The 4 command line inputs to a.out is <MinSpareServer> <MaxSpareServer> <MaxClient> <MaxRequestPerClient>.
MinSpareServer - The MinSpareServers directive sets the desired minimum number of idle child server processes. An idle process is one which is not handling a request. If there are fewer than MinSpareServers idle, then the parent process will spawn one child, wait a second, then spawn two, wait a second, then spawn four, and it will continue exponentially until it is spawning 32 children per second. After that, it becomes constant at 32 children per second. It will stop whenever it satisfies the MinSpareServers.
MaxSpareServer - The MaxSpareServers directive sets the desired maximum number of idle child server processes. An idle process is one which is not handling a request. If there are more than MaxSpareServers idle, then the parent process will kill off the excess processes 1 at a time, the processes which are not serving any client i.e. in info table, child.status!=1
MaxClient - The MaxClients directive sets the limit on the number of simultaneous requests that will be served. Any connection attempts over the MaxClients limit will be dropped. Once a child process is freed at the end of a different request, the connection will then be serviced.
MaxRequestPerChild - The MaxRequestsPerChild directive sets the limit on the number of requests that an individual child server process will handle. After MaxRequestsPerChild requests, the child process will die. If MaxRequestsPerChild is 0, then the process will never expire.

Initially, the parent process forks 2 child server processes created on startup. As the number of processes is dynamically controlled depending on the load i.e. grows and shrinks based on cutoff boundaries Min/MaxSpareServers.

An array is created to store children information. The struct of record is pid_t pid, int status, int requestsHandled - where pid is process_id of child, status is whether it is handling a client currently or not, requestsHandled is number of request/clients handling.

The communication between the parent and child is one-way communication where the child communicates that it has accepted the tcp connection and whether it is free or busy in handling a client. This process of accepting connections and handling clients is done in childFunction(). The communication between parent and child is done through Socketpairs.

Child waits over the listening socket. Whenever it accepts a connection, it prints its pid, client's ip and port. Child receives the HTTP request and sends a dummy reply - "HTTP/1.1 200 OK\n"

Child sends msg over socketpair to parent using the following structure -

```
{
  pid_t pid;    ---- pid of client
  int status;   -----status busy(1) or free(0)
```

```
  char clientIP[64];  --- clientIP address
  unsigned int port_no;  --- port number of client
}
```

The parent handles the part of updating the child information in the info table that it maintains. It waits for the message from child and then updates -
1 - if clients handles children>MaxClientsPerChild, it recycles the child i.e. delete that child pid and fork a new child with a new pid
2 - checks whether number of children servers are in boundaries set by MaxSpareServers, MinSpareServers and acts accordingly. For this, it maintains a spareChildren variable which is Total children - current requests.
3 - after each message, it updates the child info in table, currentRequests variable and spareChildren variable depending on whether the child is free or busy