REFRESHER MODULE DESIGN DOC

NAME: APOORV SINGH
ROLL NO.: 2017027

System Calls

1) **int sys_attachpage (int key)**
   This system call calls the method shm_create(). The system call is implemented in sysproc.c and takes the key with which one should associate the shared memory with. It acquires the key and passes it to shm_create(). This returns the virtual address of the memory in case of successful completion, and -1 in case of any error. The call is basically to attach a page of memory to the process.

Functions

1) **int shm_create (int key)**
   The method is implemented in vm.c. The parameters are an integer key, with which the shared memory segment will be associated with. We maintain 2 arrays, one containing all the keys and the other containing the list of the corresponding physical addresses. The method first searches for a free virtual address of the process. It then checks whether a memory segment with the same key has been created before. If yes, the free virtual address is simply mapped to the physical address. Otherwise, we allot the memory in the physical address space, and map the virtual address. The virtual address is returned in case of successful operation, else, -1 is returned.

2) **int readBuffer (char* va)**
   This method is present in ulib.c. The parameter is the virtual address. It starts reading the memory starting at the virtual address. If the character is ‘A’, it means nothing has been written to the memory yet, and it waits for it to be written. Otherwise, it reads the memory, and writes it as ‘A’, signifying that the character has been read and can be overwritten. This continues until ‘.’ is encountered, which signifies end of the message. Once it reaches the end of the page, it again starts at the beginning of the page, i.e., the virtual address.

3) **int writeBuffer (char* buff, char* va)**
   This method is present in ulib.c. The parameters are the virtual address of the memory segment and the buffer or the message to be written. It starts writing to the memory starting at the virtual address, iterating through the value in the buffer. If the character is not ‘A’, it means the value has not been read yet, and it waits for it to be read. Otherwise, it writes ‘1’ to the memory. In the end, it writes ‘.’, which signifies end of the message.

Once it reaches the end of the page, it again starts at the beginning of the page, i.e., the virtual address.

TEST CASE

```
#include "param.h"
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"
#include "fcntl.h"
#include "syscall.h"
#include "traps.h"

int main(int argc, char *argv[])
{
	int key = 2;				//allots a key
	char* buff = (char *) malloc(6000);
	char* tmp1 = buff;

	/*
	attach the page to the current process. Virtual address is returned
	*/

	int va = attachpage(key);

	/*
	Constructing the message, or the buffer
	'.' indicates end of buffer. Buffer is populated by 1s.
	*/
	for (int i = 0; i < 5000; ++i)
	{
		*buff = '1';
		buff++;
	}
	*buff = '.';


	int pid = fork();
```

```
        if (pid == 0)
        {

                //The child process reads

                int va2 = attachpage(key);
                readBuffer((char *) va2);
        }
        else
        {
                if (va != -1)
                {

                        //The parent process writes

                        char* x = (char *) va;

                        /*
                        Populate the buffer by null values, in this case, 'A'
                        */

                        for (int i = 0; i < 4096; ++i)
                    {
                      *x = 'A';
                      x++;
                    }
                        writeBuffer(tmp1, (char *) va);
                }
                wait();
        }

        exit();
}
```