CMPSC 311 - Introduction to Systems Programming

PennState

Systems and Internet Infrastructure Security Labratory

Assignment #5 – The Scatter/Gather Driver v1.3

Professor Patrick McDaniel

Scatter/Gather 1.3 (Last assignment!)

- You are to complete the device driver that sits between a virtual application and virtualized hardware devices.
- This fourth assignment will add four features/requirements:
    - more complex workloads
    - increased cache size
    - sequence numbers
    - hidden workload.

- As before, the sg_sim application is provided to you and will call your device driver functions with the basic UNIX file operations (open, read, …). You are to write the device driver code to implement the file OS operations.

Project notes ....

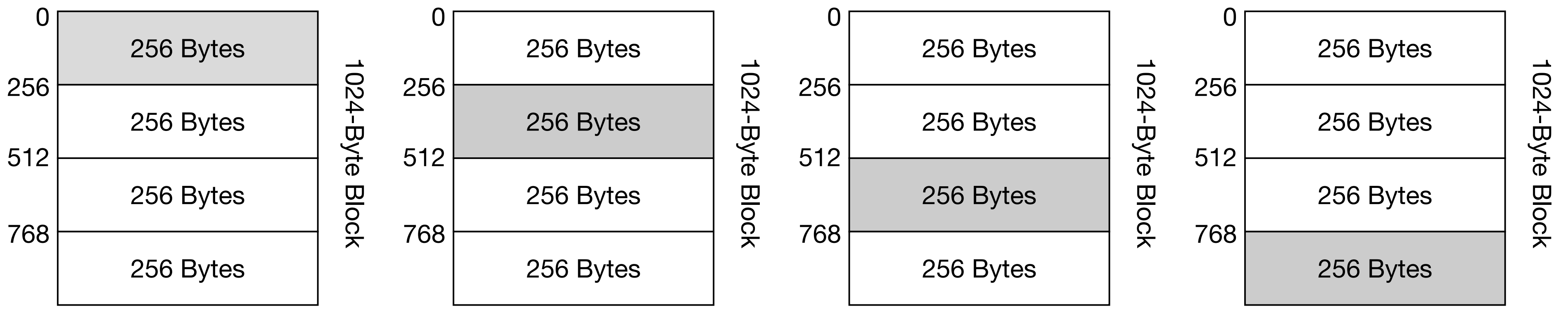
- You MUST use your code from assignment #4 and modify its function. This is called *refactoring* code, which is one of the most important skills you will learn in 311.
- NOTE: The due date for this project is hard (the last day of class). There is no 10%/day extensions for late submissions and

THERE WILL BE NO EXTENSIONS OF ANY KIND GIVEN

(we need to have the projects to grade to assign final grades in time).

Complex workloads ....

- The first feature of this assignment is to handle more complex file input and output for the system. You will implement the system with quarter block reads and writes (256 bytes).
  - Like assignment #3, only with quarter block reads/writes
  - So, there are 4 options on reads and writes ....



Cache size increase

- The second feature to be to increase the LRU cache to include many more cache lines.
- See the SG_MAX_CACHE_ELEMENTS variable in the sg_cache.h file.
- Otherwise the cache should perform in all other respects as assignment #3 (including outputting the cache hit rates).
    - Note: If you have implemented you cache in a flexible way, this will require no changes to your current code.

Cache Line	Node ID	Block ID	Data
0	667334..	892389..	`bDgk{DC$e91s7L^LYH*};][!3T}tgwa-8!A2}Bi`
1	567234..	418585..	`iWF!t#.WJQs0/M91ZVks7L^LYH*};]FRyza[!3}`
2	0	0	000000000000000000000000000000000000000
3	0	0	000000000000000000000000000000000000000
...			
N	0	0	000000000000000000000000000000000000000

Sequence numbers ….

- The third feature is to manage local and remote (node) sequence numbers.
  - Local sequence numbers are used to order the requests being posted by your driver.
  - These should be passed to the serialization function via the `sseq` variable.
  - Should be received by the deserialization function via the `sseq` variable
  - Rules
    - Initially set the local sequence number to SG_INITIAL_SEQNO
    - Each time you post a request, use the local sequence number, and increment it by one AFTER you post,
      - E.g., first post call is SG_INITIAL_SEQNO, then SG_INITIAL_SEQNO+1, then +2 …
    - If you don't do this correctly, you may see an error saying "out of sequence"

Sequence numbers ….

- The third feature is to manage local and remote (node) sequence numbers.
  - Remote sequence numbers are used to order requests received from remote nodes.
  - These should be passed to the serialization function via the `rseq` variable.
  - Should be received by the deserialization function via the `rseq` variable
  - Rules
    - The first time you receive a response from a note (identified by the rem variable), you save the returned sequence number rseq
    - Each time you post a request, use the last remote sequence number of the node you are OBTAINing or UPDATEing
    - Check the remote sequence number returned from the post is what you sent, plus 1 (i.e., the remote node will add one to the value you sent)
    - If you don’t do this correctly, you may see an error saying “out of sequence”

Sequence numbers ....

- The third feature is to manage local and remote (node) sequence numbers.
  - Remote sequence numbers are used to order requests received from remote nodes.
  - These shou able.
  - Should be r le
  - Rules
    - The first rem variable), you save
    - Each tim er of the node you are
    - Check th t you sent, plus 1 (i.e., th
    - If you don't do this correctly, you may see an error saying "out of sequence"

Hint: The way to think about sequence numbers is that you increment the local sequence number each time you send a new post, and each remote node increments the sequence number each time it receives a request. Your job is to remember what the most recent one for the local and each remote is.

Hidden workload ....

- The last feature is that you will be tested against a workload you have not seen (and will not be given). What you know:

    1. It will include quarter block reads and writes as in this assignment
    2. Include any number of files
    3. The files many be of any arbitrary size.

- Hint: Your code MUST use dynamic allocation of structures and tables to process this workload, as you don't know how big the workload will be and how it will look.
- Note: 30% of the grade for this assignment is determined by your code's ability to process this workload.

Honors option

- Honors option: Modify your code to hash blocks (using something like the gcrypt MD5 function) before sending (posting).
  - You need to hash every block coming back from the Scatter/Gather system and compare the hash values. If they differ, you should return an error from the read/write.

CMPSC 311 - Introduction to Systems Programming

Systems and Internet Infrastructure Security Labratory

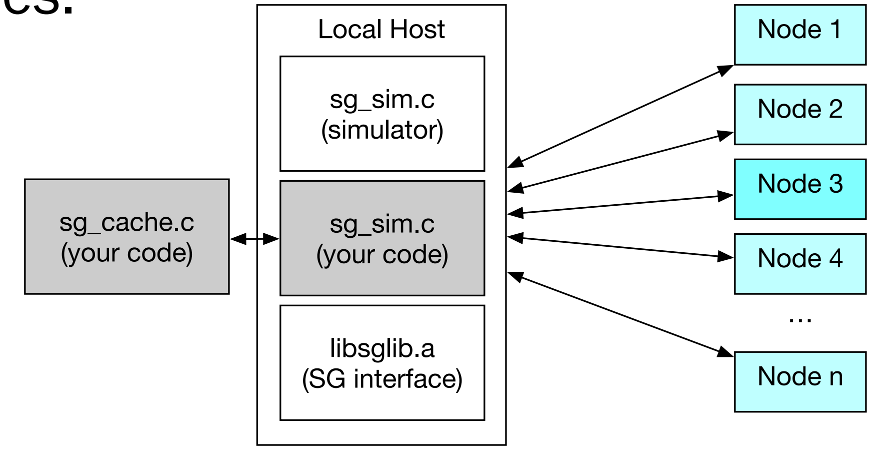
Assignment #4 – The Scatter/Gather Driver v1.2

Professor Patrick McDaniel

Assignment #4 – Scatter/Gather v1.2

- 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.

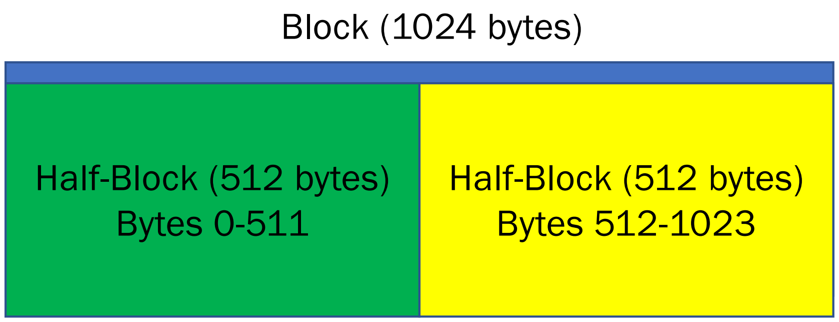
- You MUST use your code from assignment #3 and modify its function. This is called *refactoring* code, which is one of the most important skills you will learn in 311.



This assignment requires 2 modifications.

Modification #1 - Workload Complexity

- The workload is now changed to reflect more realistic computing activity
  1. Reads and writes are ½ blocks (512 bytes), but will always by on the block boundary (0) or half-block boundary (512)
  2. Workload will have multiple files, and each file may be a different length
  3. Updates to blocks, that is writing to the middle of files


Block (1024 bytes)
Half-Block (512 bytes) Bytes 0-511
Half-Block (512 bytes) Bytes 512-1023

Hint: when writing to the middle of a file, you need to (a) grab the block, (b) copy the data over, and (c) write the block back using the SG_UPDATE_BLOCK.

The new workload is `cmpsc311-assign4-workload.txt`

Modification #2 – Adding a cache

- Implement an LRU cache that caches read/written blocks.
  - The API for the cache is provided in sg_cache.h and the starter code for the implementation is in sg_cache.c.
  - The cache should be fully associative (any block can go in any cache line). It should be dynamically allocated when initSGCache() function is called. Each line should contain the node/block and data.
  - The size of the cache is defined as SG_MAX_CACHE_ELEMENTS (in sg_cache.h)
  - Add code to call the functions from your driver implementation. Any place you need to get a block from the SG system, you should check first if it is in the cache.
  - You should collect statistics on how many hits and misses you have. On close, print out the values and your hit ratio when you close the cache.

- **Requirement**: watch the lecture on caching before attempting this …

Cache functions

```
int initSGCache( uint16_t maxElements );
    // Initialize the cache of block elements
    // maxElements indicates the size of the cache in cache lines


int closeSGCache( void );
    // Close the cache of block elements, clean up remaining data


char * getSGDataBlock( SG_Node_ID nde, SG_Block_ID blk );
    // Get the data block from the block cache
    // nde is the node, blk is the block


int putSGDataBlock( SG_Node_ID nde, SG_Block_ID blk, char *block );
    // Get the data block from the block cache
    // Hint: make sure you allocate the block data for the cache before adding
```

A possible cache structure

PennState

- The cache needs to hold copies of data that were placed into the cache
- So make a structure, hold data in it, for example …

Cache Line	Node ID	Block ID	Data
0	667334..	892389..	`bDgk{DC$e91s7L^LYH*};][!3T}tgwa-8!A2}Bi`
1	567234..	418585..	`iWF!t#.WJQs0/M91ZVks7L^LYH*};]FRyza[!3}`
2	0	0	`000000000000000000000000000000000000000`
3	0	0	`000000000000000000000000000000000000000`
…			
N	0	0	`000000000000000000000000000000000000000`

Hint: 0 indicates that the cache entry is unused (0 is not legal for node or block IDs)

- Note that the cache must be a write through cache.

Honors option

- Honors option: Create an independent LFU cache with its own interface (perhaps creating a new .h and .c file. Modify the sg_sim file and add a command line option to use the LFU cache instead of the LRU cache.
- You should provide an output showing the performance of each cache as well as the total performance when you close the cache.

CMPSC 311 - Introduction to Systems Programming

PennState

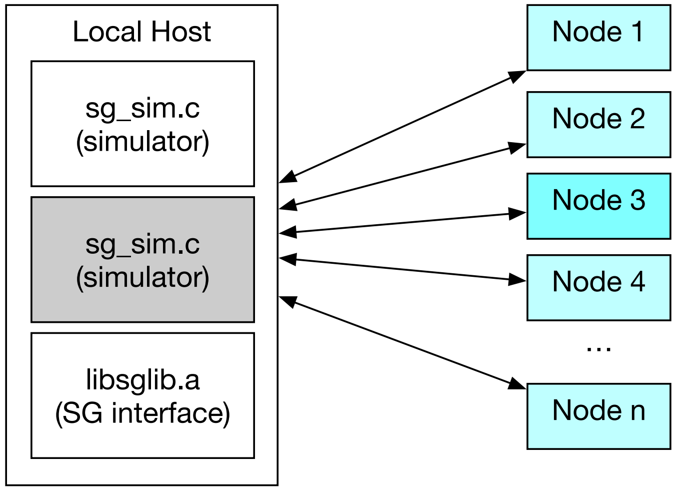
Systems and Internet Infrastructure Security Labratory

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

Professor Patrick McDaniel

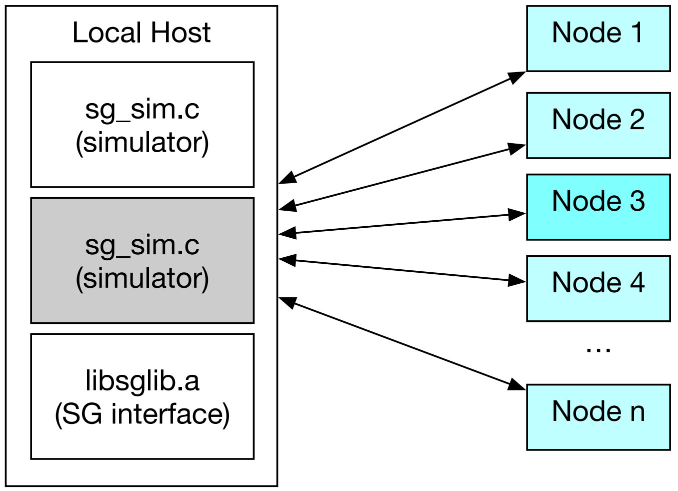
Assignment #3 – Scatter/Gather v1.1

- 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

- 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 ..

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

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

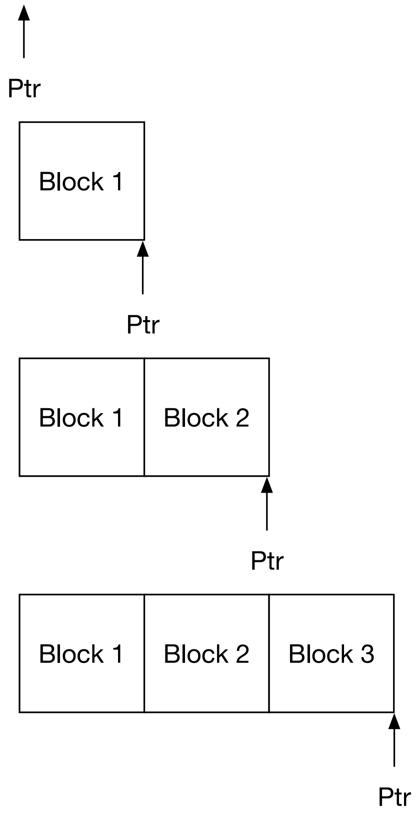
(3) sgwrite() 1024 FH=7, size=2048, pos=2048

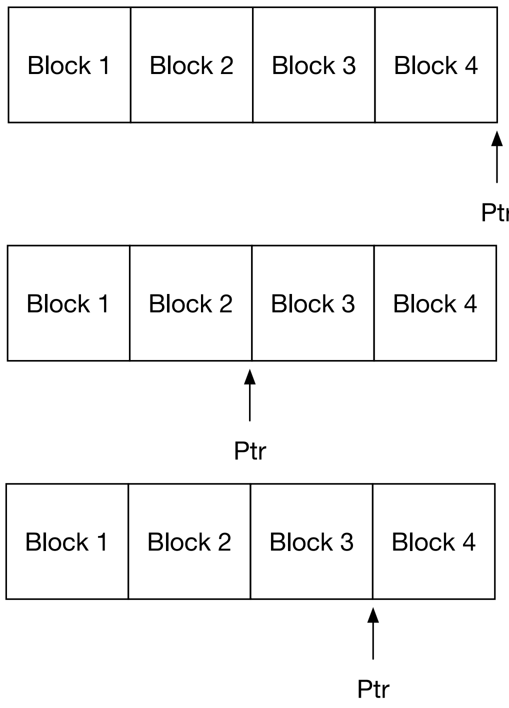
(4) sgwrite() 1024 FH=7, size=3072, pos=3072

(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.

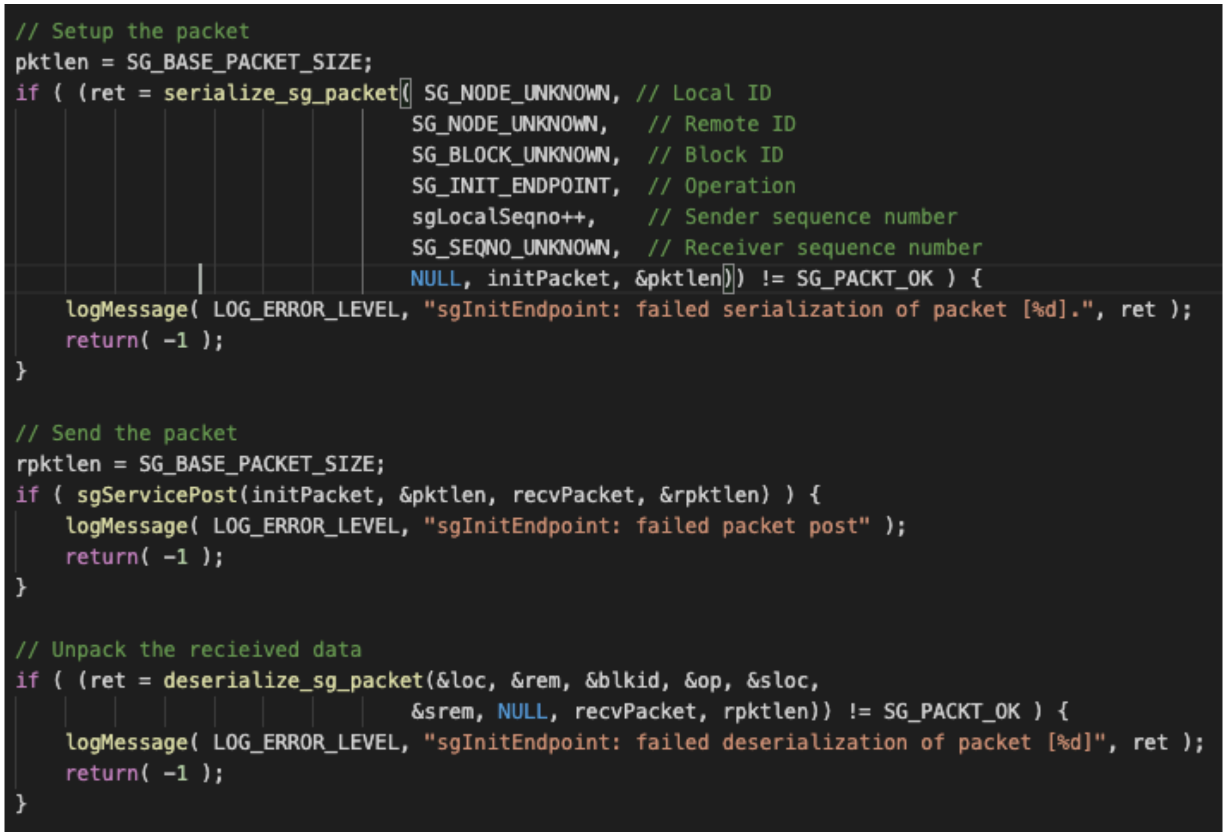
S/G At a high level

- 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`



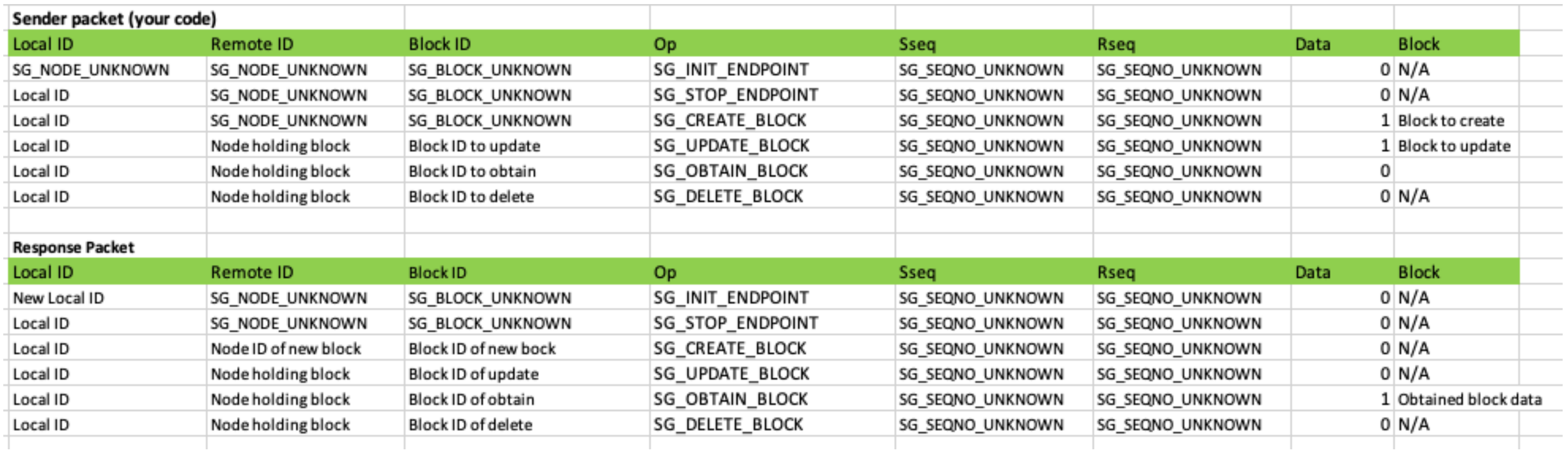
```c
// Setup the packet
pktlen = SG_BASE_PACKET_SIZE;
if ( (ret = serialize_sg_packet( SG_NODE_UNKNOWN, // Local ID
                                SG_NODE_UNKNOWN,    // Remote ID
                                SG_BLOCK_UNKNOWN,   // Block ID
                                SG_INIT_ENDPOINT,   // 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 …

- 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

- 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

- 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)

- 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)

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)

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

- 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…)