process's address space.
- Note that brk()/mmap() are allocating contiguous virtual addresses for the process.
- When virtual addresses are allocated to the process their page table entries are created. All these PTEs are invalid (yet to allocate).
- The physical memory is allocated pagewise into the page fault handler, when the memory is used (read/write).

Overcommit memory

- When memory is allocated using malloc(), mmap(), or brk(), only virtual memory/addresses are allocated in calling process's address space.
- The amount virtual memory that can be allocated in the system depends on two factors.
  - Process's address space size
  - Overcommit settings

- For 32-bit process (x86_32), max address space size is 4 GB. So maximum virtual addresses that can be given to process are 4 GB (including addresses of text, data, stack, heap, ... sections).
- Linux memory manager settings can be viewed and modified via procfs.
- /proc/sys/vm/overcommit_ratio
    - Default value is 50 i.e. 50%.
- /proc/sys/vm/overcommit_memory -- Overcommit mode
    - Default value is 0.
    - mode 0: heuristic overcommit
        - In mode 0, calls of mmap(2) with MAP_NORESERVE are not checked, and the default check is very weak, leading to the risk of getting a process "OOM-killed".
    - mode 1: always overcommit, never check
        - In mode 1, the kernel pretends there is always enough memory, until memory actually runs out.
    - mode 2: always check, never overcommit
        - In mode, the total virtual address space that can be allocated (CommitLimit in /proc/meminfo) is calculated as

```
CommitLimit = (total_RAM - total_huge_TLB) * overcommit_ratio / 100 + total_swap
```

Paging optimization

- If page fault handling involves disk IO, it is referred as major fault; otherwise it is minor fault.
- To improve paging performance major page faults (disk io) should be minimized.
- terminal> ps -e -o pid,vsz,rsz,maj_flt,min_flt,cmd

Dirty Bit

- Each entry in page table has a dirty bit.
- When page is swapped in, dirty bit is set to 0.
- When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area/disk.
- When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.

- If bit=1 (page is modified), page is physically overwritten on the swap area.

vfork()

- fork() create child process by duplicating calling process.
    - Allocates separate memory space for the child process.
    - Create a new thread to execute the child process.
- If exec() is called, it release allocated segments and reallocate as per need of new program to be loaded.
- To avoid this, BSD UNIX developed vfork() system call to create new process. This should be used only while calling exec() in child.
- vfork() creates child process virtually. It doesn't duplicate parent process; rather suspends execution of parent and continue execution of child process until exec() or exit() is called (under parent thread of execution and in parent's address space itself).
- When exec() is called, then actual memory is allocated for the child process and new thread of execution is created for the child process. Hereafter child process executes independent of the parent process.
- terminal> man 2 vfork --> Historical description

Copy On Write

- Modern fork() syscall creates a logical copy of the calling process.
- Initially, child and parent both processes share the same pages in the memory.
- When one of the process try to modify contents of a page, the page is copied first; so that parent and child both will have separate physical copies of that page. This will avoid modification of a process by another process.
- This concept is known as "copy on write".
- The primary advantage of this mechanism is to speed up process creation (fork()).

Virtual memory

- Virtual memory is the memory that can be given to a process. Typically it includes RAM size (except kernel space) + Swap area.
- https://www.youtube.com/watch?v=qcBIvnQt0Bw&list=PLiwt1iVUib9s2Uo5BeYmwkDFUh70fJPxX

Assignments

1. Implement fast file copy program (assume max file size = 1 GB).

- step 1: open src file in rdonly mode.
- step 2: get size of src file (fstat() syscall)
- step 3: map src file contents to memory using mmap()
- step 4: create dest file in rdwr mode.
- step 5: make size of dest file, same as size of src file using ftruncate()
- step 6: map dest file contents to memory using mmap() -- MAP_SHARED.
- step 7: copy src file to dest file using memcpy()
- step 8: close src and dest files.