UNIX System Programming
Lecture 15: Shared Memory

- Outline
  - Shared Memory via mmap()
  - POSIX Shared Memory
- Reference
  - man Pages: mmap, munmap, msync, shm_open, ftruncate, shm_unlink

Lecture 18: Shared Memory

- Shared memory facilities allow processes to access the same area of memory. It is the fastest way to xfer data between processes.
- The shared memory routines map the physical address of the shared segment into the _______ address space of each process.
- When using shared memory you need to ensure synchronized access to the segment.

Lecture 18: Shared Memory via mmap()

- **mmap()** is a POSIX routine that has been used traditionally to share memory between processes. It is preferred over SYSV shared memory in modern operating systems.
- **mmap()** can be used to map a file, device or _______ object into memory. It can also be used by a single process for fast file I/O (via memory access instead of read/write).

Lecture 18: Shared Memory via mmap()

- **mmap()** is a function that maps a file or device into memory. It is used to share memory between processes.
- **flags** is formed by ORing several values. Common values are MAP_SHARED or MAP_PRIVATE and MAP_ANONYMOUS. If MAP_ANONYMOUS, fd should be -1.
- **fd** is a file descriptor for a file or object.
- **offset** is the beginning byte from the file to use in the mapping. Offset must be a multiple of the _______ size as returned by sysconf(_SC_PAGE_SIZE).

Lecture 18: Shared Memory via mmap()

- **mmap()** supports two types of mappings: ___________ and anonymous (flags must contain MAP_ANONYMOUS).
- A file-backed mapping maps a file (or a portion of a file) into memory. The file must first be opened and the file descriptor passed to mmap(). Other (related) processes that inherit the file descriptor can use mmap() to map to the the same memory segment.
Lecture 18: Shared Memory
Shared Memory via mmap()

- Recall that file descriptors are inherited across an _____(), so related processes can use mmap() on an inherited file descriptor to attach to the same memory segment.
- Anon. mappings are not file-backed. Anon. mappings are preserved across a fork(), but not across an exec() call. (Forked processes do not need to call mmap() to get access to the memory segment).

Lecture 18: Shared Memory
Shared Memory via mmap()

- mmap() segments can also be _________ or private (flags has MAP_SHARED or MAP_PRIVATE).
- Updates to shared segments are visible to other processes and written back to the file (for file mappings).
- Updates to private mappings are not visible to other processes and are not written back to the file.

Lecture 18: Shared Memory
Shared Memory via mmap()

- There are four comb. of file-backed/anonymous and shared/private to consider.
- 1) Anonymous and private mappings are similar to ________(). The memory is not shared. It is initialized to 0 (unlike malloc).
- 2) File-backed and private allow (related) processes to get an initial copy of a file. Updates to the memory segment do not affect other processes or the file.

Lecture 18: Shared Memory
Shared Memory via mmap()

- 3) _________ and shared mappings can be shared between related (forked) processes. The segment is initialized to zero.
- 4) File-backed and shared maps allow related (exec’ed) processes to share updates to a memory. Updates are written to the file.
- Neither file-backed or anonymous mappings can be shared by unrelated processes.

Lecture 18: Shared Memory
Shared Memory via mmap()

- File-backed and shared mapping is useful by a single process for memory-mapped file I/O.
- The file may not be as up-to-date as the memory segment. _______() can be used to force updates to the file.
- Segments should be unmapped using munmap(). Segments are automatically unmapped when the process exits.

Lecture 18: Shared Memory
POSIX Shared Memory

- POSIX defines an API for creating memory objects that can be shared by __________ processes. The objects must still be mapped into process memory space by mmap().
- Objects are created with shm_open():
  int shm_open(const char *name,
               int oflag, mode_t mode);
- name is the name of the object. It should have a leading /.
Lecture 18: Shared Memory
POSIX Shared Memory

* **shm_open()** returns a file descriptor.
* When a POSIX shared memory object is first created (**oflag** contains **O_CREAT**) the file descriptor returned from **shm_open()** should be passed to **ftruncate()** to set the ______ of the memory segment.
* The file descriptor is then passed to **mmap()** to map the object into the processes memory space.

Lecture 18: Shared Memory
POSIX Shared Memory

* **close()** can be used to close the file descriptor. This can be done after calling **mmap** without affecting the mapping.
* A memory object can be deleted using **shm_unlink()**. If it is not deleted it will persist until the next system ______.
* They can also be deleted from the command line using “rm”. They have corresponding file names in the **/dev/shm** directory.

Lecture 18: Shared Memory
Shared Memory

* When using shared memory segments you need to take precautions to __________ access to the segment.
* The example programs demonstrate using a shared variable for exclusive access when only two processes are sharing a segment.
* Other methods (semaphores) are more efficient and must be used when more than two processes are involved.

Lecture 18: Shared Memory
In Class Exercise

* Modify the example programs to use a **pair** of POSIX (named) semaphores for two-way synchronization.
* We need named semaphores here because the processes are independent.
* You can eliminate the **sleep()** calls after adding the semaphores.