System calls for managing POSIX shared memory.

- Step 1: create a shared memory segment. Only one process needs to do create the shared memory segment, but every process that wishes to access it must open it.

  ```
  #include <sys/mman.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  int shm_open (const char * name, int oflag, int mode);
  ```

  Returns a handle (called a file descriptor) to the shared memory segment on success and -1 on error. Must link the program with the real-time library (`-lrt`) when compiling.

  name the name of the shared memory segment. It must be unique to the system, and must start with a '/' character.

  oflag the open flags. Bit-wise or (the '|' operator) one or more of the following: O_RDWR (read/write), O_CREAT (create segment if it doesn't exist), O_TRUNC (delete segment if it does exist). See the man page for shm_open[1] for available flags.

  mode The access permissions for the segment. Use pre-defined constant S_IRWXU (give process User Read, Write and eXecute permission to the segment). See stat.h man page for a full list of available flags.

- Step 2: Set the size of a shared memory segment. Only one process needs to do this, and it is usually done by the process that created the segment.

  ```
  #include <unistd.h>
  #include <sys/types.h>
  int ftruncate (int fd, unsigned int length);
  ```

  Returns 0 on success and -1 on error.

  fd The file descriptor returned by shm_open().

  length The desired size of the segment in bytes.

[1] Type "man shm_open" at the command line prompt.

- Step 3: Map a shared memory segment into a process's address space. Every process that wishes to use the shared memory segment must do this.

  ```
  #include <sys/mman.h>
  void *mmap (void *addr, unsigned int length, int protection,
  int flags, int fd, unsigned int offset);
  ```

  Returns address of memory segment in calling process's address space, or (void *) -1 on error.

  addr The address in the process's address space to use for the mapping. Use NULL to let kernel assign address (recommended).

  length The size of the shared segment. Use the same size as you used when you created the segment or weird things will happen.

  protection What the process is allowed to do to the memory. Bit-wise or of PROT_EXEC, PROT_READ, and PROT_WRITE. Use the same permissions you specified when you created the segment.

  flags Use MAP_SHARED.

  fd The file descriptor of the shared memory to map. Use the value returned by shm_open().

  offset The distance from the beginning of the shared memory segment to map. Use 0.

- Step 4: Unmap a previously mapped shared memory segment. Every process that has `mmap()`ed the shared memory segment needs to do this.

  ```
  #include <sys/mman.h>
  int munmap (int fd, unsigned int length);
  ```

  Returns 0 on success, or -1 on error.

  fd The file descriptor of the shared memory segment.

  fd The length of the shared memory segment. Use the value you specified when it was mapped with `mmap()`.

- Step 5: Remove reference to a shared memory segment and, if this is the last reference, delete it. Every process that has `shm_open()`ed the segment needs to do this.

```
#include <sys/mman.h>
int shm_unlink (int fd);
```

Link with -lrt. Returns 0 on success and -1 on error.

fd The file descriptor of a shared memory segment returned by `shm_open()`.

Types and library calls for declaring and using POSIX semaphores. Link with `-lpthread`. You must `#include <semaphore.h>` to use semaphores. `Semaphore.h` defines the semaphore type `sem_t`.

- `int sem_init (sem_t *sem, int pshared, unsigned int value);`

  Initialize a semaphore. Returns 0 on success, -1 on error.

  sem A pointer to a semaphore.

  pshared Is this semaphore to be shared by different threads (0) or by different processes (1).

  value The initial value of the semaphore.

- `int sem_post (sem_t *sem);`

  Atomically increment the semaphore. This is the `signal()` or `P()` operation. Returns 0 on success, -1 on error.

  sem A pointer to a previously initialized semaphore.

- `int sem_wait (sem_t *sem);`

  Atomically wait for a semaphore to have a positive value and then decrement it. This is the `wait()` or `V()` operation. Returns 0 on success, -1 on error.

  sem A pointer to a previously initialized semaphore.