Systems Programming and Operating Systems	**Induprakas Kodukula**
CS414, Summer 1996	**Project Description, 7/25/96**

In this programming assignment, you are required to implement distributed shared memory on a network of workstations and write a client-server application that will run on this memory system interface. The purpose of the project is to familiarize yourself with *process creation and management*, *IPC*, *network programming* and *file management*.

You can think of distributed shared memory as a block of data that is spread out on a set of machines. Each machine stores a certain amount of the data. There is one process, called a **daemon** on each machine that manages the shared memory for that machine. If there are n machines that are using/sharing the shared memory, then each of the machines is assigned a unique integer in the range of $0 \dots n-1$. This can be done by a **hostfile** mechanism.

Once shared memory exists on a bunch of machines, we need some global way to address it. Let S be the size of the entire shared memory area on all the processes. Assume that S is divided by n with no remainder and is equal to S_n. S_n is the amount of shared memory on any individual process. Furthermore, the portion of shared memory on each process corresponds to a set of consecutive addresses, with the start address of the first byte being equal to $id * S_n$, where id is the unique id that has been assigned to each process(or) by the **hostfile** mechanism discussed earlier.

For example, let us assume that we have implemented 12 bytes of shared memory on 4 machines that have been named as brahms, mozart, hayden and salieri. Your host file may look like this:

```
/* Begin hostfile */
0 brahms
1 mozart
2 hayden
3 salieri
/* End host file */
```

In addition, your global shared memory will be numbered as follows:

```
               |               |               |
 --   --   -- | --   --   -- | --   --   -- | --   --   --
| 0|  | 1|  | 2||| 3|  | 4|  | 5||| 6|  | 7|  | 8||| 9|  |10|  |11|
 --   --   -- | --   --   -- | --   --   -- | --   --   --
               |               |               |
   brahms      |   mozart      |   hayden      |   salieri
```

For instance, if a process on brahms needs location 10 of the shared memory, it must communicate with salieri to obtain that byte.

Having talked about what the shared memory itself looks like, let's now proceed onto what functionality that should be provided to an application program that wants to use the shared memory that is managed by the daemons.

The following is the first stage of the project: To implement these calls that can be used by an application program. At this stage, you may assume that before an application program can use shared memory, on a particular processor, a **daemon** to manage the shared memory has already been started up by hand.

sh_init(int size, int nproc) does a per process initialization required for allocating a global shared memory of size bytes across nproc processors.

sh_read(gptr, nbytes, lptr) transfers nbytes amount of data from global address gptr to a process allocated array lptr. You can assume that the global memory being referenced using (gptr, nbytes) lies entirely on one processor.

sh_write(gptr, nbytes, lptr) trabsfers nbytes amount of data from process allocated array lptr to shared memory address starting from gptr. You can assume that the global memory being referenced using (gptr, nbytes) lies entirely on one processor.