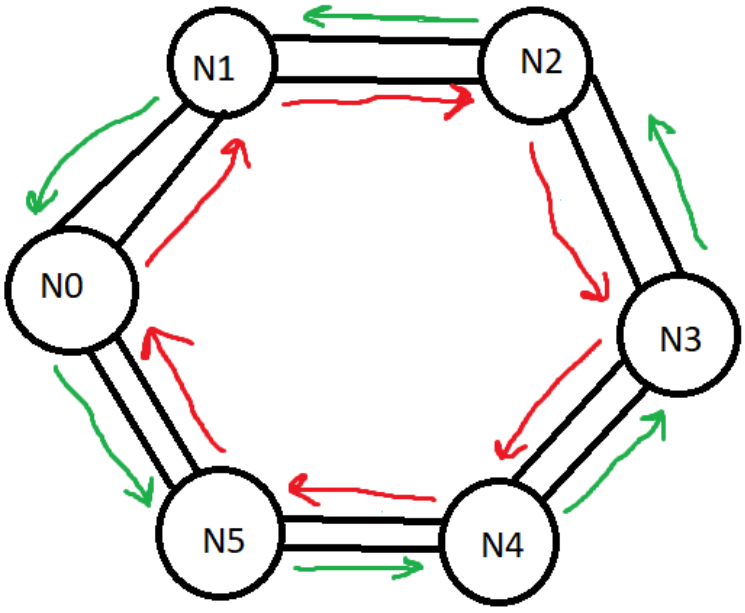
For this project, the goal is to create a ring of nodes that are connected to its neighbors. These nodes will have one way communication. For instance, the left side neighbor will be read from by the middle neighbor and will write a connection to its right.

Our goal is to be able to allow a user to choose how many nodes there are, clarify a destination node which will be the recipient of the message, and write the message to be transmitted. The message will have to be passed off between nodes till it reaches the destination. Then the header for the message will be EMPTY till it returns back to node 0 (parent node). The user would then terminate the program by using ^C.

The image below displays the process for k=5 nodes with N0 being the parent node. In the direction of the red arrows, the node is writing to the next node. In the direction of the green arrows, the node is reading from the prior node. This is done by connecting the two nodes with a pipe which can be visualized as the two black lines that connect each node.



The parent node will be responsible for creating the other nodes, taking in user inputs, and receiving the empty message at the end. The nodes will be created by generating child processes at the start when the user is prompted with a message. This will be done in the code by creating a fork. These child nodes will check to see if they are the desired destination. If they are not, they will pass the message on to the next node. If they are, it will clear out the message header and go through each node till back to the parent node.

To implement this, I had to utilize forks, signals and pipes. I first asked the user how many nodes they wanted for the ring. I then allocated the pipes for each node. After this the child process would be forked and assigned a node_id which will now be running their own node_main().

Node_main() handles all of the functions for the nodes in regards to reading and writing utilizing the pipes. It would print out the file descriptors it reads and writes from.

After the fork, then the signal handlers were incorporated. When parent_sigint_handler was triggered by a user pressing ^C, it shuts down the children and the parent will wait till all children terminate before terminating itself. SIGINT for the parent being triggers SIGTERM for the child_sigterm_handler which would terminate the child process.

The next step was to assign the read and write file descriptors to the nodes where the process aforementioned is done.

The child processes will then start their infinite loops but will be blocked till there is data in the pipe. This leads to the parent process

to ask the user for the message and destination node. Then using a header structure, it contains the destination and length of the message that will be sent. The parent then sends the message to the next node through the pipe and the children will do the same till the message reaches its destination. Once at its destination, the header will have destination as the parent node with no message. This is the empty apple part.

Once it reaches back to the parent node, the parent recognizes there is no longer a message and then will ask if the user wants to send another message and where its destination would be. Otherwise the user will press ^C and shutdown the child processes and parent process as mentioned prior utilizing the signals.