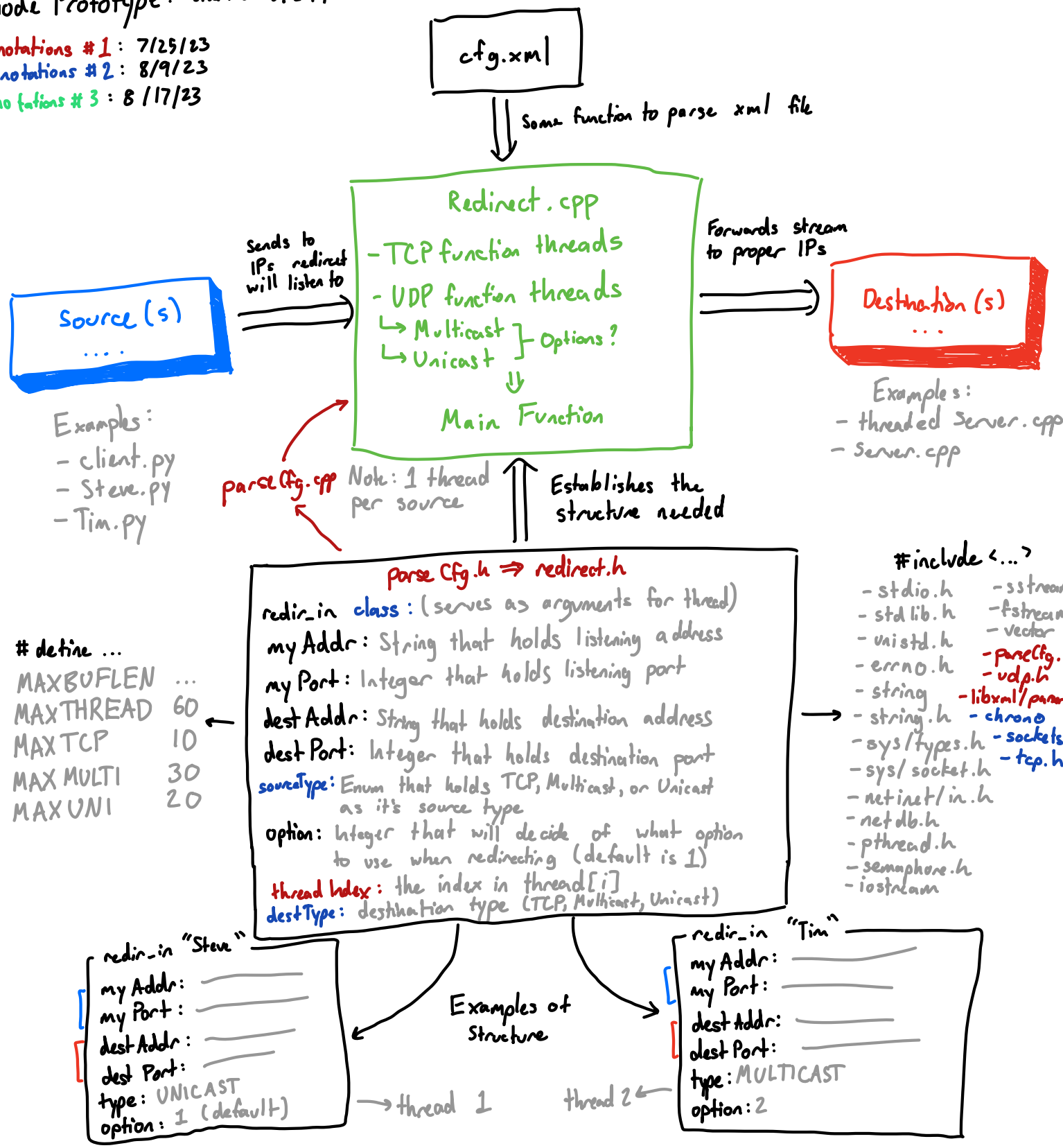
diode Prototype: started 6/29/23

Annotations #1: 7/25/23
Annotations #2: 8/9/23
Annotations #3: 8/17/23



General Idea: Each Source will have a redir_in variable that will contain all the necessary information to redirect a source to the right destination (including options & types to help). Each thread will decide on which function it needs to call based on configuration file and will "fill out" the information provided to redirect the stream.

Config File:
- Multiple instances of redir_in (different IP sources & destinations)
- Will have a function that parses the .xml into an array of redir_in (now inside parseCfg.cpp)
- Each object entry will be its own thread

Implementation Plans:

Global Variables:

int isAlive - determine if all threads should still run after receiving and redirecting a packet

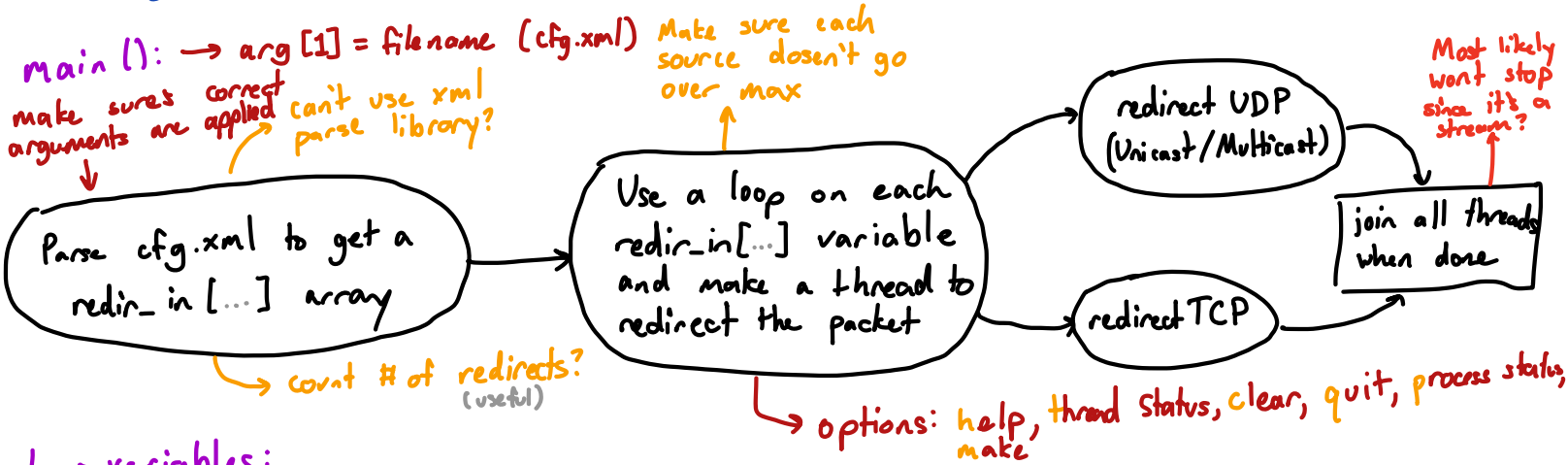
int allThreadMade - signal to start all threads

int threadCount - counts how many threads are made (so far)

redir_info [MAXTHREAD] - list of all redirection information parsed from cfg.xml

std::chrono::time_point<std::chrono::steady_clock> threadStats [MAXTHREAD] - time data of when threads last assign their own "thread stats"

main():



↳ variables:

pthread thread [MAXTHREAD] - list of all threads that the program will run

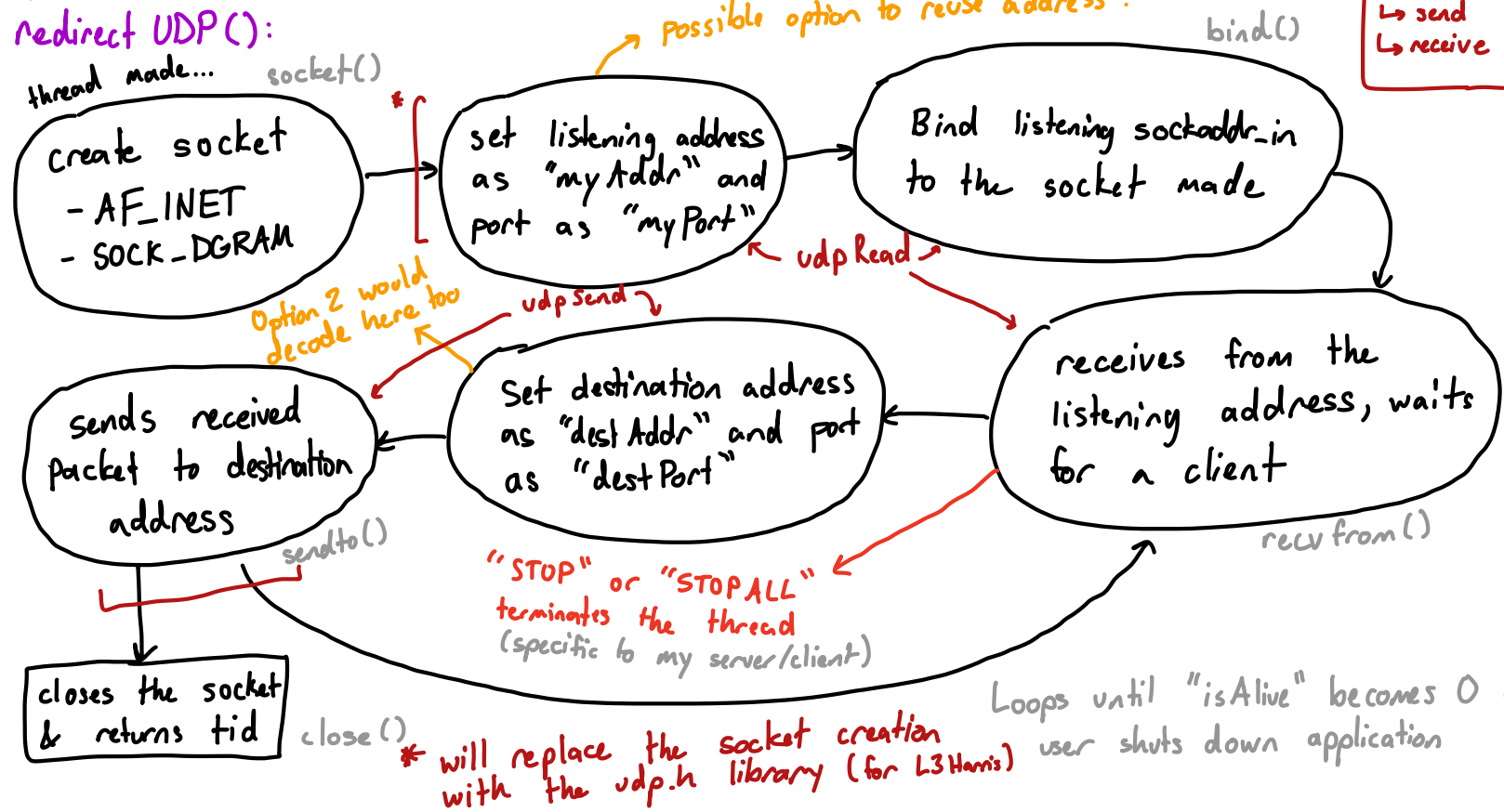
int tcpCount, multiCount, uniCount - these integers keep the count of how many sources are/were running in threads

↳ thinking main() manages the count to avoid data racing

int numJoined - counts number of threads joined while main() waits for threads to exit

int numRedir - number of times redir_in appears in cfg.xml

redirectUDP():



↳ variables

udp *udpRead, *udpSend - UDP interfaces that contain socket information (fd, addr, etc)

U8 buffer[MAXBUFLEN] - used for packet sending

int ttl - used for multicast subscription layers

int waiting - flag to indicate that thread is currently waiting on data from client

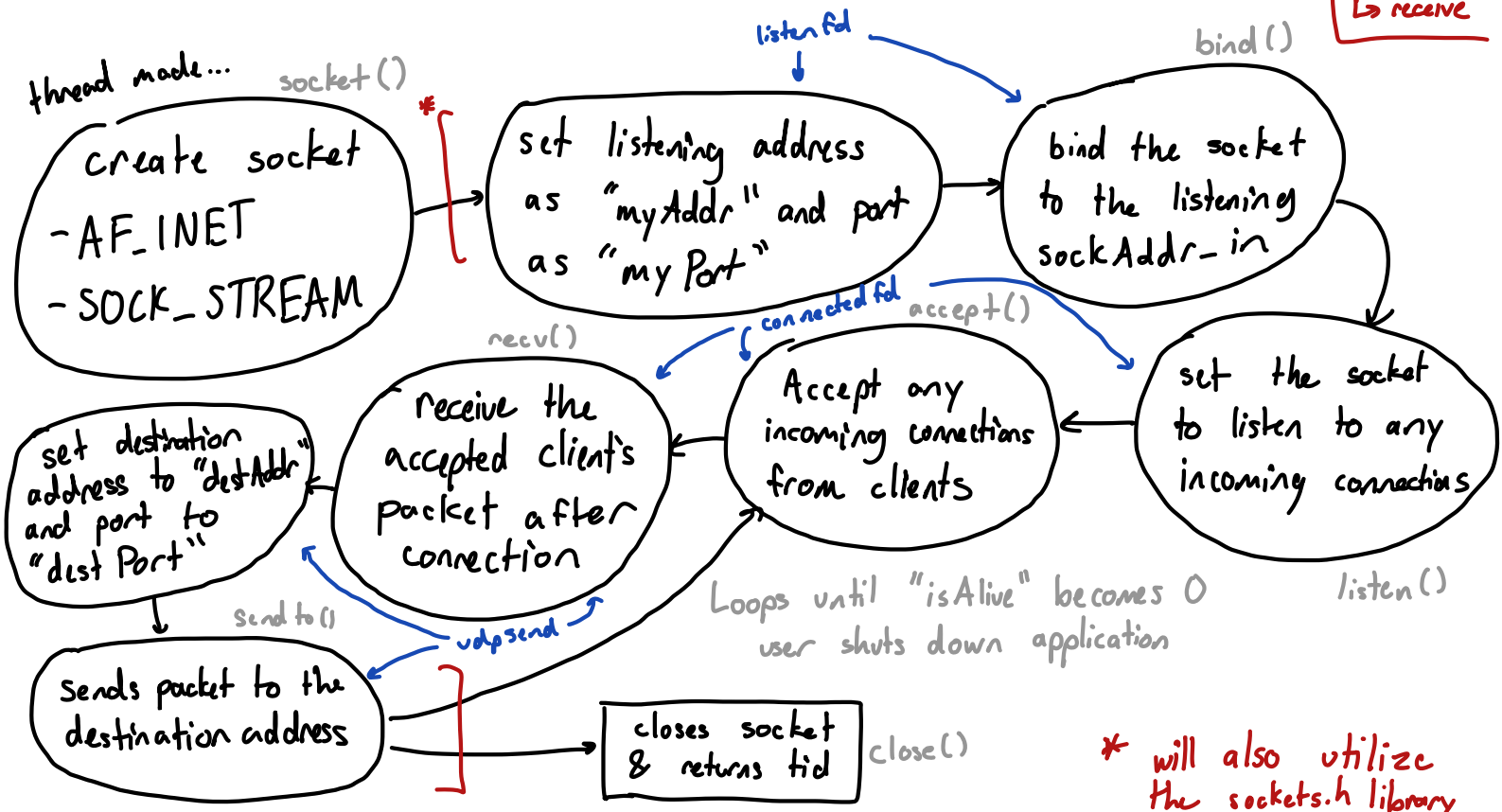
socklen_t len - holds the size of buffer

int n - the number of bytes received

static int returntid - return thread ID (helpful for tracking)

redirectTCP():

2 sockets
↳ send
↳ receive



* will also utilize the sockets.h library instead (tcp.h is not useful here unfortunately)

↳ variables:

int listenfd, connectedfd - file descriptors for the listening & accepted sockets

struct sockaddr_in CliAddr - client address information structure

socklen_t cliLen - cliAddr memory size

U8 buffer[MAXBUFLEN] - buffer used to hold stream data

udp *udpSend - Holds the socket containing information to where the buffer should be redirected

int waiting - flag that indicates if the program is waiting for information from the client

parseXML(): Now in parseCfg.cpp → <libxml/parser.h>

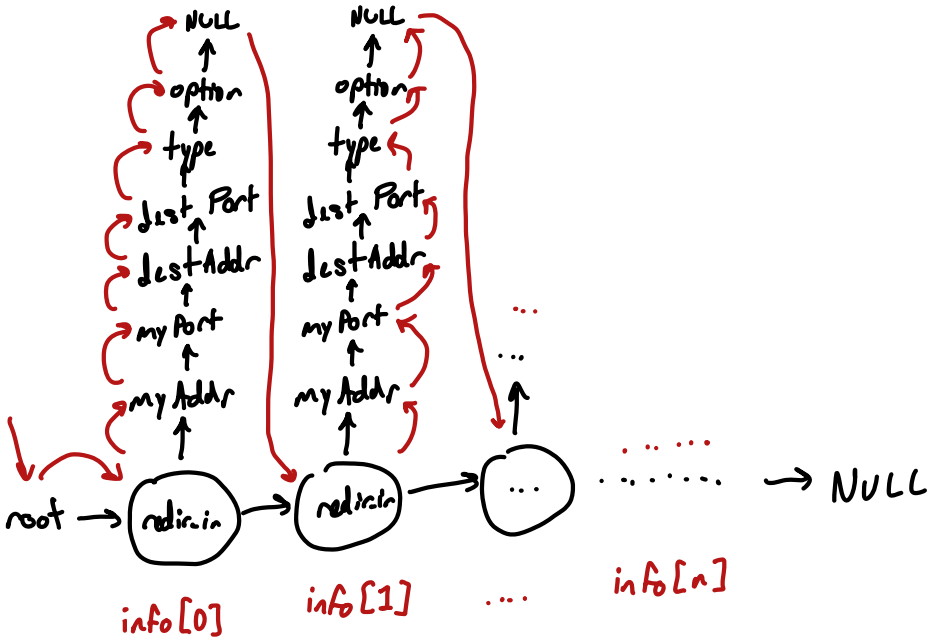
It also prints out the raw cfg.xml file for visual purposes

- Goal: have it create a list of redir_in variables from parsing a xml file containing the configuration (the end of the list is the first "blank" struct)
- Parameters:
  - ↳ char *filename - this is the string of the file name arguement (aka arg[1])
  - ↳ redir_in *output - where the list data will be outputted to (info[])
- xml format:

cfg.xml

```
<config>
    <redir_in>
        <myAddr> ——— </myAddr>
        <myPort> ——— </myPort>
        <destAddr> ——— </destAddr>
        <destPort> ——— </destPort>
        <sourceType> ——— </sourceType>
        <option> — </option>
        <destType> ——— </destType> ← Extra feature for redirection
    </redir_in>
    .... more instances of this ....
</config>
```

Data structure (parsed)



traverse through the xml file by first iterating through each redir_in and going through it's contents and copying it's data to it's respective area in each index of info[]

printRedir():

- Goal: prints out the listening IP address/port as well as the destination IP address/port on separate lines (Used for tracing & debugging)

- Parameters:
  - ↳ redir_in info - the redir_in structure you want to print on the terminal

- Example:

```
*terminal stuff*
Incoming IP: ~~~~~~~~~~~~
Destination IP: ~~~~~~~~~~~~
- - - - - - - - - - - - - - - - - -
*terminal Stuff*
```

output

makeThread():

- Goal: Utilize the pthread_create() function and also keep track of the number of threads made (mainly for conveince)
- Parameters:
  - ↳ pthread_t *thread - reference thread variable used when making a thread
  - ↳ const pthread_attr_t *attr - thread attributes, usually set to NULL
  - ↳ void *(*tFunction)(void *) - function pointer that the thread will run on
  - ↳ void *arg - the selected redir_in structure used in the thread; the parameters the function will use in the created thread

*It's basically the exact same as pthread_create but increments threadCount

currThreadStats():

- Goal: Have this run whenever 't' is pressed, it will display of which thread at each index of thread[] and it's time status
- The threads are responsible at making sure that their reserved spot in threadStats[] is pinged on a regular basis (if still alive)
- For TCP, it will not ping until a connection is secured
- Example:

```
o --- Time Since Last Ping --- o
  Thread 0: 0 seconds ago
  Thread 1: 0 seconds ago
  Thread 2: 0 seconds ago
o- - - - - - - - - - - - - - - o
```

} Time tracked using chrono library

Hardware Performance Optimization

Original Implementation:
- While the thread waits for an incoming packet, it will sleep for 0.25 seconds
- While the main thread waits to join, it will sleep until it gets a user input or if a thread joins

New Implementation:
- Utilize select() to create a 1 second timeout when a thread reads no data, it will preform redirection once it recieves data
- main thread functions like the old implemantation
- Implemented in the redirection part of both TCP & UDP
- Went from 99.8% CPU ⟶ 0.7% CPU

↳ Variables

fd_set readSet - file descriptor of the select function
struct timeval timeout - set to 1 seconds of timeout
int result - return value of the select() function, tracks changes

C++ Object Oriented Structure Overhaul (redirect++.cpp & redirect++.h)

The initial program was structured in a way where it was more procedural like c rather than C++. I was advised to utilize the object oriented features of C++ for this program. For the most part, the variables are the same but the functions are now in different places (either global functions or class functions)

Redir_in.h

- This is now it's own header file & is a class + enum instead of a structure, it holds the redirection information based on cfg.xml

Constructors:

Redir_in()

↳ Strings are set to NULL
↳ Port numbers are set to 0
↳ Option is set to 1
↳ threadIndex & tid are set to -1
↳ sourceType is set to UNICAST & destType is set to MULTICAST

The default constructor will set these default values & helps identify the end of the list in the "info[]" global variable

Redir_in (...) ← All class variables (in order)

↳ will set each parameter to it's associated value, self explanitory

Class Variables:

const char *myAddr → listening address
int myPort → listening port number
const char *destAddr → destination address
int destPort → destination port number
enum SourceType sourceType → incoming source type
int option → sending option
enum SourceType destType → outgoing source type
int threadIndex → Index of thread in "info[]" } not in cfg.xml
int tid → thread ID

Class Functions:

void printRedir()
int redirectUDP()
int redirectTCP()

These functions are now located inside of the class and will be called by the redirect() thread function

Note: ParseCfg files are generally the same

redirect++.h

includes:

- udp.h
- tcp.h
- sockets.h

} L3 Harris Specific Libraries

- parseCfg.h
- stdio.h
- stdlib.h
- unistd.h
- errno.h
- string
- string.h
- sys/types.h
- sys/socket.h
- netinet/in.h
- arpa/inet.h
- netdb.h
- pthread.h
- semaphore.h
- iostream
- sstream
- fstream
- vector
- sys/select.h
- termios
- fcntl.h
- chrono

Macros:

MAX BUFLEN = 100
MAX THREAD = 60
MAX TCP = 10
MAX MULTI = 30
MAX UNI = 20

Note: There is a new "COLORS" section of macros to have color codes to make the terminal look "stylish"

pingInfo structure:

↳ has int flag & std::chrono::time_point<std::chrono::steady_clock> *timestamp
↳ basically holds info for "pingTilAccept()" function

Global Functions:

int makeThread(pthread_t *thread, const pthread_attr_t *attr, void*(*tFunction)(void*), void *arg)
void currThread Stats(int numOfThreads)

These functions are kept the same (refer to above writings about them)

void *pingTilAccept(void *arg)

↳ Thread function that works alongside a TCP thread to keep pinging that it is alive by updating its timer during it's waiting for connections blocking statement
↳ stops once the flag is 0 (meaning TCP is connected)
↳ Very flawed way to keep track thread is alive

void *redirect(void *arg)

↳ Thread function for every thread now
↳ This function will manage which class function to call (TCP or UDP)
↳ Sets up tid & returns it as well

Overall, the general reformatting does not change functionality whatsoever and functions the same. It is now more object oriented and fully uses a class instead of a struct.