System calls:

int sharemem(char* va)
Creates a new shared memory region of size 4096 bytes (if it doesn't already exist), and maps it virtual address va. va must be page-aligned. Returns 0 on success, and -1 on failure.

API's:

int sharemem_va_init(char * va):
Must be called before read_share and write_share API's.
Maps the given virtual address (**va**) to physical page of shared memory. Makes a system call to do this. Returns -1 on failure, and 0 on success.

char* read_share(uint size):
Reads **size** number of bytes from the shared memory, and returns a char pointer to a copy of this data. If adequate data hasn't been written, then it waits until some other process writes the adequate amount of data. Returns 0 on failure. Does NOT deadlock with write_share.

Expects that sharemem_va_init(char *) has been called already.

int write_share(char *s, uint size)
Reads **size** number of bytes from **s,** and writes them to the shared memory, waits for other process to read, if the shared memory doesn't have adequate amount of space yet. Returns -1 on failure, and 0 on success.
Does NOT deadlock with read_share.

Expects that sharemem_va_init(char *) has been called already.

Test Case1:

```
#include "param.h"
#include "types.h"
#include "stat.h"

#include "fcntl.h"
#include "syscall.h"
#include "traps.h"
#include "memlayout.h"
#include "sharemem.c"
```

```
char *echoargv[] = { "echo", "ALL", "TESTS", "PASSED", 0 };
int stdout = 1;

void
mem(void)
{
  char *va = (char*) 0x00100000;

  if( sharemem_va_init(va) < 0){
     goto failed;
  }

  char* s = malloc(4096);
  for(int i =0;i<3000; i++){
     *(s + i) = '1';

  }

  if(write_share(s, 3000) == -1){
     goto failed;
  }


  int child_pid = -1;
  if((child_pid = fork() )==0){
     char *va2 = (char*) 0x00200000;
     //printf(1," succeded on fork\n");


     if( sharemem_va_init(va2) < 0){
        goto failed;
     }
     //printf(1," succeded on share_mem in child\n");


     if((s = read_share(3000)) == 0)
        goto failed;
     //char s3 = '1';
     for(int i =0;i<3000; i++){
        if(*s != '1'){
           //printf(1, "i is %d value read %d  should be %d \n", i, (int)*s, (int)s3);
           goto failed;
        }
        s++;
     }
     printf(1," succeded on read_share in child\n");


     char* s1 = malloc(4096);
```

```
    for(int i = 0; i < 3000; i++){
       *(s1 + i) = '2';

    }
    if(write_share(s1, 3000) ==  -1){
       goto failed;
    }
    printf(1," succeded on write_share in child\n");

    exit();
  }
  else{
    printf(1,"child pid is %d\n",child_pid );
    char* s2;



    wait();
    printf(1, "child has exitted\n");
    if((s2 = read_share(1500)) == 0){
       goto failed;
    }
    for(int i = 0; i < 1500; i++){
       if(*s2 != '2'){
          printf(1, "first read at i %d read %d\n",i,(int) *s2);
          goto failed;
       }
       s2+= 1;
    }

    if((s2 = read_share(1500)) == 0){
       goto failed;
    }

    for(int i = 0; i < 1500; i++){
       if(*s2 != '2'){
          goto failed;
       }
       s2+= 1;
    }


  }

  printf(1, "share_mem ok ");
  exit();
  failed:
    printf(1, "test failed!\n");
    exit();
}
```

```
int
main(int argc, char *argv[])
{
  printf(1, "share_memmemtest starting\n");
  mem();
  return 0;
}
```

Test Case2:

```
#include "param.h"
#include "types.h"
#include "stat.h"
#include "fcntl.h"
#include "syscall.h"
#include "traps.h"
#include "memlayout.h"
#include "sharemem.c"

char *echoargv[] = { "echo", "ALL", "TESTS", "PASSED", 0 };
int stdout = 1;

void
mem(void)
{
  char *va = (char*) 0x00100000;

  if( sharemem_va_init(va) < 0){
     goto failed;
  }

  int child_pid = -1;
  if((child_pid = fork() )==0){
     char *va2 = (char*) 0x00200000;
     //printf(1," succeded on fork\n");
  if( sharemem_va_init(va2) < 0){
     goto failed;
  }
     //printf(1," succeded on share_mem in child\n");
     char *s1;
     if((s1 = read_share(6000)) == 0)
        goto failed;
     //char s3 = '1';
     for(int i =0;i<6000; i++){
        if(*s1 != '1'){
           //printf(1, "i is %d value read %d  should be %d \n", i, (int)*s, (int)s3);
           goto failed;
```

```
        }
        s1++;
      }
      printf(1," succeded on READ_share once in child\n");
      if((s1 = read_share(6000)) == 0)
        goto failed;
      //char s3 = '1';
      for(int i =0;i<6000; i++){
        if(*s1 != '2'){
          //printf(1, "i is %d value read %d  should be %d \n", i, (int)*s, (int)s3);
          goto failed;
        }
        s1++;
      }
      printf(1," succeded on read_share again in child\n");
      exit();
  }
  else{
      printf(1,"child pid is %d\n",child_pid );

      char* s = malloc(8096);
      for(int i =0;i<6000; i++){
        *(s + i) = '1';
      }
      write_share(s, 6000);
      printf(1, "wrote once by parent\n");
      for(int i =0;i<6000; i++){
        *(s + i) = '2';
      }

      write_share(s, 6000);
      printf(1, "wrote again by parent\n");

      wait();
  }


  printf(1, "share_mem ok ");
  exit();
  failed:
      printf(1, "test failed!\n");
      exit();
}

int
main(int argc, char *argv[])
{
  printf(1, "memtest starting\n");
  mem();
  return 0;
```

```
}
```