CHAPEL

Socket Library
Google Summer of Code 2021

Introduction

I am **Lakshya Singh**, a Computer Science and Engineering undergraduate student at the **Indian Institute of Technology (BHU)** pursuing a Bachelor of Technology in my second year. I was introduced to the world of programming and software development in my first year. Since then, I have been very enthusiastic about deep diving into various fields of Computer Science. The areas that capture my interests are Data Structures, Algorithms, Operating Systems, Natural Language Processing and Information & Security. For most of my programming journey, I have worked primarily with **Web-based technologies**, my niche and **C/C++ program**s, and I have recently been diving into **Deep Learning** as well.

Relevant Coursework for the Project

- ★ Operating Systems
- ★ Algorithms
- ★ Data Structures
- ★ Computer System and Architecture
- ★ Object-Oriented Programming
- ★ Client and Server Architecture

Why do I wish to participate in the Google Summer of Code?

Being a technology enthusiast even before I entered college, I always enjoyed the time I spent debugging my code and PC. During my first year, I was amazed at how genuinely vast the world of Computer Science is, and there is so much for me to learn, which I genuinely enjoy. My first interaction with open source was Linux and NPM; it was a whole new realisation. I was using a project someone else developed, and it's helping me scale up my projects so quickly. The idea of collaborating with so many people working together to build amazing things attracted me to open source.

So I started contributing to OSS in this year's Hacktoberfest. I contributed to many organisations. It was delightful to work with so many people getting reviews from them, improving my skills and knowledge base, interacting with them, which wouldn't have been possible without open source.

I have similar aspirations from Google Summer of Code. I want to work more in an open-source community where we have several people from different parts of the world working as a team and building projects collaboratively. And what better than Google Summer of Code, which will also allow me to hone my skills and acquire new ones under quality mentoring organisations and best mentors' guidance.

Why do I wish to work with the Chapel project in particular?

The Chapel Project is an excellent opportunity for me to deep dive into the distributed and parallel programming and better understand what happens under the hood of a language that has been a black box for me before I started contributing to chapel code base. I have always been keen to make my programs and applications more performant, fast and easy to understand, which are among the few sub-domains of chapel ideology and principles.

The project has great mentors who are always willing to provide the best possible aid and are very responsive, friendly and ready to share their knowledge. This has been a

fantastic experience for me. It allows me to interact with mentors and other contributors and gain more knowledge while staying connected with the community and contributing more to the project through discussion and code.

What do you hope to learn over the summer?

Over the summers, I hope to learn more about Socket programming and set up HTTP enabled services allowing me to know more about how we implement modules from scratch rather than working on higher-level APIs. I also intend to learn more about parallel and distributed programming and optimise it and effectively implement it for performance. The valuable experience of working with the Chapel community will allow me to learn more about how we maintain such a huge code base and efficient workflow strategies associated with it. I hope to learn more about software development overall under the guidance of knowledgeable and experienced mentors.

How well can you comprehend and understand English? How strong is your written English?

I am proficient in communicating with people in English. I can proficiently speak, read and write in English as well as comprehend it. I do many Parliamentary Debates and MUNs, which has improved my Language skills over time, even though Hindi is my first language.

Do you have any other commitments for the summer period? Do you have planned vacations?

I have my official college summer break from 12th May to 22nd July, and during the official GSOC Period, I have **no other commitments** for the summers, so I will be able to devote more than 25 hours/week as per GSoC 2021 Guidelines. I will also be accessible during weekends after my semester starts in August and will keep the community updated about my progress.

Contact

- **Name**: Lakshya Singh
- **Github Username**: king-11
- **Email**: lakshay.sing1108@gmail.com

- **Timezone:** IST (UTC +5:30)
- The time zone will remain the same during the summers.
- The time I will comfortable in working with :
    - UTC 0330 - 0630 (IST 0900 - 1200)
    - UTC 1330 - 1430 (IST 1900 - 2000)
    - UTC 1730 - 1930 (IST 2300 - 0100)
- I can start my day a couple of hours early, even around **IST 0600 (UTC 0130)** in the morning and can end it late until IST 0200 (UTC 2030) if it helps communicate with other developers.

Coding Experience

Describe your experience with Chapel, C, C++, and any other programming experience you wish to mention.

C and **C++** have been my primary language for most of the course-related work that I have done under the courses mentioned above in the Introduction Section. I also do competitive programming for which I primarily use C++ and always stay on the lookout for new features to improve the code readability and performance.

I am also familiar with **Python** Programming and Scripting. I have used Python for Machine Learning Model Development, setting up Backend Server using Django and DjangoREST and small scripting purposes.

I am also familiar with Web Development Programming Languages like **Javascript**, **Typescript**, **SASS, NodeJS,** which I have primarily used in developing several projects and have also worked with **SQL** and **Shell Programming**.

Chapel

I started coding in the chapel at the start of February. And followed the getting started pages :

- I started with Learning Chapel Page and went over to the youtube Talk by Brad Chamberlain to get insights into Chapel's principles and working.
- I went through the Learn X in Y minutes docs for Chapel and fiddled around with the code in it.
- I have gone through the Primers as my initial introduction to distributed and parallel programming into the chapel world.
- I made several pull requests to the Chapel Project and also found some issues while working on it.
- I worked with a few modules of Chapel like **Path, DateTime, Heap, IO and Sort.**

Describe any experience with compiler development, parallel computing, or any other knowledge you know will be useful for the task.

I have been learning more about Parallel Programming under the Operating Systems Course I have this semester, where I worked on implementing several **Parallel Programming** and **Deadlock Prevention** algorithms. I am well versed in Parallel Programming concepts like mutex locks, semaphores, deadlocks etc. Programs I have implemented in due course of time using C/C++ include :

- Producer-Consumer Problem using Mutex Locks, Semaphores and Threading.
- **Banker's Algorithm** using Threading
- Matrix Multiplication using **Threading**
- **Process communication** using Pipes
- Process Scheduling

I have gone through the **C Interoperability** and **IO Module,** which will be necessary for implementing socket library functions and have also worked on PR #17453, which required me to use and enforce custom C functions for use in the Chapel DateTime Standard Module.

I have implemented several HTTP Server primarily for use as REST and GraphQL APIs from the ground up using vanilla NodeJS and Python and worked with higher-level APIs of frameworks like Django and Express. While implementing these projects, I have gained insights into working with Web Services like web sockets and internet protocols necessary for the project.

I haven't developed any production-ready application using C, which can be insecure if designed from scratch. So I have only used C to experiment and learn network programming basics to understand lower-level APIs system calls better.

Projects involving the use of **REST API** and **Socket Programming** :

- **Peer IO Backend:** I developed this Project in a **24 hr Hackathon** which serves as a REST API and provides access to services like Authentication, Database Access, etc., in a **non-blocking** manner using **NodeJS, Express, MongoDB.**
- **Shopify:** A shopping website I developed with my teammate as a part of our course project. My role was to write new views for handling **HTTP requests**, **optimising** existing **queries** made to the database, and ensuring proper error handling. Technologies used were **Django, PostgreSQL, Heroku.**
- **Chat Server:** A client-server chat CLI application developed using **Python sockets** module
- **ZenLibrary:** A full stack web application developed solely my be for helping avid book readers. I worked both as Frontend and Backend Developer in the App utilised **GraphQL API** created using **Apollo** and **MongoDB,** whereas the frontend used **NextJS**.
- **Discord Bot:** Developed a serverless function that utilises Discord's relatively new Interaction Commands. Works on **webhooks** that are based on **HTTP Request** handlers in a stateless manner. The tech stack was purely **NodeJS,** while the server code had to be adjusted for **vercel** deployment.

Familiarity with tools

- I feel very comfortable with **Git,** which I have been using for more than a year now. I know how to use some of its essential features like rebasing, checkout, resets, reflog, etc., to maintain my repository correctly. I have even done a course on Advanced Git by Nina Zakharenko, which provided me with more profound insights.
- I use the **GCC** compiler system for compiling the C code that I write and am familiar with several of its flags.
- As part of our institute's CyberSec team, I have participated in several CTFs (Capture the Flag) competitions. I have used tools like **gdb, Valgrind, strace, ltrace**, etc., for binary exploitation, pwning and reversing tasks.
- I am not that much proficient with **make** except for the part I have used several times in other open-source projects.

What experience do you have as part of a development team?

I have been a part of various development teams during my journey as a software developer for over 1.5 years. I have primarily worked as a Web Developer but have also interacted with several other developers from various fields.

Technex Tech Team

I was part of the team which was handed the task of developing a website for Technex'21 within a month. Technex is the Technical Fest of IIT BHU, which organises a plethora of technical events in various fields and organises Think Talks with the industry's pioneers. As the backend was already developed, most of our task involved creating a unique front-end design and attaching the frontend to the backend while ensuring proper error handling. We worked in a team of 7, where we decided on design principles that are neat and scalable and meet organisers' needs.

Club of Programmers (COPS)

I am a COPS IIT BHU, a group of enthusiasts who share a common interest in Computer Science. Under various sub-division of the club, we work on several fields like Competitive Programming, Machine Learning, Software Development and Cyber Security. I have worked under several projects taken by the Club, including several projects for aiding the student community and fun side projects and competitive

projects for hackathons. All projects can be found at the Github Organization COPS IIT (BHU).

Hackathons and CTFs

I have participated in several hackathons and completed all of them. Usually, we try to have the maximum number of people as allowed by hackathon organisers. It enables us to learn from each other and build up to something pretty cool and helpful quickly. Hackathons have allowed me to work on projects tirelessly to finally achieve a working prototype in a stipulated time of 1-2 days.

As I am also a part of the CyberSec Group of IIT BHU, I have participated in several CTFs where we band together to hack our way through the challenges thrown at us. This has provided me insights into web security, and we as a team used to develop over partial work done by others and develop upon it.

What is the biggest project you have worked on as a software developer? What did you learn in that project? What was your role in that project over time?

The biggest project I have worked on is **Hackalog** as a student developer under the Club of Programmers. This project aimed to provide a platform for conducting hackathons and dev sprints for the student community. When it was at the release phase, I joined the team where work had to be done more rigorously and quickly and supposed to be bug-free.

This was the first time I was working with **React,** and I wasn't familiar with any of the concepts like JSX, hooks, CSS modules etc., that were used in it. As the project was in the release phase, I had to keep my learning and work going side by side. I went through the **docs** for **React**, **React Hooks and NextJS.** I tried to level up my knowledge to get insights into their work and hence can optimise them.

My responsibilities involved refactoring the code for performance, checking for optimisation opportunities in react hooks, memory leaks, adding accessibility support, optimising the SEO experience, and adding support for Markdown editor abilities. I updated the workflow to provide support for better continuous integration and added clean code practices. We also had code review conducted amongst the peers as well as had group discussions on the project. The repository can be found at this link.

Is any of the code you have written already open source? Can you point us to some code you have written?

I started my open source contribution in **Hacktoberfest** of 2020, where I contributed to several organisations daily. Apart from the ones I stated above, the major projects I have worked on can be found on my Github Profile. The organisations I have worked within past are

Organisation	Repository	Pull Requests Made
DX Heroes	DX Scanner	Link
Operation Code	Resources_API	Link
IIT BHU Insti App	Lite Hai Backend	Link
HTTP-APIs	• Hydrus • Hydra Python Core	Link 1 Link 2
Datenanfragen	Website	Link
-	Letra Extension	Link

Apart from this, I also try to contribute to open-source tools that I use by discussing or creating issues wherever necessary after triaging.

What have you already contributed to the Chapel project? Please list pull requests and issue numbers.

I have made several pull requests, created a few issues, and am currently triaging some issues, which I will create. I have also discussed a few performance issues in standard modules with core contributors, which I intend to take up soon.

Pull Requests

PR Number	Fixes	Description	Status
#17453	#16922	Extend C's strptime to support '%f' and change function type to ref in datetime.strptime().	Merged
#17443	#16552	Add datetime and date mixed operator	Merged
#17395	#16733	Add function to retrieve time since epoch	Merged
#17388	#16767	Add support for manipulation of parts of a path, i.e. basename, dirname and extension	Merged
#17293	#17178	Optimise the number of recursive calls made in Quick Sort using Tail Optimization	Under Review
#17559	#8758	Adding Long Filename support for chapel programs	Under Review

Issues

Issue Number	Description
#17439	date.today() doesn't support local time as of now, which can cause issue with mixed type operators like datetime.now() - date.today()
#17178	Quicksort makes several recursive calls that can be used, reducing the heap-allocated and time needed to make those calls.

Survey

Had you heard about Chapel before the Summer of Code? If so, where? If not, where would you advise us to advertise?

I heard about Chapel from one of my friends who learnt about Chapel through GSOC Archives. He seemed pretty excited about the parallelism in Chapel, which intrigued me to look into Chapel.

Chapel's significant usage seems to be for research purposes, so I suppose conduction talks for university professors and students can surely increase its uptake in the community. Also, with the addition of HTTP and Socket module, the opportunity for performant backend development in the chapel will open, which can be advertised on platforms like dev.to, daily.dev, etc.

What was the first question concerning Chapel that you could not find an answer to quickly?

The first question that I could not answer quickly was how to include custom C functions/files in the chapel's build process used in standard modules. I found the solution after discussing with Lydia and Lee on the Gitter channel and Github Issue thread.

What will keep you actively engaged with the Chapel community after this summer is over?

As long as I see issues bubbling up in the chapel, I will stay involved as the core team seems very active in developing the project and interacting with them will allow me to contribute and learn from them. I will also try to improve the HTTP Library even after the end of the GSOC period, as it will undoubtedly allow me to create even better HTTP services.

Are you applying to any other organisations for this year's Google Summer of Code? If so, what is the order of your preference in case you are accepted to multiple organisations?

No, I am not applying to any other organisations except chapel for this year's Google Summer of Code.

Prerequisites

What operating system(s) do you work with?

I primarily work with Manjaro (Arch Linux) for coding purposes, while I also have Windows 10 with Windows subsystem for Linux (Ubuntu) configured.

Are you able to install software on the computer you plan to use?

I have been using Manjaro for over a year now after doing a lot of Distros hopping. I am pretty comfortable with installing any software in it, courtesy of the Arch User repository. I was able to set up Chapel in my system successfully.

Will you have access to a computer with an internet connection for your development?

Yes, my PC has access to a stable high-speed internet connection for the development period.

Self Assessment

What does useful criticism look like from your point of view as a committing student?

I think criticism is an opportunity for us to look back on our work and improve our current practices and skills. As a committing student, one should accept constructive criticism and use it to improve themselves further.

What techniques do you use to give constructive advice? How do you best like to receive constructive feedback?

I try to give constructive advice by reviewing the progress made and pointing out the specific parts of the implementation where excellent implementation details were taken care of. I also try to add updates that can ensure better readability, scalability, and performance regarding why they are helpful and keep myself open to others' views.

What is your development style? Do you prefer to figure out/discuss changes before you start coding? Or do you choose to code a proof-of-concept to see how it turns out?

I prefer to start coding after discussing all the significant aspects of implementation details and edge cases of a problem encountered in the future. This can be seen in the contributions I have made till now, where after finalising the discussion, I create a PR ready for review with all the implementation done as per discussion. Sometimes I also take the other way around when I need to explain what I am aiming for as the debate needs a prototype to discuss further or clarify the community's issue.

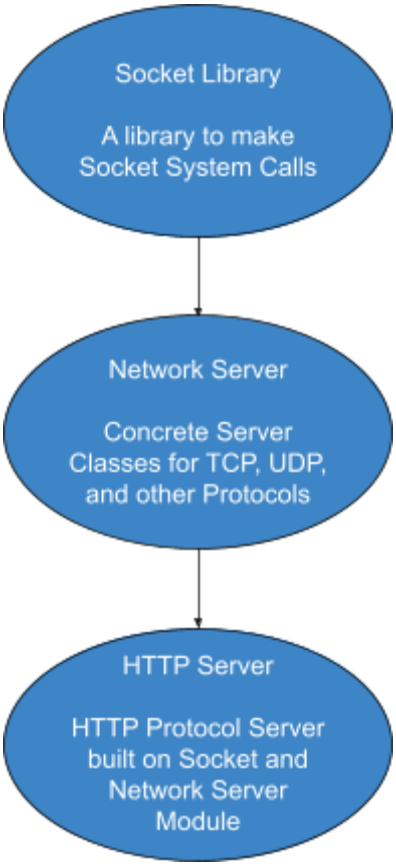
The Task

Describe the task you intend to work on. If it's one of the tasks from our ideas list, let us know which elements of it you want most to focus on, if you know?

The task is to develop a Socket Module for Chapel, which allows for increased use cases for Chapel. It will also involve creating an HTTP Server Module on top of the socket Module.

The key elements that will be of significant concern during the project include:-

1. Socket Module to integrate C Socket Functions
2. Writing good test cases and documentation
3. Ensuring Concurrency, Parallelism and Non-Blocking Nature of Procedures.
4. Building an HTTP Server on top of Socket Module



Description

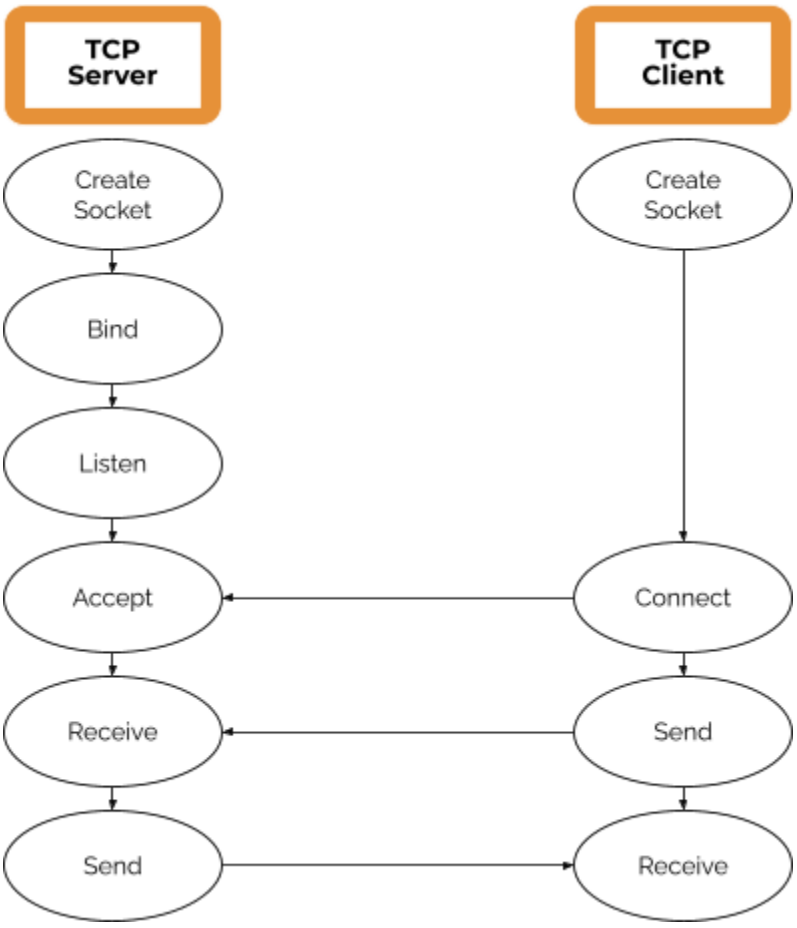
I went through C and Python Socket Library Documentation. I tried out several small programs whose primary objective was to get familiarised with performance and parallelism in sockets and several other parts related to the protocols used.

After which, I came up with the following flow for the project, which is somewhat inspired by how Python implements its socket and HTTP services. With consideration that will focus on developing network-related servers in a single module instead of two.

Sockets

Sockets are an abstraction over Pipes that allow communication between processes even if they aren't running on the same system, unlike Pipes. The most widely used Sockets are BSD sockets which are also known as Unix Sockets more commonly. Socket Module involves making system calls to the kernel through the program, which allows us to work with sockets at the network level.

There are two types of sockets Client and Server both require several steps in creating a connection as listed below :



- **Create Socket** - Returns a file descriptor for communication.
- **Bind** - Unique Address assigned to the Socket which consists of two parts IP Address and Port
- **Listen** - Server Sockets starts listening for requests from client sockets
- **Connect** - Client Socket makes connection requests to server socket by specifying its Unique Address.
- **Accept** - Server Socket accepts client socket connection request
- **Send, Receive** - Are used for communication between sockets by sending and receiving packets of data

Users can create a socket to follow various available protocols, each with different characteristics as per the user's needs. The widely used ones are:

- **Stream Sockets**: These sockets use Transmission Control Protocol (TCP) as a reliable means for transferring data and communication between processes. These involve establishing a connection between the processes hence are stateful sockets.
- **Datagram Sockets**: These are diametrical opposite of Stream Sockets. Each written data is sent in the form of packets to the specified address in an unreliable manner. There is no need to establish a connection for these sockets, and hence are stateless.

Note: I have written all the code snippets provided in the document below.

Pseudo Script to connect to Google's IP using Stream Socket

```
use Socket;

// Make a new instance of the socket record defined in the socket module provid
e domain and family in initialization
const sock = new socket(PF_INET,SOCK_STREAM);
// create record of server socket
var server: sockaddr_in;
// convert ip address from dot and number 32 bit ip address
server.s_addr = inet_addr("172.217.166.78");
// convert port to network byte order
server.port = htons(443);
// define server family as IP family
server.family = PF_INET;
try {
  sock.connect(server); // connected
} catch e {
  // handle error
}
// close the socket
sock.close()
```

Internals of Socket Module

The socket module will consist of a socket record with methods for **initialising** the socket and getting a file object from the file descriptor returned by calling the C

socket function; **bind**, **listen** and **accept** will work as stated above. In contrast, read and **write** will work using **Chapel IO**.

A pseudo script/structure for the socket module will look as follows :

```
module Socket {
  pragma "no doc"
  extern "struct tm" record sockaddr_in {
    // properties of C struct sockaddr
  }

  /* Protocol families. */
  enum ProtocolFamilies {
    PF_UNSPEC =    0,
    PF_LOCAL =   1,
    PF_INET = 2,
    PF_INET6 =  3
  }

  /* Protocol families. */
  enum SocketType {
    SOCK_STREAM =  1,
    SOCK_DGRAM =   2
  }

  /* record representing socket */
  record socket {
    pragma "no doc"
    var fileDescriptor, sock_fam, sock_type:int(32);
    // associated getter for other and above specified properties
  }

  proc socket.init(domain:ProtocolFamilies,type:SocketType) {
    // intialize values for family and type


    extern proc socket(domain:int,type:int,protocol:int): int;
    var valFD = socket(this.sock_fam,this.sock_type,0);
    if (valFD ≠ -1)
      this.fileDescriptor = valFD;

    return valFD;
  }
```

Adapting To Chapel

While the GNU C Socket Module provides the ability to interact with sockets, calls made by it are **blocking** and **single-threaded**. Some of the procedure calls that need to be customised for Chapel include :

1. Accept
2. Receive
3. Connect
4. Write

The Socket Module for Chapel needs to be made to ensure all procedures are non-blocking and can use **multiple threads** for task distribution. Design decision need to be made about how will parallelism and concurrency be secured with the sockets, some of the methods that I will explore:

- Using **Select's** multiplexing capabilities to work with making receive calls non-blocking and dealing with erroneous sockets. Select allows the program to sleep until one or more **file descriptors** are ready for IO.

```c
#define FD_MAX 56
/* Callback to make when a file descriptor is ready */
struct cb {
  void (*cb_fn) (void *); /* Function to call */
  void *cb_arg; /* Argument to pass function */
};
static struct cb rcb[FD_MAX], wcb[FD_MAX];
static fd_set rfds, wfds;

void cb_check(void) {
  fd_set trfds, twfds;
  int i, n;
  trfds = rfds;
  twfds = wfds;
  n = select(FD_MAX, &trfds, &twfds, NULL, NULL);
  if(n < 0){
    // throw error
  }
  for (i = 0; n && i < FD_MAX; i++){
    if(FD_ISSET(i,&trfds)){
      n--;
      if (FD_ISSET (i, &rfds))
        rcb[i].cb_fn (rcb[i].cb_arg);
    }
    if(FD_ISSET(i,&wfds)){
      wcb[i].cb_fn(wcb[i].cb_arg);
    }
  }
}
```

- We will look further into **ioctl** to mark sockets as non-blocking and deal with associated errors that the program will generate.
- Exploring the compatibility of **SOCK_NONBLOCK** and **accept4** with various systems

```c
int async_accept(int s){
  // accept new connections in non blocking mode by defaul
  int fd = accept4(s, NULL, 0, SOCK_NONBLOCK | SOCK_CLOEXEC);
  if(fd == -1){
    // handle or throw error
  }
  return fd;
}

```

- Using a custom C procedure to set the **O_NONBLOCK** bit of a file descriptor non-blocking with the **fcntl** system call. Module implementation needs to provide methods to handle errors like **EAGAIN, EINPROGRESS, EWOULDBLOCK, etc.**

```c
void make_async(int s) {
  int n;

  /* Make file descriptor nonblocking */
  if((n = fcntl(s,F_GETFL)) < 0 || fcntl(s,F_SETFL, n | O_NONBLOCK) < 0){
    // throw error
  }

  /* Enable keepalives to make sockets time out if servers go away. */
  n = 1;
  if (setsockopt (s, SOL_SOCKET, SO_KEEPALIVE, (void *) &n, sizeof (n)) < 0) {
    // throw error for keeplive
  }
}

```

Depending on the decision taken for making the socket, non-blocking will have to handle exceptions when **read** and **write** operation will take place using Chapel's IO Module.

Utilising Chapel IO

Chapel's IO module has support for working with file descriptors using **openfd** and **openfp.** Instead of working with C's read/recv and write functions, we can interpolate Chapel's IO module to work seamlessly with socket file descriptors providing first-hand support for parallelism. Chapel IO will also work in accepting sockets from **socket.accept,** which can then use regular IO Channels for reading and writing.

```
1  private use IO;
2
3  private proc openSocket(in s: int): file {
4    if (s == -1) then {
5      // throw error socket wasn't created
6    }
7
8    try {
9      const fdFile = openfd(s,IOHINT_PARALLEL);
10   } catch {
11     // handle error
12   }
13
14   return file;
15 }
16
```

The file object returned will be stored inside the class, encapsulating it from users. It will provide access to **read** and **write** operation on socket file descriptors after the

connection is established for **TCP** Sockets, whereas for **UDP** functions similar to C's **sendTo** and **readFrom** need to be implemented using Chapel's IO.

Also, currently, the Chapel IO makes blocking reads and writes. The project duration will involve making it work in a non-blocking manner. This will require setting up **communication** between the running process and read/write a procedure that will convey that the read/write operation requested by the file object is **completed** and any further task related to it can proceed now. I will explore possible methods in the run of the project to make the procedures non-blocking.

Chapel Sys Module Imports

The need to implement many of the **enums** and **structures** will not be required as we can import them from the Sys package directly. The functions over there are a direct port of C functions. Therefore, they will be needed to be changed for non-blocking versions while some procedures can be directly used, like :

- sys_close
- sys_fcntl
- sys_bind
- sys_select

```
1
2  private use Sys;
3
4  proc socket.close() {
5    var value = sys_close(this.fileDescriptor);
6    // handle erroneous values
7  }
8
9  proc socket.bind(ref addr: sys_sockaddr_t) {
10   var value = sys_bind(this.fileDescriptor,addr);
11   // handle erroneous values
12 }
13
```

Utility Functions

The module will include socket creation and communication functions and have utility structures abstracted for chapel's need to make it easy for users to create Web Services while working with the socket module. Some of the procedures are :

- **gethostname** method to retrieve a list of IP addresses from the domain name.
- Procedure for **converting** ip_address in string form and port in integer form to **standardised** byte forms
- **Inverse** Procedures for converting from byte forms of port and IP Address to readable forms
- **getprotocol** for converting strings to protocol enum values

We can build the module by taking Design Decisions from Python's Socket Module while extending C Socket Module's capabilities as per design decisions. Unlike C's functional format of Socket Module, we would like to make the procedures in an Object-Oriented architecture instead, allowing for encapsulation of several features and hiding implementation details from users, only making higher-level APIs available to them.

Example Utility Functions

```
use Socket;

var hostname = "www.google.com";
// hostnet record to store value from gethostname
var he: hostnet;
// searches for specified hostname and returns information about it
he = gethostname(hostname);
var ip:string;
// a list of addresses obtained from name servers
for x in he.h_addr_list do {
  if !is_c_nil(x) {
    // convert 32 bit IP to ASCII form
    ip = inet_ntoa(x);
    break;
  }
}

```

Network Server

The Network Server will be implemented as a part of the HTTP/server module. The classes listed are intended to simplify the task of writing servers based on commonly used Protocols. The basic idea for them is to inherit from a parent **BaseServer** class and build upon them as per the need for protocol.

We will create a **BaseRequestHandler** class which the user will **pass** on to the **server** class. BaseRequestHandler will contain a **handle()** method that the user will be required to override, or else a primary handler will be implemented.

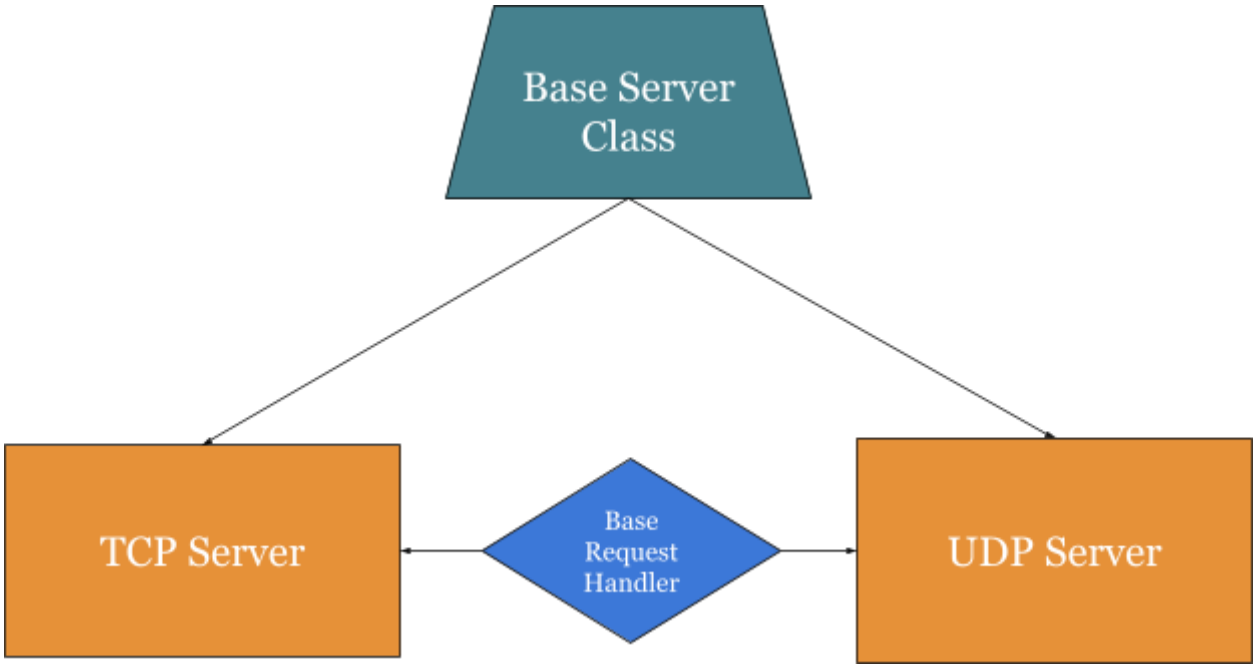
```
class BaseRequestHandler {
  var request;
  var server;
  var client_address;

  proc handle() {
    HaltWrappers.pureVirtualMethodHalt();
    // user provides the method
  }
}

```

We will create server classes (listed below) which will be instantiated by passing the *BaseRequestHandler* class instance and server address, i.e. IP and Port :

- **TCP Server**: This will use the TCP Protocol for providing continuous connection and data streaming capabilities between client and server. The user will be required to give the handler and address needed to connect and parameters for binding or to communicate so that client and server can be distinguished.
- **UDP Server**: This uses datagrams, which are discrete packets of information that may arrive out of order or be lost while in transit. The parameters are the same as for TCP Server.

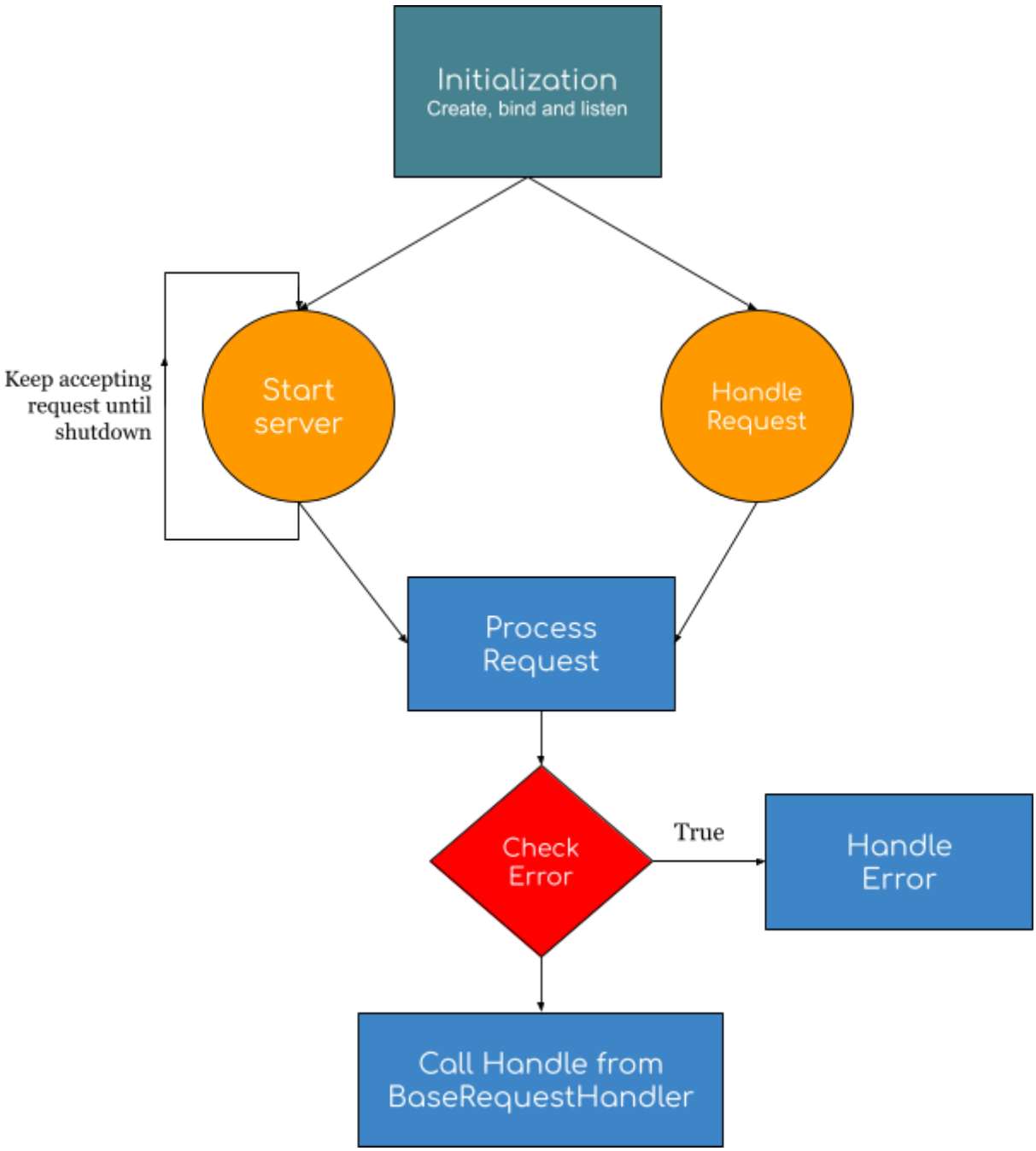


Base Server Class

The base server class will have the following methods attached to it, which will create and start the server :

- **init** procedure will handle the creation, binding of the socket and begin listening based on the provided arguments for IP and Port.
- **handleRequest** method will deal with a single request by opening the client socket for reading and writing. It will check whether the client is readable or the timeout hasn't occurred, after which it calls **processRequest** while handling any errors
- **startServer** handles any new request using **processRequest**. It will observe for shutdown using a shared atomic variable till then it will keep on listening for new request
- **processRequest** method will create a new task for accepted connection by creating a new object BaseRequestHandler provided during instantiation and add the task into a list of tasks

- **handleError** will provide basic error handling and can be overridden by the user
- **stopServer** will set the atomic Variable and then starts reaping/joining all the ongoing tasks.



```
class BaseServer {
  proc init(ip:string,port:int,soc_type:int,requestHandler:BaseRequestHandler) {
    /* initialize values and socket bind it and start listening */
  }

  proc handleRequest() {
    /* accept a new connection if it's readable and call processRequest */
  }

  proc startServer() {
    /* until a stop request is called keep on checking for new readable connections and call processRequest on them. */
  }

  proc processRequest() {
    /* accept a single request incoming request */
  }

  proc handleError {
    /* basic error handling */
  }

  proc stopServer() {
    /* clean up tasks and close the server */
  }
}

```

HTTP Server

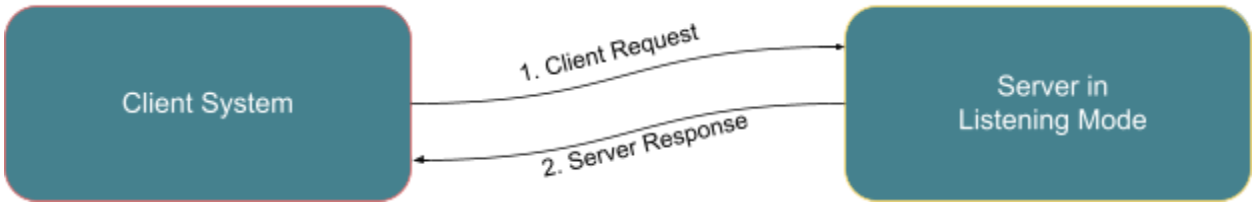
Hypertext Transfer Protocol (HTTP) uses TCP as a network transport layer. An HTTP Communication differs from regular TCP communication as the data being transferred between clients and server includes several details about itself, its host, the protocol used etc. Parts of an HTTP Response are :

- Headers
- Body
- Response Status Code
- Method

```
// no protocol bound response
var response = "Hello World!";

// http controlled response
var httpResponse = "HTTP/1.1 200 OK\nServer: Vercel\nDate: Tue, 06 Apr
 2021 20:36:34 GMT\nContent-Length:12\nContent-Type: text/plain;charset
=UTF-8\n\nHello World!";
```

The HTTP Server will be required to parse this information associated with the request into different records. Based on the type of request, the user will be required to provide a Base Handler to deal with the request.



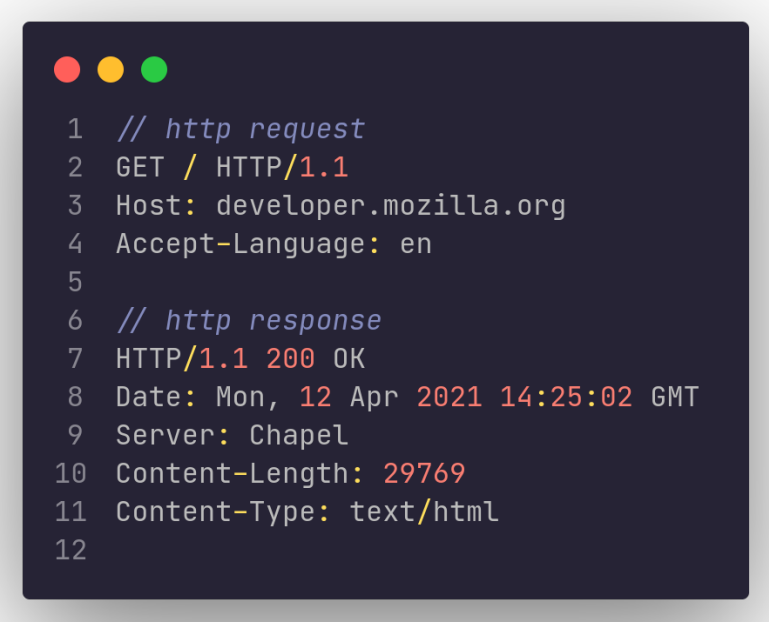
HTTP Support several methods, some of the widely used ones are as below :

1. **GET**: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data
2. **POST** - The POST method is used to submit an entity to the selected resource.
3. **DELETE** - The DELETE method deletes the specified resource.
4. **PUT** - The PUT method replaces all current representations of the target resource with the request payload.

Parsing HTTP Request

Parsing the HTTP Request and verifying whether the request is valid is an additional task for the HTTP server built over the TCP server.

As stated above, we need to parse components for requests like headers, version of HTTP, resource URL, method and body.



```
// http request
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: en

// http response
HTTP/1.1 200 OK
Date: Mon, 12 Apr 2021 14:25:02 GMT
Server: Chapel
Content-Length: 29769
Content-Type: text/html

```

The HTTP class's **parseRequest** function will utilise regex expressions to extract out the components and verify the header. The implementation of **parseRequest** and associated record will look as follows :

```
1
2  record Header {
3    var name, value:string;
4  }
5
6  record Request {
7    Methods method;
8    var url ,version:string;
9    var headers:list(Header,parSafe=true);
10   // support for POST, PUT, etc methods
11   var body: string;
12 }
13
14 proc parseRequest(const rawRequest:string) : Request throws {
15   var parseRequest:Request;
16   const splitPattern:regex = compile("[\\s]{1,2}");
17   const splitRequest = rawRequest.split(splitPattern,3);
18
19   if(splitRequest.size() ≤ 3)
20     // throw error as not valid http request
21
22   const method = splitRequest[0], url = splitRequest[1], version = splitRequest[2];
23
24   select ( method ) {
25     when "GET" do {
26       parseRequest.method = Methods.GET;
27     }
28     /* similarly parse other methods
29       if none of the methods is found
30       then request is invalid
31     */
32   }
33
34   // set the extracted URL
35   parseRequest.URL = url;
36
37   if(!version.startsWith("HTTP"))
38   // throw error if HTTP version not found
39
40   splitPattern = compile("\r?\n");
41   var remainingRequest = splitRequest[3].split(splitPattern)
42   for x in remainingRequest {
43     // empty line body separator
44     if (x == "") then break;
45
46     var pair = x.split(":",1);
47     var name = pair[0], value = pair[1];
48     parseRequest.headers.append(new Header(name,value));
49   }
50
51   // parse body from array remainingRequest
52 }
53
```

We can build the HTTP Server upon the **TCP Server Class** of the Network Servers designed above for Chapel.

The examples built in the community will be taken as a starting point as some of the projects have been worked upon a lot and can help **speed** up the **process**. One of them is a C multithreaded server pico, and I also found that a complete HTTP module is implemented in chapel-http.

The HTTP server's demonstration will include the initial task of handling GET Method requests while developing upon the GET Method handler's principles and extending it to other methods in later phases of the module.

Persistent Connections (Optional)

HTTP/1.1 includes a **keep-alive** mechanism that allows for multiple requests through the same connection. Otherwise, the connection is closed after a single request-response cycle between client and server.

It's up to the server whether it accepts the keep-alive connection or not. We can choose to implement the functionality for persistent connections at the end of the project given time availability so that the server can support HTTP/1.1 completely.

To provide persistent connections, we will be required to keep the connections open until a request without the keep-alive header is observed. For the rest operations, the HTTP server can close the connection as soon as we have handled the request.

The request parsing will involve checking for properties like **timeout** and **max** request-response cycle before closing the connection. We can have defaults for both values to deal with idle connections and ensure that CPU time is not wasted checking those connections.

Base HTTP Module Implementation

```
module http {
  // inherit classes from network server
  private use NetworkServer;

  /* HTTP Methods. */
  enum Methods {
    GET =    0,
    POST =   1,
    PUT = 2,
    DELETE =  3
  }

  class HTTPServer:TCPServer {
    var path, method:Methods;
    var readFile:file, writeFile:file;
    var headers;
    var body;

    proc init() {
      // initialize values
    }

    proc parseRequet(req) {
      /* Logic to parse headers,
      body, method from the incoming request.
      */
      return parsedRequest;
    }

    proc handle(parsedRequest) {
      /* override method from BaseServer */
    }

    proc sendRespons(code:HTTPCodes,message=""){
      // sending response back to the user
    }
  }
```

Why is this task exciting to you? Why did you choose this particular task? What do you hope to learn by working on it?

I have been working in the field of Web Development for over a year now. I have also worked on web security which is of significant concern in today's world. This particular task excites me because it will allow me to work with the **lower-level APIs** of Web Technology, which I have always wanted, instead of just working with Higher Level Abstractions. This task will help me get insights to answer better the question **How the Web works**?

I have chosen this particular project because I have always looked around for a while working with Web Technologies is **Better Performance.** I have moved from Django to NodeJS from ReactJS to VueJS in search of performance, which Chapel provides at first hand and is one of the **core fundamentals** of the language. This seems to be a longstanding **wishlist** from Chapel Community and will be a leap for Chapel's usage in the community.

The task will provide me with a deeper understanding of the working of **Web Services** and associated **Protocols**. It will also allow me to understand better the development and architecture of other famous Backend Frameworks like Django, Express, etc. By the end of this project, I would have gained a better knowledge of **socket programming** and **HTTP services** and details about their internal moving parts. As a student, this project will allow me to **interact** with knowledgeable and **experienced mentors**. I will get to know about developing libraries from the ground up and working on large **open-source** projects.

Provide a rough estimated timeline for your work on the task. This timeline should take into account any non-coding time, such as exams, GSoC midterms, and vacation. Describe milestones you expect to achieve as you work towards the task.

Pre GSOC Period	
	April 14, 2021 - May 17, 2021
April 14, 2021 - May 17, 2021	➢ Work on issues : #16394, #8758, #17439, #7662 and merge pending PRs ➢ Triage possibility of optimisation in Heap Module for merging list ➢ Work on timezone class implementation for DateTime module ➢ Learn more about non-blocking IO and C networking. ➢ Stay connected with the community and learn chapel
Community Bonding Period	
	May 17, 2021 - June 7, 2021
May 17, 2021 - May 30, 2021	➢ Learn more about Data Parallelism and Task Parallelism ➢ Discuss Approaches to integrate parallelism in Socket and HTTP Module ➢ Read through IO, C Interoperability, Sys Module
May 31, 2021 - June 6, 2021	➢ Implement mock structure and pseudo-code for Library ➢ Discuss design decision ➢ Priorities Task at hand in sub-components

Coding Period	
	June 7, 2021 - August 16, 2021
June 7, 2021 - June 21, 2021	➢ Finalise the design decisions for the socket module ➢ Start integrating C functions into socket module to ensure non-blocking nature ➢ Develop necessary sub-records for essential socket interaction ➢ Add test and example for essential socket functions
June 22, 2021 - June 27, 2021	➢ Update procedures for parallelism ➢ Bug fixes and refactors module architecture for optimisation ➢ Add more tests as needed
June 28, 2021 - July 7, 2021	➢ Add utility functions for Socket Module ➢ Add tests for updated server functions
July 8, 2021 - July 11, 2021	➢ Discuss design for the network server and its final implementation ➢ Add classes for Network Servers
Mid Term Evaluations	
	July 12 - 16, 2021
July 19, 2021 - July 22, 2021	➢ Add new test cases for Network Server Classes. ➢ Bug fixes and refactoring code for the network server module ➢ Design Decision for HTTP module prepare pseudo-codes
July 22, 2021 - August 5, 2021	➢ Start implementing the HTTP module ➢ implement abstract and base classes for HTTP module ➢ Add new test and examples for the HTTP module
August 6, 2021 - August 11, 2021	➢ Final code refactoring

	➢ Update docs and examples ➢ Bug fixes and check for memory leaks ➢ Update procedures for better performance and compatibility
August 12, 2021 - August 15, 2021	➢ Discuss with the community about next steps ➢ Get ideas about current implementation and chance for improvements ➢ Decide next milestones for module
Students Submit Code and Final Evaluations	
August 16 - 23, 2021	
Post GSOC Period	
Based on community and core developers' recommendation, work on new HTTP and network server modules. Stay connected with the community and keep on adding and improving the project.	

References

- https://summerofcode.withgoogle.com/how-it-works/
- https://www.man7.org/linux/man-pages/man7/socket.7.html
- https://linux.die.net/man/3/send
- https://docs.python.org/3/howto/sockets.html
- https://dzone.com/articles/parallel-tcpip-socket-server-with-multi-threading
- https://www.educative.io/edpresso/how-to-implement-tcp-sockets-in-c
- https://www.gnu.org/software/libc/manual/html_node/Sockets.html
- http://www.dcs.gla.ac.uk/~johnson/teaching/CS-1Q/slides/lecture4/net.pdf
- https://chapel-lang.org/gsoc/ideas.html#sockets-library
- https://www.binarytides.com/socket-programming-c-linux-tutorial/
- https://realpython.com/python-sockets/

- https://docs.python.org/3/library/http.server.html
- https://docs.python.org/3/library/socketserver.html
- https://gist.github.com/laobubu/d6d0e9beb934b60b2e552c2d03e1409e
- https://github.com/marcoscleison/chapel-http