CMPSC 311 - Introduction to Systems Programming

PennState

Systems and Internet Infrastructure Security Labratory

Assignment #3 – The Scatter/Gather Driver v1.1

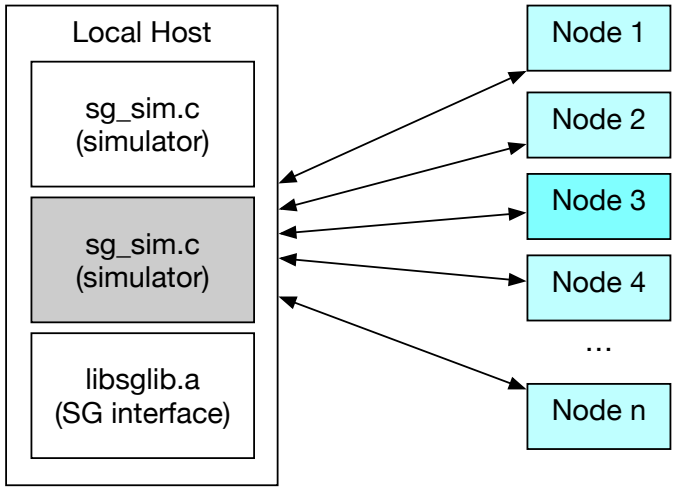
Professor Patrick McDaniel

Assignment #3 – Scatter/Gather v1.1

PennState

- The next assignment: You are to *extend* your device driver that sits between the virtual application and virtualized hardware devices.
  - As before, the application makes use of the abstraction you provide called the SG driver. You will make modifications to the code to implement several new features.

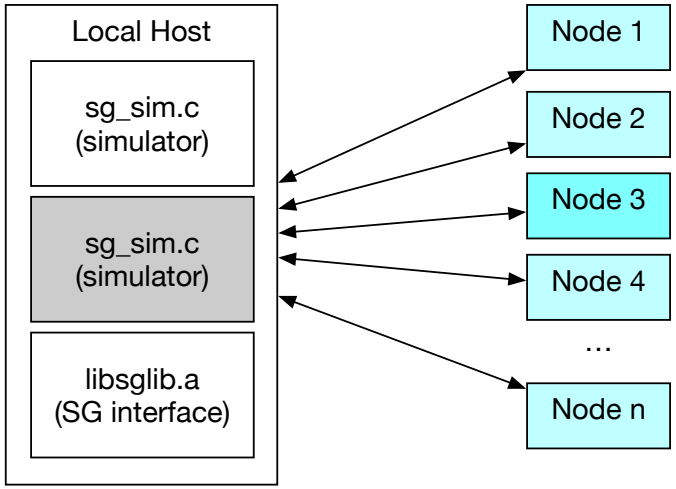
- You MUST use your code from assignment #2 and modify its function. This is called *refactoring* code, which is one of the most important skills you will learn in 311.



This assignment

PennState

- This assignment you will implement basic filesystem functions and surrounding code to provide a “local node” to the SG system. You will need to provide code to
  - Initialize the system (provided as example code)
  - Open virtual files
  - Read/write from the files
  - Close the files
  - Shut down the system.



- Your code will be implemented in `sg_driver.c`

Example of file I/O for assignment ..

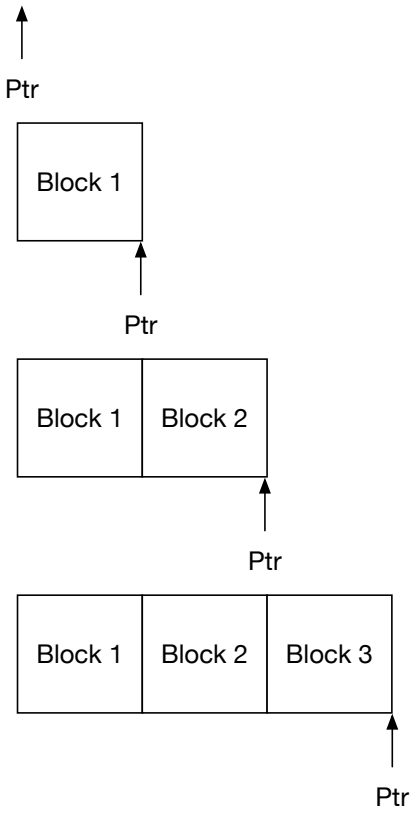
PennState

(1) sgopen() FH=7, size=0, pos=0

(2) sgwrite() 1024 FH=7, size=1024, pos=1024

(3) sgwrite() 1024 FH=7, size=2048, pos=2048

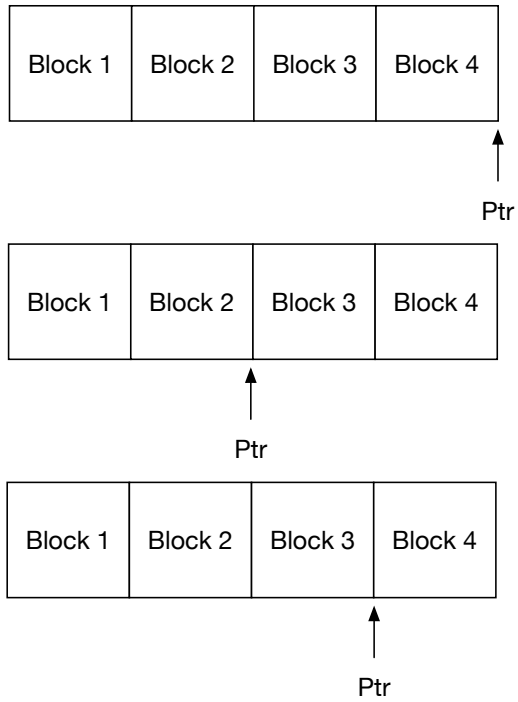
(4) sgwrite() 1024 FH=7, size=3072, pos=3072



(5) sgwrite() 1024 FH=7, size=4096, pos=4096

(6) sgseek() 2048 FH=7, size=4096, pos=2048

(7) sgread() 1024 FH=7, size=4096, pos=3072



Your driver (the functions you implement)

- `SgFHandle sgopen( const char *path );` - This function will open a file (named path, e.g.,) in the filesystem. If the file does not exist, it should be created and set to zero length. If it does exist, it should be opened and its read/write position should be set to the first byte. Note that there are no subdirectories in the filesystem, just files (so you can treat the path as a filename). The function should return a unique file handle used for subsequent operations or -1 if a failure occurs.
- `int sgclose( SgFHandle fh );` - This function closes the file referenced by the file handle that was previously open. The function should fail (and return -1) if the file handle is bad or the file was not previously open.
- `int sgread( SgFHandle fh, char *buf, size_t len );` - This function should read count bytes from the file referenced by the file handle at the current position. Note that if there are not enough bytes left in the file, the function should read to the end of the file and return the number of bytes read. If there are enough bytes to fulfill the read, the function should return count. The function should fail (and return -1) if the file handle is bad or the file was not previously open.

Your driver (cont.)

- `int sgwrite( SgFHandle fh, char *buf, size_t len );` - The function should write count bytes into the file referenced by the file handle. If the write goes beyond the end of the file the size should be increased. The function should always return the number of bytes written, e.g., count. The function should fail (and return -1) if the file handle is bad or the file was not previously open.
- `int sgseek( SgFHandle fh, size_t off );` - The function should set the current position into the file to loc, where 0 is the first byte in the file. The function should fail (and return -1) if the loc is beyond the end of the file, the file handle is bad or the file was not previously open.
- `int sgshutdown( void );` -- The function should shut down the system, including powering off the devices and closing all files.

S/G At a high level

PennState

- The scatter/gather system posts (scatters) blocks across many nodes, which are then retrieve (gathered) from the devices.
  - You will write code to perform this function.
  - The rules are:
    - You start the system with SG_INIT_ENDPOINT, stop with SG_STOP_ENDPOINT
    - Nodes are identified by node IDs (SG_Node_ID)
    - Blocks are identified by block IDs (SG_Block_ID)
    - If a block is new, you CREATE it (SG_CREATE_BLOCK) in which you are told where it was stored (a node ID) and what its unique identifier is (a block ID(
    - If a block is old, you gather the block back from the system (SG_OBTAIN_BLOCK) using the node and block ID returned from the create
    - You can update the contents of the block using the (SG_UPDATE_BLOCK) [NOT USED IN THIS ASSIGNMENT]

Send requests to the S/G system ....

- To send a request, you use the serialization/ deserialization code you wrote.
- The sgServicePost function sends the serialized packets and receives the response packet ...
- Note: I have provided a full code example in the function `sgInitEndpoint`

```
// Setup the packet
pktlen = SG_BASE_PACKET_SIZE;
if ( (ret = serialize_sg_packet( SG_NODE_UNKNOWN, // Local ID
                                 SG_NODE_UNKNOWN,   // Remote ID
                                 SG_BLOCK_UNKNOWN,  // Block ID
                                 SG_INIT_ENDPOINT,  // Operation
                                 sgLocalSeqno++,    // Sender sequence number
                                 SG_SEQNO_UNKNOWN,  // Receiver sequence number
                                 NULL, initPacket, &pktlen)) != SG_PACKT_OK ) {
    logMessage( LOG_ERROR_LEVEL, "sgInitEndpoint: failed serialization of packet [%d].", ret );
    return( -1 );
}

// Send the packet
rpktlen = SG_BASE_PACKET_SIZE;
if ( sgServicePost(initPacket, &pktlen, recvPacket, &rpktlen) ) {
    logMessage( LOG_ERROR_LEVEL, "sgInitEndpoint: failed packet post" );
    return( -1 );
}

// Unpack the recieived data
if ( (ret = deserialize_sg_packet(&loc, &rem, &blkid, &op, &sloc,
                                  &srem, NULL, recvPacket, rpktlen)) != SG_PACKT_OK ) {
    logMessage( LOG_ERROR_LEVEL, "sgInitEndpoint: failed deserialization of packet [%d]", ret );
    return( -1 );
}
```

Posting field use (by operation)

- For this assignment, use this table to set the packet fields (by operation type)
  - Sender packet – what you send to the system
  - Response packet – what you can expect from the system

Sender packet (your code)							
Local ID	Remote ID	Block ID	Op	Sseq	Rseq	Data	Block
SG_NODE_UNKNOWN	SG_NODE_UNKNOWN	SG_BLOCK_UNKNOWN	SG_INIT_ENDPOINT	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	SG_NODE_UNKNOWN	SG_BLOCK_UNKNOWN	SG_STOP_ENDPOINT	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	SG_NODE_UNKNOWN	SG_BLOCK_UNKNOWN	SG_CREATE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	1	Block to create
Local ID	Node holding block	Block ID to update	SG_UPDATE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	1	Block to update
Local ID	Node holding block	Block ID to obtain	SG_OBTAIN_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	
Local ID	Node holding block	Block ID to delete	SG_DELETE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Response Packet							
Local ID	Remote ID	Block ID	Op	Sseq	Rseq	Data	Block
New Local ID	SG_NODE_UNKNOWN	SG_BLOCK_UNKNOWN	SG_INIT_ENDPOINT	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	SG_NODE_UNKNOWN	SG_BLOCK_UNKNOWN	SG_STOP_ENDPOINT	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	Node ID of new block	Block ID of new bock	SG_CREATE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	Node holding block	Block ID of update	SG_UPDATE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A
Local ID	Node holding block	Block ID of obtain	SG_OBTAIN_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	1	Obtained block data
Local ID	Node holding block	Block ID of delete	SG_DELETE_BLOCK	SG_SEQNO_UNKNOWN	SG_SEQNO_UNKNOWN	0	N/A

Simplifying Assumptions ...

PennState

- You can assume the following for this assignment (this makes the project way easier, trust me).
  - All reads and writes will be exactly SG_BLOCK_SIZE long (1024)
  - All reads and writes will be on a block boundary.
  - You will not have to do any SG_UPDATE_BLOCK or SG_DELETE_BLOCK operations.
  - The files will not contain more than 20 blocks.
  - You can ignore the sequence numbers for the purposes of this assignment.
- Note: do not try to use code from a previous semester. This is a very different assignment (although it looks similar).

Honors Option

PennState

- Use the gcrypt encryption functions to encrypt every block
  - Fix a 256-bit AES key
  - Encrypt the data block in the serialization function
  - Decrypt the data block in the deserialization function

Pseudocode for assignment

PennState

- The first thing that will be called is your sgopen() call

1) Check to see if initialized, call the initialization function if needed

2) Assignment a file handle (an arbitrary integer unique identifying the filename/path)

3) Add the file to some data structure recording the mapping of the file handle to the filename

    3a) Set the file pointer to 0

    3b) Set the file size to 0

4) Return

Pseudocode for assignment (cont)

PennState

- Next, you will receive a set of sgwrite()s

1) Check if file handle to see if assigned before, error if not
2) If file pointer points to end of the file
   2a) Create a new block containing the contents of the write [SG_CREATE_BLOCK op]
   2b) Save node/block IDs as the current block in the data structure
   2c) Increase the file size by length of write
   2d) Set file pointer to the end of the file
3) If file pointer NOT at end of file
   ABORT--this should never happen in this assignment
3) Return number of bytes written

Pseudocode for assignment (cont)

PennState

Thereafter, you will receive serval sgseek()s and sgread()s:

Seek:

1) Check if file handle to see if assigned before, error if not
2) Check if seek position <= file size
3) Set file position to the seek position
4) Return the seek position

Read:

1) Check if file handle to see if assigned before, error if not
2) If file pointer points to end of the file, error reading beyond end of file
3) Look at data structure and figure out what block to retrieve
4) Retrieve the block [SG_OBTAIN_BLOCK op]
5) Copy the data from the retrieved block to passed in buf
6) Update the file position by adding len
7) Return the number of bytes read

Pseudocode for assignment (cont)

PennState

Eventually, you will receive a call to sgclose();

1) Check if file handle to see if assigned before, error if not
2) Clean up data structure, mark file as closed
3) Return 0 (success)

FAQ

PennState

- Do we use Unix open, read, etc?
  - You are writing the device driver code (open, close, etc.) so that the application (sg_sim) can interact with the storage devices (see figure from slide 2). Therefore, you do *not* have access to Unix open, close, etc. You must implement them.
- How do we implement them?
  - Looking at the pseudocode, you notice that you must use some data structure(s) to store information about each file whenever it is opened (name, handle, length, … blocks used, etc.). When sg_sim calls your functions, you look at your structures and use SgServicePost to interact with the storage devices as needed (by passing it your serialized packet containing the operation info).
- I'm lost.
  - You must organize the information about the files, as the application is calling your functions to execute certain operations. Start with open and determine how you can record file information that can then be used later when a read/write is called (hint: initialize your structure…)