CS450 Fall 2019
Programming Assignment 3
Jingwen Pickens
Anthony Pollizi

Test Programs

These below test cases can be run by entering the **bold** name in the running XV6 shell

1. Test **sharedmem** uses getsharedpage(0,1) to get a single shared page of memory with key 0. The parent sets the value at the address returned by that call to 6. It then forks a child process which also uses getsharedpage(0,1) to get a single shared page of memory with key 0. The child process then enters a loop which can only be broken when the value at the first address in the shared memory page is equal to 6. After the loop is broken, it will set that value to 7. The parent then sets the value of the first address of the shared memory page to 6 and enters a loop which can only be broken when the child sets that value to 7.

   The expected result of this test is that the child process' loop will break when the parent executes the next line after forking, which sets the value necessary to break the child loop to the address where the child loop will look for that value. After the child loop breaks, it is expected that the parent loop will break as the child executes the instruction putting the value to break the parent loop in the address where the parent loop is looking for it.

   The actual result of this test is that the program hangs forever because the value set in the shared page by the parent process is not able to be read by the child process, and vice versa.

2. **Sharedmemfree** test program gets a shared page using key 0 and then sets a value in it, then prints that value. (Because printf clears the contents of shared memory pages, the value is then put back in the address.) Freesharedpage is then called using the key 0, and the value at the address is then printed again.

   The expected and actual output of this test is that the value put in the shared page will be printed the first time, then after being freed, the value will no longer be accessible at that memory address due to the removal of the page table entry in the process' page table.
   This indicates that the mapping of the virtual address to the shared physical address has been successfully removed in the freesharedpages function.

3. **Sharedmemreset** test program gets 5 pages using key 0, sets a value to the address returned, prints that value and the address where it is located, prints the exact same thing a 2nd time.
   The expected result of this test is that the two output lines are identical and match the value set to that address. The actual result of this test is that the first print behaves as expected, and the 2nd print shows that the same address now contains the value 0 instead of what was put there.

4. **Sharedmemfreenoget** test tries to free a shared page that has not been allocated. Doing so results in the message “Calling process not associated with that key”, which is the expected output of this test.

5. **Sharedmemnofree** test gets a shared page and then exits without freeing it. Doing so is expected to still result in the page being freed. This is the case, and the exit function which calls freesharedpage on all keys associated with the process when it the process ends, also prints a message saying “Shared page wasn’t freed before exit. Freeing…”

6. **Sharedmemdoublefree** gets a shared page and then frees the key associated with that page twice. This test is expected to first free the shared page and then attempt to do so again, which will throw an error stating that “Calling process not associated with that key” and also throw a trap 14 page fault.

7. **Sharedmemtwokeys** test allocates two shared pages each with different keys, and puts values in each. The first value is put in the first allocated page before the 2nd page is requested. The program then prints the values and frees the pages. It is expected that the contents stored in these pages will be printed, however only the 2nd page has its contents intact by the print statement. This suggests that the getsharedpage function getting the shared page for the other key also destroyed the contents of the first shared page.

8. **Sharedmemsamekeypage** test gets two shared pages both with the same key. Sets a different value to each address and then prints the contents of both addresses. This test is expected to print the second value set to the 2nd address, as the 2nd write to the address should override the first one if they are storing the value at the same location. This test instead prints the two different values assigned to those addresses, indicating that although internally the physical address to which the two virtual addresses are mapped is the same, the contents are not stored in the same place.

These test cases were selected to show what is expected to happen and what actually happens under a variety of ways in which the two system calls Getsharedpage and Freesharedpage can be used.
Test 1 is designed to execute to completion if and only if a value can be communicated between a forked child process and the parent process.
Test 2 is designed to ensure that using the Freesharedpage system call on a key which corresponds to a shared page with a value in it makes that value inaccessible to the process which has freed that key.
Test 3 is designed to demonstrate the bizarre functionality of the printf function to delete values stored in shared memory pages.

Test 4 is designed to ensure that using the Freesharedpage system call on a key which the calling process is not associated with has no effect and is acknowledged by an error printout in the kernel's execution of the system call.
Test 5 is designed to ensure that processes which use Getsharedpage but fail to free it before exiting still decrement the reference count of the shared page and allow it to be freed from physical memory if there are no other processes using that page.
Test 6 is designed to ensure that processes which try to free a page which they were once associated with but have already freed receive the appropriate error message informing it that that page has already been freed and that the calling process is no longer associated with that key.
Test 7 is designed to ensure that multiple shared pages can be used by a single process, and also demonstrates that the system call Getsharedmem also deletes values from the shared pages.
Test 8 is designed to ensure that if a process has requested the same shared page with the same key twice and writes a value into one reference to that shared page, that both references to that page share the same contents.