

```
1 CC = gcc
2 LDFLAGS = -lm -lnsl
3 CFLAGS = -g
4 TARGET = media_transfer parser server client
5
6 default: $(TARGET)
7
8 server: server.o
9     gcc $(CFLAGS) -o $@ $? media_transfer.o parser.o $(LDFLAGS) -lpthread
10
11 client: client.o
12     gcc $(CFLAGS) -o $@ $? media_transfer.o parser.o $(LDFLAGS)
13
14 media_transfer: media_transfer.o
15     gcc $(CFLAGS) -c media_transfer.c
16
17 parser: parser.o
18     gcc $(CFLAGS) -c parser.c
19
20 clean:
21     -rm -f *.o *~
22
23 cleanall: clean
24     -rm -f $(TARGET)
```

```
1  get song.mp3
2  list
3  get queue.h
4  list
5  exit
```

```
1 # mserver configuration file
2 # remove the pond sign to activate a configuration
3 PortNum: 1334
4 # Block: 2048
5 Threads: 4
6 Buffers: 10
7 Sched: SJF
8 # Directory: /media/
9
```

```

1  /* A simple echo server using TCP */
2  #include <arpa/inet.h>
3  #include <dirent.h>
4  #include <errno.h>
5  #include <fcntl.h>
6  #include <netdb.h>
7  #include <netinet/in.h>
8  #include <pthread.h>
9  #include <stdio.h>
10 #include <string.h>
11 #include <strings.h>
12 #include <stdlib.h>
13 #include <sys/socket.h>
14 #include <sys/types.h>
15 #include <sys/stat.h>
16 #include <poll.h>
17 #include <unistd.h>
18 #include <time.h>
19
20 #include "parser.h"
21 #include "media_transfer.h"
22 #include "queue.h"
23 #include "linked_list.h"
24
25 #define SERVER_TCP_PORT    3000    /* well-known port */
26 #define HEADERLEN          256    /* header packet length */
27 #define CONFIG_BUFFER      256    /* length of buffer for config line */
28
29 typedef enum sched_type {
30     FIFO,
31     RANDOM,
32     SJF
33 } sched_type_e;
34
35 typedef struct hanlder_arg {
36     int port;                /* port number server is running on */
37     int client_socket;       /* port number of client to deal with */
38 } handler_arg_t;
39
40 typedef struct {
41     int sd;                  /* server socket descriptor */
42     int port;                /* port number server will listen on */
43     int num_threads;         /* no.of threads - can be modified using CL Arg #3 */
44     int max_requests;        /* max no.of client requests server can have at any
45     time - can be modified using CL Arg number #4 */
46     char * directory;        /* place to look for media files */
47     pthread_t *handlers;     /* array of threads server can handle client
48     requests */
49     pthread_mutex_t lock;    /* mutex lock to do some thread safe
50     functionanlities */
51     pthread_cond_t cond;     /* condition variable */
52     Queue *job_queue;        /* process client request queue */
53     list *job_list;
54     sched_type_e scheduling_type; /* FIFO or RANDOM processing */
55 } server_config_t;
56
57 server_config_t config; // lets make config global.
58
59 /*
60  * @param config: struct to store config value from rc script
61  * @param configrc: file to read config information from
62  * @returns: success - 1 or failure - 0
63  * reads config file in to config struct
64  */
65
66 int parse_configuration(server_config_t *config, char *configrc);
67
68 /*
69  * @param config: server configurtion struct

```

```

67     * @returns number of chars printed
68     * prints server configuration summary
69     */
70 int print_configuration(server_config_t *config);
71
72 /*
73  * @param filepath - name of the file for which extension is needed
74  * @returns
75  *     point to first char in extension
76  */
77 const char *get_file_ext(const char *filename);
78
79 /*
80  * @param config: server config struct
81  * initializes threads, and locks for config struct
82  */
83 int initialize_thread_pool(server_config_t *config);
84
85 /*
86  * @param arg - handler args passing
87  * @returns
88  *     1 if client wants to disconnect
89  *     0 if client wants to continue
90  * Fulfills client requests
91  */
92
93 void handle_request(void*arg);
94
95 void get_sjf (void *arg);
96 void handle_sjf (void *client_request);
97
98 /*
99  * @param arg - server config arg will be passed
100  * takes a job from job queue.
101  */
102 void *watch_requests(void *arg);
103
104 int main(int argc, char * argv[]) {
105     char * config_file = NULL;
106
107     switch(argc) {
108     case 1:
109         config_file = "mserver.config";
110         break;
111     case 3:
112         if (strcmp(argv[1], "-c") == 0) config_file = argv[2];
113         break;
114     }
115
116     /* seed for random generator */
117     srand(time(0));
118
119     /* init server configuration */
120     char pwd[BUFLEN];
121     getcwd(pwd, BUFLEN);
122
123     /* fill in default config */
124     config.port = SERVER_TCP_PORT;
125     config.directory = pwd;
126     config.num_threads = 4;
127     config.max_requests = 10;
128     config.scheduling_type = RANDOM;
129
130     /* override default config if file provided */
131     if (argc == 3) {
132         switch(parse_configuration(&config, config_file)) {
133         case -1:
134             fprintf(stderr, "Unable to open configuration file!\n");

```

```

136         break;
137     case -2:
138         fprintf(stderr, "Configuration error!\n");
139         break;
140     }
141 } else {
142     parse_configuration(&config, config_file);
143 }
144 config.handlers = (pthread_t*)malloc(sizeof(pthread_t)*(config.num_threads));
145
146 if (config.scheduling_type == SJF) {
147     config.job_list = create_list();
148     config.job_queue = NULL;
149 } else {
150     config.job_queue = createQueue(config.max_requests);
151     config.job_list = NULL;
152 }
153
154 /* switch current working dir to media dir */
155 int ret = chdir(config.directory);
156 if (ret != 0) {
157     printf("cannot change to dir %s.\n", config.directory);
158     exit(1);
159 }
160
161 struct sockaddr_in server;
162
163 /* Create a stream socket */
164 if ((config.sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
165     fprintf(stderr, "Can't create a socket\n");
166     exit(1);
167 }
168
169 /* Bind an address to the socket */
170 bzero((char *)&server, sizeof(struct sockaddr_in));
171 server.sin_family = AF_INET;
172 server.sin_port = htons(config.port);
173 server.sin_addr.s_addr = htonl(INADDR_ANY);
174 if (bind(config.sd, (struct sockaddr *)&server, sizeof(server)) == -1) {
175     fprintf(stderr, "Can't bind name to socket\n");
176     exit(1);
177 }
178 config.scheduling_type = SJF;
179 /* initialize threads and mutex locks */
180 initialize_thread_pool(&config);
181
182 /* print sever configuration */
183 print_configuration(&config);
184
185 /* queue up to config.max_requests connect requests */
186 listen(config.sd, config.max_requests);
187
188 /* loop forever and add requests to queue */
189 while(1) {
190
191     /* get a new connection req on server socket*/
192     int new_client_sd = accept(config.sd, NULL, NULL);
193
194     /* if found a request, enqueue it for processing */
195     if (new_client_sd > 0) {
196         char enqueue_time[TIME_BUFFER_LEN];
197         get_time_spec_to_string(enqueue_time, TIME_BUFFER_LEN);
198         printf("\n%s: Main: Accepting New Connection: %d\n", enqueue_time,
199             new_client_sd);
200
201         handler_arg_t *arg = (handler_arg_t*)malloc(sizeof(handler_arg_t));
202         arg->port = config.port;
203         arg->client_socket = new_client_sd;

```

```

204     printf("%s: Main: Adding New Client to the Job queue...\n", enqueue_time);
205     /* Locks the queue to add job */
206     pthread_mutex_lock(&(config.lock));
207
208     /* add connectiong to queue */
209
210     if (config.scheduling_type == SJF) {
211         get_sjf(arg);
212     } else
213         enqueue(config.job_queue, (void*) arg);
214
215     /* give up the lock on the queue */
216     pthread_mutex_unlock(&(config.lock));
217
218     get_time_spec_to_string(enqueue_time, TIME_BUFFER_LEN);
219     printf("%s: Main: Added New Client to the Job queue\n", enqueue_time);
220
221     }
222 }
223 }
224
225 int parse_configuration(server_config_t *config, char *configrc) {
226     if (configrc == NULL) return -1;
227     if (config == NULL) config = malloc(sizeof(server_config_t));
228
229     FILE * c_file = fopen(configrc, "r");
230     if (c_file == NULL) return -1;
231
232     char buf[CONFIG_BUFFER];
233
234     while (fgets(buf, CONFIG_BUFFER, c_file) != NULL) {
235         if (strstr(buf, "#")) *(strstr(buf, "#")) = '\0';
236         if (strstr(buf, "\n")) *(strstr(buf, "\n")) = '\0';
237         if (buf[0] == '\0') continue;
238         if (strstr(buf, ":") == NULL) return -2;
239
240         char *split = strstr(buf, ": ");
241         *split = '\0';
242
243         char *key = buf;
244         char *value = split + 2;
245
246         // Fast and loose config parsing, only checking to see if config
247         // line contains the key value, thus, if you have something like:
248         // PortNumThreads: 5, it will match to PortNum and nothing else.
249         if (strstr(key, "PortNum")) config->port = atoi(value);
250         else if (strstr(key, "Threads")) config->num_threads = atoi(value);
251         else if (strstr(key, "Sched")) {
252             if (strcmp(value, "FIFO") == 0) config->scheduling_type = FIFO;
253             else if (strcmp(value, "Random") == 0) config->scheduling_type = RANDOM;
254             else if (strcmp(value, "SJF") == 0) config->scheduling_type = SJF;
255         } else if (strstr(key, "Directory")) {
256             config->directory = malloc(strlen(value));
257             strcpy(config->directory, value);
258         }
259     }
260 }
261
262 int print_configuration(server_config_t *config) {
263     printf("*****Server configuration*****\n");
264     printf("Port Number: %d\n", config->port);
265     printf("Num Threads: %d\n", config->num_threads);
266     printf("Max Reqs: %d\n", config->max_requests);
267     printf("Media Path: %s \n", config->directory);
268     printf("Sched type: %d\n", config->scheduling_type);
269     printf("*****\n");
270     return 0;
271 }
272

```

```

273 const char *get_file_ext(const char *filename) {
274     const char *dot_loc = strrchr(filename, '.');
275     if(!dot_loc || dot_loc == filename) {
276         return "Unknown";
277     }
278     return dot_loc + 1;
279 }
280
281 int initialize_thread_pool(server_config_t *config) {
282     if (pthread_mutex_init(&(config->lock), NULL) != 0) {
283         printf("\n mutex init has failed\n");
284         return -1;
285     }
286     int i;
287     for (i = 0; i < config->num_threads; ++i) {
288         if(pthread_create(&(config->handlers[i]), NULL, watch_requests, (void*)config)
289             != 0) {
290             printf("Failed to create a thread");
291             exit(1);
292         }
293     }
294     return 0;
295 }
296
297 void *watch_requests(void *arg) {
298     server_config_t *config = (server_config_t*)arg;
299
300     void *job = NULL;
301
302     while(1) {
303
304         pthread_mutex_lock(&(config->lock));
305
306         if (config->scheduling_type == SJF) {
307             job = get_job(config->job_list);
308         }
309         else if(!isEmpty(config->job_queue)) {
310             if(config->scheduling_type == FIFO) {
311                 job = dequeue(config->job_queue);
312             }
313             else if (config->scheduling_type == RANDOM) {
314                 job = random_dequeue(config->job_queue);
315             }
316         }
317
318         pthread_mutex_unlock(&(config->lock));
319
320         if(job) {
321             char time_processing_start[TIME_BUFFER_LEN];
322             get_time_spec_to_string(time_processing_start, TIME_BUFFER_LEN);
323             printf("%s: Watch Request: Thead %lu: Handling client %d\n",
324                 time_processing_start, pthread_self(), ((handler_arg_t*)job)->client_socket);
325
326             if (config->scheduling_type == SJF)
327                 handle_sjf(job);
328             else
329                 handle_request(job);
330
331             job = NULL;
332         }
333     }
334 }
335
336 void get_sjf (void *arg)
337 {
338     /* Some vairable declaration */
339     char time_buf[TIME_BUFFER_LEN];
340     handler_arg_t* info = ((handler_arg_t*)arg);

```



```

340
341     /* Print out client information */
342     struct sockaddr_in client_socket_addr;
343     socklen_t len;
344     len = sizeof(client_socket_addr);
345     char client_ip[32];
346     unsigned int ephemeral_port;
347
348     bzero(&client_socket_addr, len);
349
350     if (getsockname(info->client_socket, (struct sockaddr *)&client_socket_addr, &len)
351         == 0) {
352         /* get ip and the temp port*/
353         inet_ntop(AF_INET, &client_socket_addr.sin_addr, client_ip, sizeof(client_ip));
354         ephemeral_port = ntohs(client_socket_addr.sin_port);
355
356         /* print contents of ss*/
357         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
358         printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d\n", time_buf,
359             client_ip, ephemeral_port);
360         fflush(stdout);
361     }
362
363     char buf[BUFLEN] = {0};
364     char *bp = buf;
365     int bytes_to_read = BUFLen;
366     int n = 0;
367     while ((n = read(info->client_socket, bp, bytes_to_read)) > 0) {
368         bp += n;
369         bytes_to_read -= n;
370     }
371
372     if (bp <= 0) {
373         // client probably disconnected
374         close(info->client_socket);
375     }
376
377     get_time_spec_to_string(time_buf, BUFLen);
378     printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d : Command Received
379         string: %s", time_buf, client_ip, ephemeral_port, buf);
380
381     /* put a null character at the end */
382     int size = strlen(buf);
383     buf[strcspn(buf, "\n")] = 0;
384
385     char *job = malloc(strlen(buf));
386     strcpy(job, buf);
387
388     switch(get_command_from_request(buf)) {
389     case LIST:
390         add_job(config.job_list, job, 1, arg);
391         break;
392     case GET: {
393         FILE *fp = fopen(&(buf[4]), "rb");
394
395         if (fp == NULL) {
396             break;
397         }
398
399         fseek(fp, 0L, SEEK_END);
400         size_t len = ftell(fp);
401         fseek(fp, 0L, SEEK_SET);
402         fclose(fp);
403
404         add_job(config.job_list, job, len, arg);
405         break;
406     }
407     default:
408         add_job(config.job_list, job, 0, arg);
409         break;

```

```

406     }
407 }
408
409 void handle_sjf (void *client_request)
410 {
411     char time_buf[TIME_BUFFER_LEN];
412
413     struct node *req = (struct node *)client_request;
414     handler_arg_t* info = ((handler_arg_t*)req->owner);
415
416     /* Print out client information */
417     struct sockaddr_in client_socket_addr;
418     socklen_t len;
419     len = sizeof(client_socket_addr);
420     char client_ip[32];
421     unsigned int ephemeral_port;
422
423     bzero(&client_socket_addr, len);
424
425     if (getsockname(info->client_socket, (struct sockaddr *)&client_socket_addr, &len)
426 == 0) {
427         /* get ip and the temp port*/
428         inet_ntop(AF_INET, &client_socket_addr.sin_addr, client_ip, sizeof(client_ip));
429         ephemeral_port = ntohs(client_socket_addr.sin_port);
430
431         /* print contents of ss*/
432         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
433         printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d\n", time_buf,
434 client_ip, ephemeral_port);
435         fflush(stdout);
436     }
437
438     char *job = (char *)req->job;
439
440     switch(get_command_from_request(job)) {
441     case LIST: {
442         char listing[1024];
443         get_media_list(".", listing, 1024);
444         // send the header packet
445         send_header(info->client_socket, info->port, strlen(listing), "Text", 100);
446         if(send(info->client_socket, listing, strlen(listing), 0) == -1) {
447             get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
448             printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d : Error
449 sending list\n", time_buf, client_ip, ephemeral_port);
450         }
451         break;
452     }
453     case GET: {
454         // get the length of the file needed to be read.
455         FILE *fp = fopen(&(job[4]), "rb");
456
457         if (fp == NULL) {
458             send_header(info->client_socket, info->port, 0, "", 404);
459             break;
460         }
461
462         fseek(fp, 0L, SEEK_END);
463         size_t len = ftell(fp);
464         fseek(fp, 0L, SEEK_SET);
465         fclose(fp);
466
467         // get file extension
468         const char *extension = get_file_ext(job + 4);
469
470         // send header information
471         send_header(info->client_socket, info->port, len, extension, 100);
472
473         get_time_spec_to_string(req->job, TIME_BUFFER_LEN);
474         printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d : Sent Header

```

```

472         Information\n", time_buf, client_ip, ephemeral_port);
473         // send requested media
474         send_media(info->client_socket, job + 4, len);
475
476         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
477         printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Sent: %s\n",
478             time_buf, client_ip, ephemeral_port, job);
479         break;
480     }
481     case EXIT:
482         close(info->client_socket);
483         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
484         printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Closed
485             connection with client: %d\n", time_buf, client_ip, ephemeral_port,
486             info->client_socket);
487         return ;
488     default:
489         // invalid request header
490         send_header(info->client_socket, info->port, 0, "", 301);
491         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
492         printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Invalid
493             request\n", time_buf, client_ip, ephemeral_port);
494         break;
495     }
496 }
497
498 void handle_request(void *client_sd)
499 {
500     /* Some vairable declaration */
501     char time_buf[TIME_BUFFER_LEN];
502     handler_arg_t* info = ((handler_arg_t*)client_sd);
503
504     /* Print out client information */
505     struct sockaddr_in client_socket_addr;
506     socklen_t len;
507     len = sizeof(client_socket_addr);
508     char client_ip[32];
509     unsigned int ephemeral_port;
510
511     bzero(&client_socket_addr, len);
512
513     if (getsockname(info->client_socket, (struct sockaddr *)&client_socket_addr, &len)
514         == 0) {
515         /* get ip and the temp port*/
516         inet_ntop(AF_INET, &client_socket_addr.sin_addr, client_ip, sizeof(client_ip));
517         ephemeral_port = ntohs(client_socket_addr.sin_port);
518
519         /* print contents of ss*/
520         get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
521         printf("%s: Handle Request: Client IP: %s Ephemeral Port: %d\n", time_buf,
522             client_ip, ephemeral_port);
523         fflush(stdout);
524     }
525 }
526
527 while(1) {
528     char buf[BUFLen] = {0};
529     char *bp = buf;
530     int bytes_to_read = BUFLen;
531     int n = 0;
532     while ((n = read(info->client_socket, bp, bytes_to_read)) > 0) {
533         bp += n;
534         bytes_to_read -= n;
535     }
536
537     if (bp <= 0) {
538         // client probably disconnected
539         close(info->client_socket);
540     }
541 }

```

```

534     }
535     get_time_spec_to_string(time_buf, BUFLLEN);
536     printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Command Received
string: %s", time_buf, client_ip, ephemeral_port, buf);

537
538     /* put a null character at the end */
539     int size = strlen(buf);
540     buf[strcspn(buf, "\n")] = 0;
541
542     switch(get_command_from_request(buf)) {
543         case LIST: {
544             char listing[1024];
545             get_media_list(".", listing, 1024);
546             // send the header packet
547             send_header(info->client_socket, info->port, strlen(listing), "Text",
100);
548             if(send(info->client_socket, listing, strlen(listing), 0) == -1) {
549                 get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
550                 printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d :
Error sending list\n", time_buf, client_ip, ephemeral_port);
551             }
552             break;
553         }
554         case GET: {
555             // get the length of the file needed to be read.
556             FILE *fp = fopen(&(buf[4]), "rb");
557
558             if (fp == NULL) {
559                 send_header(info->client_socket, info->port, 0, "", 404);
560                 break;
561             }
562
563             fseek(fp, 0L, SEEK_END);
564             size_t len = ftell(fp);
565             fseek(fp, 0L, SEEK_SET);
566             fclose(fp);
567
568             // get file extension
569             const char *extension = get_file_ext(buf + 4);
570
571             // send header information
572             send_header(info->client_socket, info->port, len, extension, 100);
573
574             get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
575             printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Sent
Header Information\n", time_buf, client_ip, ephemeral_port);
576
577             // send requested media
578             send_media(info->client_socket, buf + 4, len);
579
580             get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
581             printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Sent:
%s\n", time_buf, client_ip, ephemeral_port, buf);
582             break;
583         }
584         case EXIT:
585             close(info->client_socket);
586             get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
587             printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Closed
connection with client: %d\n", time_buf, client_ip, ephemeral_port,
info->client_socket);
588             return ;
589         default:
590             // invalid request header
591             send_header(info->client_socket, info->port, 0, "", 301);
592             get_time_spec_to_string(time_buf, TIME_BUFFER_LEN);
593             printf("%s: Handle_Request: Client IP: %s Ephemeral Port: %d : Invalid
request\n", time_buf, client_ip, ephemeral_port);
594             break;

```

```
595         }
596     }
597 }
598
```

```

1  /* A simple TCP client */
2  #include <stdio.h>
3  #include <netdb.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <string.h> //Added string library
8  #include <strings.h> //For bzero function
9  #include <stdlib.h> //Added standard library
10 #include <unistd.h>
11 #include <signal.h>
12
13 #include "media_transfer.h"
14 #include "parser.h"
15
16 #define SERVER_TCP_PORT      (3000)
17
18 int main(int argc, char **argv)
19 {
20     sigaction(SIGPIPE, &(struct sigaction){SIG_IGN}, NULL);
21
22     int n, bytes_to_read;
23     int batch_mode = 0;
24     int sd, port;
25     struct hostent *hp;
26     struct sockaddr_in server;
27     char *host, *bp, rbuf[BUFLEN], sbuf[BUFLEN];
28
29     switch(argc) {
30     case 2:
31         host = argv[1];
32         if (strrchr(host, ':')) {
33             port = atoi(strrchr(host, ':') + 1);
34             char *ope = strrchr(host, ':');
35             *ope = 0;
36         } else port = SERVER_TCP_PORT;
37         break;
38     case 3:
39         host = argv[1];
40         if (strrchr(host, ':')) {
41             port = atoi(strrchr(host, ':') + 1);
42             char *ope = strrchr(host, ':');
43             *ope = 0;
44         } else port = SERVER_TCP_PORT;
45         batch_mode = 1;
46         break;
47     default:
48         fprintf(stderr, "Usage: %s <host>[:port] [script]\n", argv[0]);
49         exit(1);
50     }
51
52     /* Create a stream socket */
53     if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
54         fprintf(stderr, "Can't create a socket\n");
55         exit(1);
56     }
57
58     /* Find the server to connect to */
59     bzero((char *)&server, sizeof(struct sockaddr_in));
60     server.sin_family = AF_INET;
61     server.sin_port = htons(port);
62     if ((hp = gethostbyname(host)) == NULL) {
63         fprintf(stderr, "Can't get server's address\n");
64         exit(1);
65     }
66
67     printf("h_length = %d\n", hp->h_length);
68
69     bcopy(hp->h_addr_list[0], (char *)&server.sin_addr, hp->h_length);

```

```

70
71     /* Connecting to the server */
72     if (connect(sd, (struct sockaddr *)&server, sizeof(server)) == -1) {
73         fprintf(stderr, "Can't connect\n");
74         exit(1);
75     }
76     printf("Connected: server's address is %s\n", hp->h_name);
77
78     if(batch_mode) {
79         process_batch(sd, argv[2]);
80     }
81     else {
82         char time_stamp[TIME_BUFFER_LEN];
83         get_time_spec_to_string(time_stamp, TIME_BUFFER_LEN);
84         while (1) {
85             printf("%s: TX: ", time_stamp);
86             fgets(sbuf, BUFLen, stdin);          /* get user's text */
87             if(strcmp(sbuf, "exit\n") == 0) {
88                 write(sd, sbuf, BUFLen);
89                 close(sd);
90                 break;
91             }
92             else {
93                 printf("%s: Sent Command: %s\n", time_stamp, sbuf);
94                 handle_command(sd, sbuf, BUFLen);
95             }
96         }
97     }
98     return 0;
99 }

```

```

1  #ifndef __LINKED_LIST_H__
2  #define __LINKED_LIST_H__
3
4  #include <stdlib.h>
5
6  typedef struct node {
7      void *job;
8      size_t job_size;
9      void *owner;
10     struct node *next;
11     struct node *prev;
12 } list;
13
14 list *create_list () {
15     list *new_list = (list *) malloc(sizeof(list));
16     new_list->job = NULL;
17     new_list->job_size = 0;
18     new_list->owner = NULL;
19     new_list->next = NULL;
20     new_list->prev = NULL;
21
22     return new_list;
23 }
24
25 void add_job (list *l, void *job, size_t job_size, void *owner) {
26     struct node *job_node = (struct node *) malloc(sizeof(struct node));
27     job_node->job = job;
28     job_node->job_size = job_size;
29     job_node->owner = owner;
30
31     struct node *current = l;
32     while (current->next != NULL) {
33         if (job_size < current->job_size) {
34             job_node->next = current;
35             job_node->prev = current->prev;
36             current->prev->next = job_node;
37             current->prev = job_node;
38
39             return;
40         }
41
42         current = current->next;
43     }
44
45     current->next = job_node;
46     job_node->prev = current;
47 }
48
49
50
51 void *get_job (list *l) {
52     list *nd = l->next;
53     if (!nd) return NULL;
54     void *job = nd->job;
55
56     if (l->next->next)
57         l->next->next->prev = l;
58     l->next = l->next->next;
59
60     return job;
61 }
62
63 #endif
64

```



```

1  #ifndef __QUEUE_H__
2  #define __QUEUE_H__
3
4  #include <limits.h>
5  #include <stdlib.h>
6
7  typedef struct {
8      int front, rear, size;
9      unsigned capacity;
10     void** job;
11 } Queue;
12
13 Queue* createQueue(unsigned capacity)
14 {
15     Queue* queue = (Queue*)malloc(
16         sizeof(Queue));
17     queue->capacity = capacity;
18     queue->front = queue->size = 0;
19
20     // This is important, see the enqueue
21     queue->rear = capacity - 1;
22     queue->job = (void*)malloc(
23         queue->capacity * sizeof(int));
24     return queue;
25 }
26
27 int isFull(Queue* queue)
28 {
29     return (queue->size == queue->capacity);
30 }
31
32 // Queue is empty when size is 0
33 int isEmpty(Queue* queue)
34 {
35     return (queue->size == 0);
36 }
37
38 void enqueue(Queue* queue, void* item)
39 {
40     if (isFull(queue))
41         return;
42     queue->rear = (queue->rear + 1)
43         % queue->capacity;
44     queue->job[queue->rear] = item;
45     queue->size = queue->size + 1;
46 }
47
48 void* dequeue(Queue* queue)
49 {
50     if (isEmpty(queue))
51         return NULL;
52     void* item = queue->job[queue->front];
53     queue->front = (queue->front + 1)
54         % queue->capacity;
55     queue->size = queue->size - 1;
56     return item;
57 }
58
59 void* random_dequeue(Queue *queue)
60 {
61     if (isEmpty(queue))
62         return NULL;
63     else if (queue->size == 1)
64         return dequeue(queue);
65
66     int lower_limit = 0;
67     int upper_limit = queue->size - 1;
68
69     int random_index = (rand() % (upper_limit - lower_limit) + 1) + lower_limit;

```

```
70
71     /* swap the random index with the one at front and then call dequeue */
72
73     /* get the pointer at random index, and make a copy of it*/
74     void *temp = queue->job[random_index];
75
76     /* the pointer at random index points to same place as front pointer*/
77     queue->job[random_index] = queue->job[0];
78
79     /* front pointer now points where the old random index pointed to */
80     queue->job[0] = temp;
81
82     /* return normal dequeue - random pointer will be returned */
83     return dequeue(queue);
84 }
85
86 #endif
```

```

1  #ifndef _MEDIA_TRASNFER_H_
2  #define _MEDIA_TRASNFER_H_
3
4  #include <stdio.h>
5
6  /*
7   * @param fp      - pointer to the media to be sent
8   * @param sockfd  - client socke to send the media to
9   */
10 int send_media(int sockfd, const char *media_path, size_t length);
11
12 /*
13  * @param sockfd  - client socket to receive the media on
14  * @param filename - filename to write received data to
15  */
16 int receive_media(int sockfd, const char *media_path, size_t length);
17
18 /*
19  * @param path      - sends lists all the media under this path
20  * @param buffer     - place to store the listing to
21  * @param buffer_size - size of the buffer passed
22  * @returns
23  *      1 if success, -1 if failure
24  */
25 int get_media_list(const char *path, char *buffer, size_t buffer_size);
26
27 /*
28  * @param client_socket - client socket to send header to
29  * @param port          - port socket is hosted on
30  * @param media_size    - size of media to be sent
31  * @param media_type    - type of the media to be sent
32  * @returns
33  *      1 if sucess, -1 if fail
34  */
35 int send_header(int client_socket, int port, size_t media_size, const char *media_type,
36 int status);
37 #endif

```

```

1  #include <arpa/inet.h>
2  #include <dirent.h>
3  #include <netdb.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <unistd.h>
10
11 #include "media_transfer.h"
12
13 #define LEN 1024
14
15 int send_media(int sockfd, const char *media_path, size_t length) {
16     int n;
17     char *data = malloc(length);
18
19     FILE *fp = fopen(media_path, "rb");
20     if(fp == NULL){
21         printf("File: %s, not Found", media_path);
22         return -1;
23     }
24
25     size_t sent = 0;
26     fread(data, length, 1, fp);
27     while(sent < length) {
28         size_t t = send(sockfd, data, length, 0);
29         if (t != -1) {
30             sent += t;
31         } else {
32             perror("send_media");
33             exit(1);
34         }
35     }
36
37     fclose(fp);
38     free(data);
39     return 1;
40 }
41
42 int receive_media(int sockfd, const char *filename, size_t length) {
43     unsigned int n = 0;
44     size_t pos = 0;
45     FILE *fp;
46     char buffer[LEN];
47     char *media = malloc(length);
48
49     while (1) {
50         n = read(sockfd, buffer, LEN);
51         if (n < 0) continue;
52         memcpy(media + pos, buffer, n);
53         pos += n;
54         if (pos >= length) break;
55     }
56
57     fp = fopen(filename, "w");
58     fwrite(media, length, 1, fp);
59     fclose(fp);
60     free(media);
61
62     return 1;
63 }
64
65 int get_media_list(const char *path, char *buffer, size_t buffer_size) {
66     DIR *dh = opendir(path);
67     struct dirent *d;
68     struct stat fstat;
69

```

```

70     int n = 0;
71     n += sprintf(buffer, "\tSize\t\tName\n");
72     while((d = readdir(dh)) != NULL) {
73         stat(d->d_name, &fstat);
74         n += sprintf(buffer + n, "\t%ld\t\t%s\n", fstat.st_size, d->d_name);
75     }
76     closedir(dh);
77     return 1;
78 }
79
80 int send_header(int client_socket, int port, size_t media_size, const char *media_type,
81 int status) {
82     char host[256];
83     char *IP;
84     struct hostent *host_entry;
85     int hostname;
86
87     //find the host name
88     hostname = gethostname(host, sizeof(host));
89     if(hostname == -1) {
90         printf("Cannot find host information");
91     }
92
93     //find host information
94     host_entry = gethostbyname(host);
95     if(host_entry == NULL) {
96         printf("Cannot find the host from id\n");
97     }
98
99     //Convert into IP string
100     IP = inet_ntoa(*((struct in_addr*) host_entry->h_addr_list[0]));
101
102     // create the header
103     char header[LEN];
104     int n = 0;
105     n += sprintf(header, "Status: %d\r\n", status);           // req is valid
106     n += sprintf(header + n, "Host: %s:%d\r\n", IP, port);    // append host
107     n += sprintf(header + n, "Type: %s\r\n", media_type);      // append file type
108     n += sprintf(header + n, "Length: %ld\r\n\r\n", media_size); // append file
109     length
110
111     // finally send the header packet
112     if(send(client_socket, header, n, 0) == -1) {
113         return -1;
114     }
115     else{
116         return 0;
117     }
118 }

```

```

1  #ifndef _PARSER_H_
2  #define _PARSER_H_
3
4  #define BUFLLEN          (256)      /* buffer length */
5  #define TIME_BUFFER_LEN  128      /* lenght of time buffer to print time stamp*/
6
7  typedef struct {
8      int status;
9      size_t length;
10     char *type;
11     char *host;
12 } header;
13
14 enum commands {
15     INVALID,
16     LIST,
17     GET,
18     COMMENT,
19     EXIT
20 };
21
22 typedef enum commands command_t;
23
24 /*
25  * A constructor function for header struct
26  * @returns an empty header struct
27  */
28 header create_header();
29
30 /*
31  * @param request - string line to validate
32  * @returns
33  *     1 if valid, -1 if not
34  */
35 command_t get_command_from_request(const char *request);
36
37 /*
38  * @param header - buffer containing header information
39  * @param line_number - specfic line of header buffer to return
40  * @returns
41  *     a particular line from header buffer
42  */
43 char * get_line(char * header_text, unsigned int line_number);
44
45 /*
46  * @param string - buffer to find occurence of chracter from
47  * @param c       - value of char whose occurence to be found
48  * @param n       - number of occurences to be found
49  * @returns       - position index of the nth occurence.
50  */
51 int get_occurrence_n(char * string, char c, int n);
52
53 /*
54  * @param buf - buffer to store the time spec in
55  * @param buflen - size of the buffer
56  */
57 void get_time_spec_to_string(char *buf, size_t buflen);
58
59 /*
60  * @param str - string to find the number of lines it contains
61  * @returns    - number of lines in a string
62  */
63 int count_lines(char const *str);
64
65 /*
66  * @param socket - socket id to receive header text from
67  * @returns       - prints and then returns a buffer containing header text
68  */
69 char * read_header_text(int socket);

```

```

70
71  /*
72  * @param header_text - buffer to read from
73  * @param header_ptr  - storage location to store information
74  * @returns           - success or failure
75  */
76  int buffer_to_header(char * header_text, header *ptr);
77
78  /* Handle command from a string value
79  * @param socker      - socket to use for server communication
80  * @param command     - command string read from usr or file
81  * @param len         - len of incoming command
82  * @returns           - success or failure
83  */
84  int handle_command(int socket, char *command, int len);
85
86  /*
87  * Handle any command request from client
88  * @param server_socket - socket to communicate to server
89  * @param command       - string containing the full get <filename>
90  * @param               - strlen of command
91  * @returns             - success or failure
92  */
93  int process_command(int server_socket, char *command, int len);
94
95  /*
96  * Runs commands from batch script
97  * @param clientrc_path - path to read client commands from
98  */
99  int process_batch(int socket, char * clienrc_path);
100
101  #endif

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <time.h>
6  #include "media_transfer.h"
7  #include "parser.h"
8
9  /*
10 * This functions checks if a request contains
11 * "list" or "get" as the first few bytes. Function,
12 * then returns a command type based on request.
13 */
14 command_t get_command_from_request(const char *request) {
15     if(request == NULL) {
16         return INVALID;
17     }
18     else if(request[0] == '#') {
19         return COMMENT;
20     }
21     else if(strncmp(request, "list", 4) == 0) {
22         return LIST;
23     }
24     else if(strncmp(request, "get", 3) == 0) {
25         int len = strlen(request);
26         if(len <= 4) { // no file name specified
27             printf("No file name specified for get command\n");
28             return INVALID;
29         }
30         return GET;
31     }
32     else if(strncmp(request, "exit", 4) == 0) {
33         return EXIT;
34     }
35     else {
36         return INVALID;
37     }
38 }
39
40 /*
41 * A constructor function for header struct
42 * @returns an empty header struct
43 */
44 header create_header() {
45     header h;
46     h.status = 0;
47     h.length = 0;
48     h.type = 0;
49     h.host = 0;
50
51     return h;
52 }
53
54 /*
55 * @param string - buffer to find occurrence of chracter from
56 * @param c      - value of char whose occurrence to be found
57 * @param n      - number of occurrences to be found
58 * @returns      - position index of the nth occurrence.
59 */
60 int get_occurrence_n(char * string, char c, int n) {
61     if (string != NULL) {
62         int occ = 0;
63         int i;
64         for (i = 0; i < strlen(string); i++) {
65             if (string[i] == c) {
66                 if ((++occ) == n) return i;
67             }
68         }
69     }

```



```

70
71     return -1;
72 }
73
74 void get_time_spec_to_string(char *buf, size_t buflen) {
75     struct timespec ts;
76     timespec_get(&ts, 1); //TIME_UTC = 1
77     char temp[buflen];
78     strftime(temp, buflen, "%D %T", gmtime(&ts.tv_sec));
79     sprintf(buf, "%s.%09ld UTC", temp, ts.tv_nsec);
80 }
81
82 /*
83  * @param str - string to find the number of lines it contains
84  * @returns    - number of lines in a string
85  */
86 int count_lines(char const *str)
87 {
88     char const *p = str;
89     int count;
90     for (count = 0; ; ++count) {
91         p = strstr(p, "\r\n");
92         if (!p)
93             break;
94         p = p + 2;
95     }
96     return count - 1;
97 }
98
99 /*
100  * @param header - buffer containing header information
101  * @param line_number - specific line of header buffer to return
102  * @returns
103  *     a particular line from header buffer
104  */
105 char * get_line(char * header_text, unsigned int line_number) {
106     char * ret = 0;
107     int line_count = 1;
108     int start = -2;
109     int cur = 0;
110     int i;
111     for (i = 0; i < line_number; ++i) {
112         start = cur;
113         cur = start + 2;
114         while (header_text[cur] && header_text[cur] != '\r') {
115             if (header_text[cur + 1] && header_text[cur + 1] == '\n') break;
116             cur++;
117         }
118
119         if (header_text[cur + 2] && header_text[cur + 2] == '\r') {
120             if (header_text[cur + 3] && header_text[cur + 3] == '\n') {
121                 break;
122             }
123         }
124
125         line_count++;
126     }
127     if (line_number > line_count) return NULL;
128
129     if (line_number == 1) {
130         ret = calloc(cur + 1, sizeof(char));
131         strncpy(ret, header_text, cur);
132     }
133     else {
134         ret = calloc(cur - start - 1, sizeof(char));
135         strncpy(ret, header_text + start + 2, cur - start - 2);
136     }
137
138     return ret;

```

```

139 }
140
141 /*
142  * @param socket - socket id to receive header text from
143  * @returns      - prints and then returns a buffer containing header text
144  */
145 char * read_header_text(int socket) {
146     char buffer[BUFLLEN] = {0};
147     int buf_ind = 0;
148     int ret_size = 0;
149     int cont = 1;
150     char *header_text = NULL;
151     while (cont) {
152         while (buf_ind < BUFLLEN && 1 == read(socket, &buffer[buf_ind], 1)) {
153             if (buf_ind > 2 &&
154                 '\n' == buffer[buf_ind] &&
155                 '\r' == buffer[buf_ind - 1] &&
156                 '\n' == buffer[buf_ind - 2] &&
157                 '\r' == buffer[buf_ind - 3])
158             {
159                 cont = 0;
160                 break;
161             }
162             buf_ind++;
163         }
164
165         buf_ind++;
166
167         if (header_text == NULL) {
168             header_text = (char*)malloc(buf_ind * sizeof(char) + 1);
169             memset(header_text, 0, buf_ind + 1);
170             strncpy(header_text, buffer, buf_ind);
171
172             ret_size = buf_ind + 1;
173         } else {
174             header_text = (char*) realloc(header_text, (ret_size += buf_ind));
175             memset(header_text + ret_size - 1, 0, 1);
176             strncat(header_text, buffer, buf_ind);
177         }
178     }
179
180     memset(buffer, 0, BUFLLEN);
181     buf_ind = 0;
182 }
183
184 //printf("%s\n", header_text);
185 return header_text;
186 }
187
188 /*
189  * @param header_text - buffer to read from
190  * @param h            - storage location to store information
191  * @returns            - success or failure
192  */
193 int buffer_to_header(char * header_text, header *h) {
194
195     if(!header_text) {
196         return -1;
197     }
198
199     char * line = NULL;
200     int current = 1;
201     int additional_count = 0;
202     while ((line = get_line(header_text, current)) != NULL) {
203         int token_loc = get_occurrence_n(line, ':', 1);
204         if (token_loc > 0) {
205             char key[token_loc + 1];
206             char value[strlen(line) - token_loc];
207

```

```

208     memset(key, 0, sizeof(key));
209     memset(value, 0, sizeof(value));
210     int i;
211     for (i = 0; i < sizeof(key) - 1; i++) key[i] = line[i];
212     for (i = 0; i < sizeof(value) - 1; i++) value[i] = line[token_loc + i + 2];
213
214     if (strcmp(key, "Status") == 0) h->status = atoi(value);
215     else if (strcmp(key, "Host") == 0) {
216         h->host = malloc(sizeof(value));
217         strcpy(h->host, value);
218     } else if (strcmp(key, "Type") == 0) {
219         h->type = malloc(sizeof(value));
220         strcpy(h->type, value);
221     } else if (strcmp(key, "Length") == 0) h->length = atoi(value);
222     }
223
224     free(line);
225     line = NULL;
226     if (++current > count_lines(header_text)) break;
227 }
228 return 1;
229 }
230
231 /* Handle command from a string value
232  * @param socker          - socket to use for server communication
233  * @param command         - command string read from usr or file
234  * @param len             - len of incoming command
235  * @returns               - success or failure
236  */
237 int handle_command(int socket, char *command, int len) {
238     switch (get_command_from_request(command)) {
239         case GET:
240             process_command(socket, command, BUFLLEN);
241             break;
242         case LIST:
243             process_command(socket, command, BUFLLEN);
244             break;
245         case EXIT:
246             printf("Good bye\n");
247             return 1;
248             break;
249         case INVALID:
250             printf("Invalid Command: %s\n", command);
251         default:
252             break;
253     }
254     return 1;
255 }
256
257 /*
258  * Handle get request from client
259  * @param server_socket - socket to communicate to server
260  * @returns - success or failure
261  */
262 int process_command(int server_socket, char *command, int len) {
263
264     /* send out user command */
265     write(server_socket, command, len);
266
267     // read header response
268     char *header_text = read_header_text(server_socket);
269     char time_stamp[TIME_BUFFER_LEN];
270     get_time_spec_to_string(time_stamp, BUFLLEN);
271     printf("%s: Header Response Received\n", time_stamp);
272     if(!header_text) {
273         perror("fatal error\n");
274     }
275
276     // store buffer information to header struc

```

```

277     header h = create_header();
278     buffer_to_header(header_text, &h);
279
280     free(header_text);
281     header_text = NULL;
282
283     get_time_spec_to_string(time_stamp, TIME_BUFFER_LEN);
284     printf("%s: Status:%d Host:%s Length:%ld Type:%s \n", time_stamp, h.status,
h.host, h.length, h.type);
285
286     switch (h.status) {
287         case 100:
288             if (strcmp(h.type, "Text") == 0) {
289                 char list[h.length + 1];
290                 list[h.length];
291                 memset(list, 0, h.length + 1);
292
293                 size_t received = 0;
294
295                 while (received < h.length) {
296                     if (read(server_socket, list + received, 1)) ++received;
297                 }
298
299                 printf("%s\n", list);
300                 get_time_spec_to_string(time_stamp, TIME_BUFFER_LEN);
301                 printf("%s: File Listing Received\n", time_stamp);
302             }
303             else {
304                 command[strcspn(command, "\n")] = 0;
305                 // get output name of the file from user
306                 char output_name[BUFLEN];
307                 printf("%s: Name of the file to put data received from server to: ",
time_stamp);
308                 fgets(output_name, BUFLen, stdin);
309                 output_name[strcspn(output_name, "\n")] = 0;
310
311                 // store to the output file
312                 receive_media(server_socket, output_name, h.length);
313                 get_time_spec_to_string(time_stamp, TIME_BUFFER_LEN);
314                 printf("%s: Media Received and Downloaded\n", time_stamp);
315             }
316             break;
317         case 301:
318             fprintf(stderr, "Unknown command!\n");
319             break;
320         case 404:
321             fprintf(stderr, "File not found!\n");
322             break;
323         default:
324             fprintf(stderr, "Undefined error!\n");
325             break;
326     }
327 }
328
329
330 /*
331  * Runs commands from batch script
332  * @param clienrc_path - path to read client commands from
333  */
334 int process_batch(int socket, char * clienrc_path) {
335     if(!clienrc_path) {
336         perror("Could not find script path\n");
337         return - 1;
338     }
339
340     FILE* fp = fopen(clienrc_path, "r");
341     if(!fp) {
342         perror("Could not find script path\n");
343         return -1;

```

```
344     }
345     char buffer[BUFLLEN];
346     while(fgets(buffer, BUFLLEN, fp)){
347         switch(get_command_from_request(buffer)) {
348             case GET:
349                 handle_command(socket, buffer, BUFLLEN);           /* send it out */
350                 break;
351             case LIST:
352                 handle_command(socket, buffer, BUFLLEN);           /* send it out */
353                 break;
354             case EXIT:
355                 return 1;
356             default:
357                 break;
358         }
359     }
360 }
361
```

SJF Traces

```
Script started on Fri 30 Apr 2021 06:21:13 PM EDT
[daxpate@in-csci-rrpc02 Project-4]$ make
gcc -g -c media_transfer.c
gcc -g -c parser.c
[daxpate@in-csci-rrpc02 Project-4]$ ./server
*****Server configuration*****
Port Number: 1334
Num Threads: 4
Max Reqs: 10
Media Path: /home/daxpate/ece40800/Project-4
Sched type: 2
*****
```

```
04/30/21 22:22:19.931429242 UTC: Main: Accepting New Connection: 5
04/30/21 22:22:19.931429242 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:22:19.931558717 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:27.310656367 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:23:27.310736915 UTC: Main: Added New Client to the Job queue

04/30/21 22:23:27.310758541 UTC: Main: Accepting New Connection: 6
04/30/21 22:23:27.310758541 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:23:27.310794516 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:27.310791989 UTC: Watch Request: Thead 140395902371584: Handling client 0
04/30/21 22:23:27.310835365 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:31.090705719 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:23:31.090758125 UTC: Main: Added New Client to the Job queue

04/30/21 22:23:31.090777397 UTC: Main: Accepting New Connection: 7
04/30/21 22:23:31.090777397 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:23:31.090812990 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:31.090813366 UTC: Watch Request: Thead 140395885586176: Handling client 0
04/30/21 22:23:31.090873986 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:31.090851031 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:23:31.090932877 UTC: Main: Added New Client to the Job queue
04/30/21 22:23:31.090949967 UTC: Watch Request: Thead 140395893978880: Handling client 0

04/30/21 22:23:31.090956573 UTC: Main: Accepting New Connection: 9
04/30/21 22:23:31.090956573 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:23:31.091024547 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:23:31.090980235 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:31.361506351 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:24:31.361558104 UTC: Main: Added New Client to the Job queue

04/30/21 22:24:31.361577666 UTC: Main: Accepting New Connection: 8
04/30/21 22:24:31.361577666 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:24:31.361605656 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:31.361604664 UTC: Watch Request: Thead 140395877193472: Handling client 0
04/30/21 22:24:31.361667877 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:31.361643858 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:24:31.361730088 UTC: Main: Added New Client to the Job queue

04/30/21 22:24:31.361754699 UTC: Main: Accepting New Connection: 11
04/30/21 22:24:31.361754699 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:24:31.361796462 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:31.361753928 UTC: Watch Request: Thead 140395885586176: Handling client 0
04/30/21 22:24:31.361841621 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:47.327535383 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:24:47.327581513 UTC: Main: Added New Client to the Job queue

04/30/21 22:24:47.327603906 UTC: Main: Accepting New Connection: 10
04/30/21 22:24:47.327603906 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:24:47.327632196 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:47.327630238 UTC: Watch Request: Thead 140395893978880: Handling client 0
04/30/21 22:24:47.327670086 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:24:47.327695354 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:24:47.327729749 UTC: Main: Added New Client to the Job queue
04/30/21 22:24:47.327773681 UTC: Watch Request: Thead 140395877193472: Handling client 0
04/30/21 22:24:47.327804381 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334

04/30/21 22:25:09.150677649 UTC: Main: Accepting New Connection: 13
04/30/21 22:25:09.150677649 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:25:09.150709497 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:25:19.193575576 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
```

```

Command Received string: list
04/30/21 22:25:19.193630907 UTC: Main: Added New Client to the Job queue
04/30/21 22:25:19.193677938 UTC: Watch Request: Thead 140395877193472: Handling client 0
04/30/21 22:25:19.193726035 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334

04/30/21 22:25:55.382138598 UTC: Main: Accepting New Connection: 12
04/30/21 22:25:55.382138598 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:25:55.382170464 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:25:58.407233882 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:25:58.407279803 UTC: Main: Added New Client to the Job queue
04/30/21 22:25:58.407326689 UTC: Watch Request: Thead 140395902371584: Handling client 0
04/30/21 22:25:58.407395067 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334

04/30/21 22:26:38.283256548 UTC: Main: Accepting New Connection: 14
04/30/21 22:26:38.283256548 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:26:38.283288904 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:28:07.222541269 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: 04/30/21 22:28:07.222567759 UTC: Main: Added New Client to the Job queue

04/30/21 22:28:07.222603493 UTC: Main: Accepting New Connection: 15
04/30/21 22:28:07.222603493 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:28:07.222614235 UTC: Watch Request: Thead 140395885586176: Handling client 0
04/30/21 22:28:07.222678046 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:28:07.222651556 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:28:07.222739385 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:28:07.222767274 UTC: Main: Added New Client to the Job queue
04/30/21 22:28:07.222812008 UTC: Watch Request: Thead 140395893978880: Handling client 0
04/30/21 22:28:07.222838523 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:28:07.224033895 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Invalid request

04/30/21 22:28:21.836176963 UTC: Main: Accepting New Connection: 16
04/30/21 22:28:21.836176963 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:28:21.836208229 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:29:17.796608289 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:29:17.796664531 UTC: Main: Added New Client to the Job queue

04/30/21 22:29:17.796686460 UTC: Main: Accepting New Connection: 17
04/30/21 22:29:17.796686460 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:29:17.796714730 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:29:17.796711237 UTC: Watch Request: Thead 140395893978880: Handling client 0
04/30/21 22:29:17.796774530 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:29:17.796750862 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: get r.mp3
04/30/21 22:29:17.799260936 UTC: Main: Added New Client to the Job queue
04/30/21 22:29:17.799272923 UTC: Watch Request: Thead 140395885586176: Handling client 0
04/30/21 22:29:17.799331226 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334

04/30/21 22:29:17.799309484 UTC: Main: Accepting New Connection: 19
04/30/21 22:29:17.799309484 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:29:17.799401487 UTC: Handle Request: Client IP: 10.234.136.56 Ephemeral Port: 1334
04/30/21 22:29:17.799420888 UTC: Handle_Request: Client IP: 10.234.136.56 Ephemeral Port: 1334 :
Command Received string: list
04/30/21 22:29:17.799446670 UTC: Main: Added New Client to the Job queue

04/30/21 22:29:17.799462158 UTC: Main: Accepting New Connection: 21
04/30/21 22:29:17.799462459 UTC: Watch Request: Thead 140395902371584: Handling client 0
04/30/21 22:29:17.799462158 UTC: Main: Adding New Client to the Job queue...
04/30/21 22:29:17.799456475 UTC: Watch Request: Thead 140395877193472: Handling client 0
Segmentation fault (core dumped)
[daxpate@in-csci-rrpc02 Project-4]$ exit
exit

```

Script done on Fri 30 Apr 2021 06:29:48 PM EDT



```
Script started on Fri 30 Apr 2021 05:58:30 PM EDT
[pakpatel@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 21:59:48.562711370 UTC: TX: list
04/30/21 21:59:48.562711370 UTC: Sent Command: list
```

```
04/30/21 22:00:51.632465243 UTC: Header Response Received
04/30/21 22:00:51.632514832 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
      Size      Name
4096      .
70      ..
4096      .git
473      .gitignore
445      Makefile
2353      client.c
39      clientrc
1297      linked_list.h
2706      media_transfer.c
1125      media_transfer.h
8911      old.c
9114      parser.c
2615      parser.h
1961      queue.h
9733309      r.mp3
9733309      received.mp3
113      send.txt
9733309      song.mp3
13312      client.o
35776      client
18175      server.c
34072      server.o
48904      server
163      mserver.config
0      dp_sjf_02_server.script
0      dp_sjf_03.script
0      dp_sjf_tesla.script
12520      media_transfer.o
18848      parser.o
```

```
04/30/21 22:00:51.634162503 UTC: File Listing Received
04/30/21 21:59:48.562711370 UTC: TX: get song.mp3
04/30/21 21:59:48.562711370 UTC: Sent Command: get song.mp3
```

```
^C
[pakpatel@in-csci-rrpc03 Project-4]$
[pakpatel@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:05:44.939141222 UTC: TX: list
04/30/21 22:05:44.939141222 UTC: Sent Command: list
```

```
^C
[pakpatel@in-csci-rrpc03 Project-4]$ exit
exit
```

```
Script done on Fri 30 Apr 2021 06:07:08 PM EDT
```

```
Script started on Fri 30 Apr 2021 05:58:33 PM EDT
[pakpatel@in-csci-rrpc04 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:00:08.498528630 UTC: TX: list
04/30/21 22:00:08.498528630 UTC: Sent Command: list
```

```
04/30/21 22:01:51.848854911 UTC: Header Response Received
04/30/21 22:01:51.848906203 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
  Size      Name
  4096      .
  70        ..
  4096      .git
  473        .gitignore
  445        Makefile
  2353      client.c
  39         clientrc
  1297      linked_list.h
  2706      media_transfer.c
  1125      media_transfer.h
  8911      old.c
  9114      parser.c
  2615      parser.h
  1961      queue.h
  9733309   r.mp3
  9733309   received.mp3
  113       send.txt
  9733309   song.mp3
  13312     client.o
  35776     client
  18175     server.c
  34072     server.o
  48904     server
  163       mserver.config
  0         dp_sjf_02_server.script
  0         dp_sjf_03.script
  0         dp_sjf_tesla.script
  12520     media_transfer.o
  18848     parser.o
```

```
04/30/21 22:01:51.850612608 UTC: File Listing Received
04/30/21 22:00:08.498528630 UTC: TX: list
04/30/21 22:00:08.498528630 UTC: Sent Command: list
```

```
^C
[pakpatel@in-csci-rrpc04 Project-4]$ exit
exit
```

```
Script done on Fri 30 Apr 2021 06:07:12 PM EDT
```

```
Script started on Fri 30 Apr 2021 05:58:34 PM EDT
[pakpatel@in-csci-rrpc06 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:00:25.148390722 UTC: TX: list
04/30/21 22:00:25.148390722 UTC: Sent Command: list
```

```
04/30/21 22:02:06.823176118 UTC: Header Response Received
04/30/21 22:02:06.823224540 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
  Size      Name
  4096      .
  70        ..
  4096      .git
  473       .gitignore
  445       Makefile
  2353      client.c
  39        clientrc
  1297      linked_list.h
  2706      media_transfer.c
  1125      media_transfer.h
  8911      old.c
  9114      parser.c
  2615      parser.h
  1961      queue.h
  9733309   r.mp3
  9733309   received.mp3
  113       send.txt
  9733309   song.mp3
  13312     client.o
  35776     client
  18175     server.c
  34072     server.o
  48904     server
  163       mserver.config
  0         dp_sjf_02_server.script
  0         dp_sjf_03.script
  0         dp_sjf_tesla.script
  12520     media_transfer.o
  18848     parser.o
```

```
04/30/21 22:02:06.824861931 UTC: File Listing Received
04/30/21 22:00:25.148390722 UTC: TX: get r.mp3
04/30/21 22:00:25.148390722 UTC: Sent Command: get r.mp3
```

```
^C
[pakpatel@in-csci-rrpc06 Project-4]$ exit
exit
```

```
Script done on Fri 30 Apr 2021 06:07:15 PM EDT
```

```
Script started on Fri 30 Apr 2021 05:58:36 PM EDT
[johjbake@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 21:59:46.652695899 UTC: TX: list
04/30/21 21:59:46.652695899 UTC: Sent Command: list
```

```
04/30/21 22:00:51.631174711 UTC: Header Response Received
04/30/21 22:00:51.631229883 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
```

Size	Name
4096	.
70	..
4096	.git
473	.gitignore
445	Makefile
2353	client.c
39	clientrc
1297	linked_list.h
2706	media_transfer.c
1125	media_transfer.h
8911	old.c
9114	parser.c
2615	parser.h
1961	queue.h
9733309	r.mp3
9733309	received.mp3
113	send.txt
9733309	song.mp3
13312	client.o
35776	client
18175	server.c
34072	server.o
48904	server
163	mserver.config
0	dp_sjf_02_server.script
0	dp_sjf_03.script
0	dp_sjf_tesla.script
12520	media_transfer.o
18848	parser.o

```
04/30/21 22:00:51.632969714 UTC: File Listing Received
04/30/21 21:59:46.652695899 UTC: TX: get r.mp3
04/30/21 21:59:46.652695899 UTC: Sent Command: get r.mp3
```

```
get received.mp3
^C
[johjbake@in-csci-rrpc03 Project-4]$ list
bash: list: command not found...
[johjbake@in-csci-rrpc03 Project-4]$ get song.mp3
bash: get: command not found...
Similar commands are::
'git'
'GET'
[johjbake@in-csci-rrpc03 Project-4]$ GET song.mp3
Can't connect to song.mp3:80 (Bad hostname)
```

```
LWP::Protocol::http::Socket: Bad hostname 'song.mp3' at /usr/share/perl5/LWP/Protocol/http.pm line 51.
```

```
[johjbake@in-csci-rrpc03 Project-4]$ ^C
[johjbake@in-csci-rrpc03 Project-4]$ list
bash: list: command not found...
[johjbake@in-csci-rrpc03 Project-4]$ ls -l
total 28780
-rwxr-xr-x 1 johjbake students 35776 Apr 30 17:43 client
-rw-r--r-- 1 johjbake students 2353 Apr 30 17:35 client.c
-rw-r--r-- 1 johjbake students 13312 Apr 30 17:43 client.o
-rw-r--r-- 1 johjbake students 39 Apr 30 17:35 clientrc
-rw-r--r-- 1 johjbake students 0 Apr 30 17:58 jb_sjf_03.script
-rw-r--r-- 1 johjbake students 0 Apr 30 17:58 jb_sjf_04.script
-rw-r--r-- 1 johjbake students 0 Apr 30 17:59 jb_sjf_tesla.script
-rw-r--r-- 1 johjbake students 1297 Apr 30 17:35 linked_list.h
-rw-r--r-- 1 johjbake students 445 Apr 30 17:35 Makefile
-rw-r--r-- 1 johjbake students 2706 Apr 30 17:35 media_transfer.c
-rw-r--r-- 1 johjbake students 1125 Apr 30 17:35 media_transfer.h
-rw-r--r-- 1 johjbake students 12520 Apr 30 17:55 media_transfer.o
-rw-r--r-- 1 johjbake students 164 Apr 30 17:35 mserver.config
-rw-r--r-- 1 johjbake students 8911 Apr 30 17:35 old.c
-rw-r--r-- 1 johjbake students 9114 Apr 30 17:35 parser.c
-rw-r--r-- 1 johjbake students 2615 Apr 30 17:35 parser.h
-rw-r--r-- 1 johjbake students 18848 Apr 30 17:55 parser.o
-rw-r--r-- 1 johjbake students 1961 Apr 30 17:35 queue.h
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 received.mp3
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 r.mp3
```

```

-rw-r--r-- 1 johjbake students 113 Apr 30 17:35 send.txt
-rwxr-xr-x 1 johjbake students 48904 Apr 30 17:43 server
-rw-r--r-- 1 johjbake students 18145 Apr 30 17:35 server.c
-rw-r--r-- 1 johjbake students 34040 Apr 30 17:43 server.o
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 song.mp3
[johjbake@in-csci-rrpc03 Project-4]$ list
bash: list: command not found...
[johjbake@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:03:15.920073127 UTC: TX: list
04/30/21 22:03:15.920073127 UTC: Sent Command: list

04/30/21 22:03:18.948637093 UTC: Header Response Received
04/30/21 22:03:18.948701256 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
    Size      Name
    4096      .
    70        ..
    4096      .git
    473       .gitignore
    445       Makefile
    2353      client.c
    39        clientrc
    1297      linked_list.h
    2706      media_transfer.c
    1125      media_transfer.h
    8911      old.c
    9114      parser.c
    2615      parser.h
    1961      queue.h
    9733309   r.mp3
    9733309   received.mp3
    113       send.txt
    9733309   song.mp3
    13312     client.o
    35776     client
    18175     server.c
    34072     server.o
    48904     server
    163       mserver.config
    0         dp_sjf_02_server.script
    0         dp_sjf_03.script
    0         dp_sjf_tesla.script
    12520     media_transfer.o
    18848     parser.o

04/30/21 22:03:18.952086755 UTC: File Listing Received
04/30/21 22:03:15.920073127 UTC: TX: get send.txt
04/30/21 22:03:15.920073127 UTC: Sent Command: get send.txt

^C
[johjbake@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:05:42.374282421 UTC: TX: list
04/30/21 22:05:42.374282421 UTC: Sent Command: list

^C
[johjbake@in-csci-rrpc03 Project-4]$ exit
exit

```

Script done on Fri 30 Apr 2021 06:07:29 PM EDT

```
Script started on Fri 30 Apr 2021 05:58:48 PM EDT
[johjbake@in-csci-rrpc04 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:00:05.724885189 UTC: TX: list
04/30/21 22:00:05.724885189 UTC: Sent Command: list
```

```
04/30/21 22:01:51.847463211 UTC: Header Response Received
04/30/21 22:01:51.847510658 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
Size      Name
4096      .
70         ..
4096      .git
473        .gitignore
445        Makefile
2353       client.c
39         clientrc
1297       linked_list.h
2706       media_transfer.c
1125       media_transfer.h
8911       old.c
9114       parser.c
2615       parser.h
1961       queue.h
9733309    r.mp3
9733309    received.mp3
113        send.txt
9733309    song.mp3
13312      client.o
35776      client
18175      server.c
34072      server.o
48904      server
163        mserver.config
0          dp_sjf_02_server.script
0          dp_sjf_03.script
0          dp_sjf_tesla.script
12520      media_transfer.o
18848      parser.o
```

```
04/30/21 22:01:51.849199492 UTC: File Listing Received
04/30/21 22:00:05.724885189 UTC: TX: get parser.c
04/30/21 22:00:05.724885189 UTC: Sent Command: get parser.c
```

```
^M^C
[johjbake@in-csci-rrpc04 Project-4]$ get parser.h
bash: get: command not found...
Similar commands are::
'git'
'GET'
[johjbake@in-csci-rrpc04 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:02:29.622244380 UTC: TX: list
04/30/21 22:02:29.622244380 UTC: Sent Command: list
```

```
04/30/21 22:02:39.670748437 UTC: Header Response Received
04/30/21 22:02:39.670811849 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
Size      Name
4096      .
70         ..
4096      .git
473        .gitignore
445        Makefile
2353       client.c
39         clientrc
1297       linked_list.h
2706       media_transfer.c
1125       media_transfer.h
8911       old.c
9114       parser.c
2615       parser.h
1961       queue.h
9733309    r.mp3
9733309    received.mp3
113        send.txt
9733309    song.mp3
13312      client.o
35776      client
18175      server.c
```

```
34072      server.o
48904      server
163        mserver.config
0          dp_sjf_02_server.script
0          dp_sjf_03.script
0          dp_sjf_tesla.script
12520      media_transfer.o
18848      parser.o
```

```
04/30/21 22:02:39.674273862 UTC: File Listing Received
04/30/21 22:02:29.622244380 UTC: TX: get song.mp3
04/30/21 22:02:29.622244380 UTC: Sent Command: get song.mp3
```

```
^C
[johjbake@in-csci-rrpc04 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:05:45.818752852 UTC: TX: list
04/30/21 22:05:45.818752852 UTC: Sent Command: list
```

```
^C
[johjbake@in-csci-rrpc04 Project-4]$ exit
exit
```

Script done on Fri 30 Apr 2021 06:07:14 PM EDT

Script started on Fri 30 Apr 2021 05:59:17 PM EDT

[johjbake@tesla Project-4]\$ ls -l

total 28780

```
-rwxr-xr-x 1 johjbake students 35776 Apr 30 17:43 client
-rw-r--r-- 1 johjbake students 2353 Apr 30 17:35 client.c
-rw-r--r-- 1 johjbake students 13312 Apr 30 17:43 client.o
-rw-r--r-- 1 johjbake students 39 Apr 30 17:35 clientrc
-rw-r--r-- 1 johjbake students 0 Apr 30 17:58 jb_sjf_03.script
-rw-r--r-- 1 johjbake students 0 Apr 30 17:58 jb_sjf_04.script
-rw-r--r-- 1 johjbake students 0 Apr 30 17:59 jb_sjf_tesla.script
-rw-r--r-- 1 johjbake students 1297 Apr 30 17:35 linked_list.h
-rw-r--r-- 1 johjbake students 445 Apr 30 17:35 Makefile
-rw-r--r-- 1 johjbake students 2706 Apr 30 17:35 media_transfer.c
-rw-r--r-- 1 johjbake students 1125 Apr 30 17:35 media_transfer.h
-rw-r--r-- 1 johjbake students 12520 Apr 30 17:55 media_transfer.o
-rw-r--r-- 1 johjbake students 164 Apr 30 17:35 mservr.config
-rw-r--r-- 1 johjbake students 8911 Apr 30 17:35 old.c
-rw-r--r-- 1 johjbake students 9114 Apr 30 17:35 parser.c
-rw-r--r-- 1 johjbake students 2615 Apr 30 17:35 parser.h
-rw-r--r-- 1 johjbake students 18848 Apr 30 17:55 parser.o
-rw-r--r-- 1 johjbake students 1961 Apr 30 17:35 queue.h
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 received.mp3
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 r.mp3
-rw-r--r-- 1 johjbake students 113 Apr 30 17:35 send.txt
-rwxr-xr-x 1 johjbake students 48904 Apr 30 17:43 server
-rw-r--r-- 1 johjbake students 18145 Apr 30 17:35 server.c
-rw-r--r-- 1 johjbake students 34040 Apr 30 17:43 server.o
-rw-r--r-- 1 johjbake students 9733309 Apr 30 17:35 song.mp3
```

[johjbake@tesla Project-4]\$ ./client in-csci-rrpc02:1334

h\_length = 4

Connected: server's address is in-csci-rrpc02

04/30/21 22:03:59.921906506 UTC: TX:

04/30/21 22:03:59.921906506 UTC: Sent Command:

Invalid Command:

04/30/21 22:03:59.921906506 UTC: TX: ^C

[johjbake@tesla Project-4]\$

[johjbake@tesla Project-4]\$ ./client in-csci-rrpc02:1334

h\_length = 4

Connected: server's address is in-csci-rrpc02

04/30/21 22:05:52.446396454 UTC: TX: list

04/30/21 22:05:52.446396454 UTC: Sent Command: list

^C

[johjbake@tesla Project-4]\$ exit

exit

Script done on Fri 30 Apr 2021 06:07:33 PM EDT



```
Script started on Fri 30 Apr 2021 05:58:56 PM EDT
[daxpate@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 21:59:40.469087843 UTC: TX: list
04/30/21 21:59:40.469087843 UTC: Sent Command: list
```

```
04/30/21 22:00:47.865520171 UTC: Header Response Received
04/30/21 22:00:47.865570817 UTC: Status:100 Host:134.68.136.32:1334 Length:522 Type:Text
  Size      Name
  4096      .
  70        ..
  4096      .git
  473       .gitignore
  445       Makefile
  2353      client.c
  39        clientrc
  1297      linked_list.h
  2706      media_transfer.c
  1125      media_transfer.h
  8911      old.c
  9114      parser.c
  2615      parser.h
  1961      queue.h
  9733309   r.mp3
  9733309   received.mp3
  113       send.txt
  9733309   song.mp3
  13312     client.o
  35776     client
  18175     server.c
  34072     server.o
  48904     server
  163       mserver.config
  0         dp_sjf_02_server.script
  0         dp_sjf_03.script
  0         dp_sjf_tesla.script
  12520     media_transfer.o
  18848     parser.o
```

```
04/30/21 22:00:47.867123875 UTC: File Listing Received
04/30/21 21:59:40.469087843 UTC: TX: get r.mp3
04/30/21 21:59:40.469087843 UTC: Sent Command: get r.mp3
```

```
^C
[daxpate@in-csci-rrpc03 Project-4]$ ./client in-csci-rrpc02:1334
h_length = 4
Connected: server's address is in-csci-rrpc02
04/30/21 22:05:42.993658304 UTC: TX: get r.mp3
04/30/21 22:05:42.993658304 UTC: Sent Command: get r.mp3
```

```
^C
[daxpate@in-csci-rrpc03 Project-4]$ exit
exit
```

```
Script done on Fri 30 Apr 2021 06:07:24 PM EDT
```

## FIFO Traces

```
Script started on Thu 08 Apr 2021 07:20:17 PM EDT
[daxpate@in-csci-rrpc01 Project-3]$ ./server
*****Server configuration*****
Port Number: 3000
Num Threads: 3
Max Regs: 10
Media Path: /home/daxpate/ece40800/Project-3
*****
```

```
04/08/21 23:20:31.640086852 UTC: Main: Accepting New Connection: 5
04/08/21 23:20:31.640086852 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:31.640221024 UTC: Main: Added New Client to the Job queue
04/08/21 23:20:31.640221126 UTC: Watch Request: Thead 139875339474688: Handling client 5
04/08/21 23:20:31.640265601 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 3000

04/08/21 23:20:34.612238389 UTC: Main: Accepting New Connection: 6
04/08/21 23:20:34.612238389 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:34.612259037 UTC: Main: Added New Client to the Job queue
04/08/21 23:20:34.612260051 UTC: Watch Request: Thead 139875331081984: Handling client 6
04/08/21 23:20:34.612290091 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 3000
04/08/21 23:20:38.175981635 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 3000 :
Command Recevied string: list

04/08/21 23:20:39.776127279 UTC: Main: Accepting New Connection: 7
04/08/21 23:20:39.776127279 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:39.776150170 UTC: Main: Added New Client to the Job queue
04/08/21 23:20:39.776149538 UTC: Watch Request: Thead 139875347867392: Handling client 7
04/08/21 23:20:39.776178395 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 3000

04/08/21 23:20:40.640938779 UTC: Main: Accepting New Connection: 8
04/08/21 23:20:40.640938779 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:40.640976789 UTC: Main: Added New Client to the Job queue

04/08/21 23:20:41.719012898 UTC: Main: Accepting New Connection: 9
04/08/21 23:20:41.719012898 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:41.719051702 UTC: Main: Added New Client to the Job queue
04/08/21 23:20:47.852451302 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:20:47.852451302 UTC Ephemeral Port: 3000 : Command Recevied string: get
r.mp3
04/08/21 23:20:47.854130804 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:20:47.854130804 UTC Ephemeral Port: 3000 : Sent Header Information
04/08/21 23:20:57.884379411 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:20:57.884379411 UTC Ephemeral Port: 3000 : Sent: get r.mp3

04/08/21 23:20:58.213331980 UTC: Main: Accepting New Connection: 10
04/08/21 23:20:58.213331980 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:20:58.213376001 UTC: Main: Added New Client to the Job queue
04/08/21 23:21:04.339736340 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 3000 :
Command Recevied string: list

04/08/21 23:21:13.756194070 UTC: Main: Accepting New Connection: 11
04/08/21 23:21:13.756194070 UTC: Main: Adding New Client to the Job queue...
04/08/21 23:21:13.756236799 UTC: Main: Added New Client to the Job queue
04/08/21 23:21:36.766463243 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:21:36.766463243 UTC Ephemeral Port: 3000 : Command Recevied string: get
song.mp3
04/08/21 23:21:36.768571274 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:21:36.768571274 UTC Ephemeral Port: 3000 : Sent Header Information
04/08/21 23:21:46.929214149 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:21:46.929214149 UTC Ephemeral Port: 3000 : Sent: get song.mp3
04/08/21 23:21:49.283459847 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 3000 :
Command Recevied string: list
04/08/21 23:21:52.454196108 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
25928 server.o
4321604/08/21 23:21:52.454196108 UTC Ephemeral Port: 3000 : Command Recevied string: exit
04/08/21 23:21:52.454321366 UTC: Handle_Request: Client IP: 13521 server-fifo
166 mserver.config
```

```
25928          server.o
4321604/08/21 23:21:52.454321366 UTC Ephemeral Port: 3000 : Closed connection with client:
6
04/08/21 23:21:52.454361655 UTC: Watch Request: Thead 139875331081984: Handling client 8
04/08/21 23:21:52.454377907 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 3000
04/08/21 23:21:52.454396883 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 3000 :
Command Recevied string: list
04/08/21 23:21:52.456726217 UTC: Handle_Request: Client IP:      13521          server-fifo
166          mserver.config
25928          server.o
[daxpate@in-csci-rrpc01 Project-3]$ exit
exit
```

Script done on Thu 08 Apr 2021 07:22:22 PM EDT

Script started on 2021-04-08 17:50:35-04:00 [TERM="xterm-256color" TTY="/dev/pts/3" COLUMNS="195" LINES="47"]

```
pi@raspberrypi:~/repos/ece40800/Project-3 $ make
gcc -g -c -o media_transfer.o media_transfer.c
gcc -g -c media_transfer.c
gcc -g -c -o parser.o parser.c
gcc -g -c parser.c
gcc -g -c -o server.o server.c
gcc -g -o server server.o media_transfer.o parser.o -lm -lnsl -lpthread
gcc -g -c -o client.o client.c
gcc -g -o client client.o media_transfer.o parser.o -lm -lnsl
```

```
pi@raspberrypi:~/repos/ece40800/Project-3 $ ./server
```

\*\*\*\*\*Server configuration\*\*\*\*\*

Port Number: 1234

Num Threads: 4

Max Regs: 10

Media Path: /home/pi/repos/ece40800/Project-3

\*\*\*\*\*

```
04/08/21 21:51:05.224235906 UTC: Main: Accepting New Connection: 5
04/08/21 21:51:05.224235906 UTC: Main: Adding New Client to the Job queue...
04/08/21 21:51:05.228580789 UTC: Watch Request: Thread 3066995808: Handling client 5
04/08/21 21:51:05.228665732 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234
04/08/21 21:51:05.228577159 UTC: Main: Added New Client to the Job queue
04/08/21 21:51:09.244611698 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: list
```

```
04/08/21 21:52:11.181261627 UTC: Main: Accepting New Connection: 6
04/08/21 21:52:11.181261627 UTC: Main: Adding New Client to the Job queue...
04/08/21 21:52:11.181328792 UTC: Main: Added New Client to the Job queue
04/08/21 21:52:11.181336551 UTC: Watch Request: Thread 3058603104: Handling client 6
04/08/21 21:52:11.181363477 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234
04/08/21 21:52:12.687719411 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: list
04/08/21 21:52:16.919490219 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: get song.mp3
04/08/21 21:52:16.919685789 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Sent Header Information
04/08/21 21:52:26.827175881 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Sent: get song.mp3
04/08/21 21:52:31.600172260 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: get r.mp3
04/08/21 21:52:31.600406108 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Sent Header Information
04/08/21 21:52:36.766682435 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Sent: get r.mp3
04/08/21 21:52:38.351151265 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: exit
04/08/21 21:52:38.351313521 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Closed connection with client: 6
04/08/21 21:52:40.313385603 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Command Received string: exit
04/08/21 21:52:40.313656449 UTC: Handle Request: Client IP: 192.168.0.124 Ephemeral Port: 1234 : Closed connection with client: 5
```

^C

```
pi@raspberrypi:~/repos/ece40800/Project-3 $ ls
ansi2html.sh  client  client.o  'get receive.mp3'  media_transfer.c  media_transfer.o  old.c  parser.h  queue.h  r.mp3  server  server.o  typescript
a.out        client.c  clientrc  Makefile          media_transfer.h  mservr.config    parser.c  parser.o  received.mp3  send.txt  server.c  song.mp3
pi@raspberrypi:~/repos/ece40800/Project-3 $ ans2
```

```

/* A simple echo server using TCP */
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <pthread.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <poll.h>
#include <unistd.h>

#include "parser.h"
#include "media_transfer.h"
#include "queue.h"

#define SERVER_TCP_PORT      3000      /* well-known port */
#define HEADER_LEN          256        /* header packet length */
#define THREADS              4

typedef struct handler_arg {
    int port;
    int client_socket;
} handler_arg_t;

typedef struct {
    int num_threads;                // no.of threads - can be modified using CL Args number 3
    pthread_t *handlers;           // array of threads

    int max_requests;              // max no.of client requests server can have at any time -
    can be modified using CL Args number 4
    Queue *job_queue;              // data structure to hold server reqs

    char * directory;              // place to look for media files
} server_config_t;

/*
 * @param sockfd - client socket for hadlign request
 */
void *handle_request(void *arg);
void *watch_requests(void *config);

// Add a new file descriptor to the set
void add_to_pfds(struct pollfd *pfds[], int newfd, int *fd_count, int *fd_size)
{
    // If we don't have room, add more space in the pfds array
    if (*fd_count == *fd_size) {
        *fd_size *= 2; // Double it

        *pfds = realloc(*pfds, sizeof(**pfds) * (*fd_size));
    }

    (*pfds)[*fd_count].fd = newfd;
    (*pfds)[*fd_count].events = POLLIN; // Check ready-to-read

    (*fd_count)++;
}

// Remove an index from the set
void del_from_pfds(struct pollfd pfds[], int i, int *fd_count)
{
    // Copy the one from the end over this one
    pfds[i] = pfds[*fd_count-1];

    (*fd_count)--;
}

/*
 * @param filepath - name of the file for which extension is needed
 * @returns
 *      point to first char in extension
 */
const char *get_file_ext(const char *filename);

pthread_mutex_t lock;

```

```

pthread_mutex_t th_lock;
pthread_cond_t th_cond;

void setup_handlers(server_config_t *config) {
    if(pthread_mutex_init(&th_lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
        return;
    }

    for (int i = 0; i < config->num_threads; ++i) {
        if(pthread_create(&(config->handlers[i]), NULL, watch_requests, (void*)config) !=
0) {
            printf("Failed to create a thread");
            exit(1);
        }
    }
}

int main(int argc, char **argv)
{
    int n, bytes_to_read;
    int sd, new_sd, port;
    socklen_t client_len;
    struct sockaddr_in server, client;
    char *bp, buf[BUFLEN];
    int num_threads = 1;
    int max_req = 1;
    char *dir = ".";
    switch(argc) {
        case 1:
            port = SERVER_TCP_PORT;
            break;
        case 2:
            port = atoi(argv[1]);
            break;
        case 3:
            port = atoi(argv[1]);
            num_threads = atoi(argv[2]);
            max_req = atoi(argv[3]);
        case 4:
            port = atoi(argv[1]);
            num_threads = atoi(argv[2]);
            max_req = atoi(argv[3]);
            break;
        case 5:
            port = atoi(argv[1]);
            num_threads = atoi(argv[2]);
            max_req = atoi(argv[3]); //in future, sched type will be set here
            break;
        case 6:
            port = atoi(argv[1]);
            num_threads = atoi(argv[2]);
            max_req = atoi(argv[3]);
            dir = argv[5];
            break;
        default:
            fprintf(stderr, "Usage: %s [port]\n", argv[0]);
            exit(1);
    }

    // allocate num threads asked for
    int ret = chdir(dir);
    if(ret != 0) {
        printf("did not find the direcorey specidified: %s", dir);
        exit(1);
    }
    char pwd[BUFLEN];
    getcwd(pwd, BUFLEN);
    pthread_t threads[num_threads];
    server_config_t config;
    config.handlers = threads;
    config.num_threads = num_threads;
    config.max_requests = max_req;
    config.job_queue = createQueue(max_req);
    config.directory = pwd;

    printf("Server config set to:\n");
    printf("Num Threads: %d\n", config.num_threads);
    printf("Max Reqs: %d\n", config.max_requests);
    printf("Media Path: %s/ \n", config.directory);
    fflush(stdout);
}

```

```

setup_handlers(&config);

/* Create a stream socket */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    fprintf(stderr, "Can't create a socket\n");
    exit(1);
}

if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
    perror("setsockopt(SO_REUSEADDR)");

if (setsockopt(sd, SOL_SOCKET, SO_KEEPALIVE, &(int){1}, sizeof(int)) < 0)
    perror("setsockopt(SO_KEEPALIVE)");

/* Bind an address to the socket */
bzero((char *)&server, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&server,
sizeof(server)) == -1) {
    fprintf(stderr, "Can't bind name to socket\n");
    exit(1);
}

/* queue up to 5 connect requests */
listen(sd, 5);

int fd_count = 0;
int fd_size = 5;
struct pollfd *pfd = malloc(sizeof *pfd * fd_size);

// Add the listener to set
pfd[0].fd = sd;
pfd[0].events = POLLIN; // Report ready to read on incoming connection

fd_count = 1;

for (;;) {
    int poll_count = poll(pfd, fd_count, -1);

    if (poll_count == -1) {
        perror("poll");
        exit(1);
    }

    // Run through the existing connections looking for data to read
    for(int i = 0; i < fd_count; i++) {
        // Check if someone's ready to read
        if (pfd[i].revents & POLLIN) { // We got one!!

            if (pfd[i].fd == sd) {
                // If listener is ready to read, handle new connection

                client_len = sizeof client;
                new_sd = accept(sd,
                    (struct sockaddr *)&client,
                    &client_len);

                if (new_sd == -1) {
                    perror("accept");
                } else {
                    add_to_pfd(&pfd, new_sd, &fd_count, &fd_size);

                    printf("server: new connection\n");
                }
            } else {
                // If not the listener, we're just a regular client
                int sender_fd = pfd[i].fd;

                int error = recv(sender_fd, NULL, 1, MSG_PEEK |
MSG_DONTWAIT);

                if (error == 0) {
                    printf("socket hungup\n");
                    close(pfd[i].fd); // Bye!

                    del_from_pfd(pfd, i, &fd_count);
                } else {

```



```

(hanlder_arg_t*)malloc(sizeof(hanlder_arg_t));
hanlder_arg_t *arg =
arg->port = port;
arg->client_socket = sender_fd;

enqueue(config.job_queue, (void*) arg);
pthread_mutex_lock(&th_lock);
pthread_cond_broadcast(&th_cond);
pthread_mutex_unlock(&th_lock);
    }
}
}

pthread_mutex_destroy(&lock);
return 0;
}

void *watch_requests(void *arg) {
server_config_t *config = (server_config_t*)arg;
void *job = NULL;
for (;;) {
pthread_mutex_lock(&th_lock);

pthread_mutex_lock(&lock);

if (!isEmpty(config->job_queue)) {
job = dequeue(config->job_queue);
}

pthread_mutex_unlock(&lock);

if (job != NULL) {
hanlder_arg_t* info = ((hanlder_arg_t*)job);
// printf("%d\n", info->client_socket);
handle_request(job);
}

job = NULL;

pthread_cond_wait(&th_cond, &th_lock);
pthread_mutex_unlock(&th_lock);
}
}

void *handle_request(void *client_sd)
{
char buf[BUFLen] = {0};
hanlder_arg_t* info = ((hanlder_arg_t*)client_sd);

char *bp = buf;
int bytes_to_read = BUFLen;
int n = 0;

while ((n = read(info->client_socket, bp, bytes_to_read)) > 0) {
bp += n;
bytes_to_read -= n;
}

if (bp <= 0) {
// client probably disconnected
close(info->client_socket);
}

int size = strlen(buf);
printf("%d \n", size);
printf("RCVD: %s", buf);
printf("client socket: %d\n", info->client_socket);
buf[size - 1] = '\0';

switch(get_command_from_request(buf)) {
case LIST:{
char listing[1024];
get_media_list(".", listing, 1024);
// send the header packet
send_header(info->client_socket, info->port, strlen(listing), "Text", 100);
if(send(info->client_socket, listing, strlen(listing), 0) == -1) {
printf("error sending list\n");
}
}
}
}

```

```

        break;
    }
    case GET: {
        // get the length of the file needed to be read.
        FILE *fp = fopen(&(buf[4]), "rb");

        if (fp == NULL) {
            send_header(info->client_socket, info->port, 0, "", 404);
            break;
        }

        fseek(fp, 0L, SEEK_END);
        size_t len = ftell(fp);
        fseek(fp, 0L, SEEK_SET);
        fclose(fp);

        // get file extension
        const char *extension = get_file_ext(buf + 4);

        // send header information
        send_header(info->client_socket, info->port, len, extension, 100);

        // send requested media
        send_media(info->client_socket, buf + 4, len);

        printf("SEND: %s\n", buf);
        break;
    }
    default:
        send_header(info->client_socket, info->port, 0, "", 301);
        break;
}

}

const char *get_file_ext(const char *filename) {
    const char *dot_loc = strrchr(filename, '.');
    if(!dot_loc || dot_loc == filename) {
        return "Unknown";
    }
    return dot_loc + 1;
}

```

```

Script started on Thu 08 Apr 2021 06:47:27 PM EDT
[pakpatel@in-csci-rrpc04 Project-3]$ ./client in-csci-rrpc01:3000
h_length = 4
Connected: server's address is in-csci-rrpc01
04/08/21 22:50:23.290318008 UTC: TX: list
04/08/21 22:50:23.290318008 UTC: Sent Command: list

04/08/21 22:50:53.022564103 UTC: Header Response Received
04/08/21 22:50:53.022638146 UTC: Status:100 Host:10.234.136.55:3000 Length:1196 Type:Text
Size      Name
4096      .
50        ..
4096      .git
473       .gitignore
39        clientrc
1125      media_transfer.h
113       send.txt
9733309   song.mp3
445       Makefile
2353      client.c
2706      media_transfer.c
8911      old.c
2615      parser.h
1961      queue.h
9733309   r.mp3
9733309   received.mp3
9114      parser.c
17355     ansi2html.sh
20596     client.html
34304     client.script
9733309   get_receive.mp3
22157     server-exe.html
8192      server.script
9733309   test.mp3
12479     server.c
5352      dax-tesla.script
9733309   receive-dax.mp3
194       typescript
9097      server-load.script
0         dax-tesla-2.script
12479     server-copy.c
20596     caleb-client.html
168       caleb-mserver.config
3956      caleb-rrpc03.script
43216     caleb-server
9733309   caleb-song.mp3
20596     caleb-tesla-client.html
9733309   caleb-tesla.mp3
6991      caleb-tesla.script
9733309   caleb-test.mp3
8911      old-parth.c
2178      parth-rrpc
1927      parth-rrpc2
113       parth-send.txt
3494      parth-tesla
2550      parth-tesla2
9733309   parth-test.mp3
9733309   receive-parth.mp3
9733309   song-parth123.mp3
13521     server-fifo
166       mserver.config
25928     server.o
43216     server
13312     client.o
35728     client
12520     media_transfer.o
18848     parser.o
0         server-fifo-2.script

04/08/21 22:50:53.030003127 UTC: File Listing Received
04/08/21 22:50:23.290318008 UTC: TX: get song.mp3
04/08/21 22:50:23.290318008 UTC: Sent Command: get song.mp3

04/08/21 22:51:25.446816265 UTC: Header Response Received
04/08/21 22:51:25.446851033 UTC: Status:100 Host:10.234.136.55:3000 Length:9733309 Type:mp3
04/08/21 22:51:25.446851033 UTC: Name of the file to put data received from server to: parth-song-fifo.mp3
04/08/21 22:51:36.112922662 UTC: Media Received and Downloaded
04/08/21 22:50:23.290318008 UTC: TX: exit
[pakpatel@in-csci-rrpc04 Project-3]$ exit

```

```
exit
```

```
Script done on Thu 08 Apr 2021 06:52:03 PM EDT
```

Script started on 2021-04-08 17:50:51-04:00 [TERM="xterm-256color" TTY="/dev/tty1" COLUMNS="148" LINES="32"]

Project-3 git:(exe-traces) ./client 192.168.0.124:1234

h\_length = 4

Connected: server's address is 192.168.0.124

04/08/21 21:51:05.322832700 UTC: TX: list

04/08/21 21:51:05.322832700 UTC: Sent Command: list

04/08/21 21:51:09.347689600 UTC: Header Response Received

04/08/21 21:51:09.347758100 UTC: Status:100 Host:127.0.1.1:1234 Length:494 Type:Text

Size	Name
473	.gitignore
0	typescript
8196	a.out
9733309	received.mp3
4096	.git
10716	client.o
113	send.txt
2706	media_transfer.c
4096	.
1125	media_transfer.h
2353	client.c
39	clientrc
4096	..
21128	server.o
17355	ansi2html.sh
47200	server
12479	server.c
2615	parser.h
8911	old.c
34432	client
9114	parser.c
167	mserver.config
10332	media_transfer.o
9733309	r.mp3
1961	queue.h
9733309	song.mp3
4096	.vscode
15996	parser.o
445	Makefile

04/08/21 21:51:09.392414800 UTC: File Listing Received

04/08/21 21:51:05.322832700 UTC: TX: get song.mp3

04/08/21 21:51:05.322832700 UTC: Sent Command: get song.mp3

04/08/21 21:52:17.021055600 UTC: Header Response Received

04/08/21 21:52:17.021092900 UTC: Status:100 Host:127.0.1.1:1234 Length:9733309 Type:mp3

04/08/21 21:52:17.021092900 UTC: Name of the file to put data received from server to: test.mp3

04/08/21 21:52:27.074399900 UTC: Media Received and Downloaded

04/08/21 21:51:05.322832700 UTC: TX: exit

Project-3 git:(exe-traces) mv typescript client.script

Random Sched Traces

```
[daxpate@in-csci-rrpc01 Project-3]$ script server-load.script
```

```
Script started, file is server-load.script
```

```
[daxpate@in-csci-rrpc01 Project-3]$
```

```
[daxpate@in-csci-rrpc01 Project-3]$ ./server
```

```
*****Server configuration*****
```

```
Port Number: 1234
```

```
Num Threads: 3
```

```
Max Reqs: 10
```

```
Media Path: /home/daxpate/ece40800/Project-3
```

```
*****
```

```
04/08/21 22:58:59.110122973 UTC: Main: Accepting New Connection: 5
```

```
04/08/21 22:58:59.110122973 UTC: Main: Adding New Client to the Job queue...
```

```
04/08/21 22:58:59.110227938 UTC: Main: Added New Client to the Job queue
```

```
04/08/21 22:58:59.110228632 UTC: Watch Request: Thead 139803194681088: Handling client 5
```

```
04/08/21 22:58:59.110255459 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 1234
```

```
04/08/21 22:59:01.695403789 UTC: Main: Accepting New Connection: 6
```

```
04/08/21 22:59:01.695403789 UTC: Main: Adding New Client to the Job queue...
```

```
04/08/21 22:59:01.695424819 UTC: Main: Added New Client to the Job queue
```

```
04/08/21 22:59:01.695423971 UTC: Watch Request: Thead 139803211466496: Handling client 6
```

```
04/08/21 22:59:01.695450680 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 1234
```

```
04/08/21 22:59:04.613565699 UTC: Main: Accepting New Connection: 7
```

```
04/08/21 22:59:04.613565699 UTC: Main: Adding New Client to the Job queue...
```

```
04/08/21 22:59:04.613588709 UTC: Main: Added New Client to the Job queue
```

```
04/08/21 22:59:04.613589976 UTC: Watch Request: Thead 139803203073792: Handling client 7
```

```
04/08/21 22:59:04.613623407 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 1234
```

```
04/08/21 22:59:05.552166743 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: list
```

```
04/08/21 22:59:08.546578521 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: list
```

```
04/08/21 22:59:11.306005918 UTC: Main: Accepting New Connection: 8
```

```
04/08/21 22:59:11.306005918 UTC: Main: Adding New Client to the Job queue...
```

```
04/08/21 22:59:11.306043784 UTC: Main: Added New Client to the Job queue
```

```
04/08/21 22:59:13.402649233 UTC: Main: Accepting New Connection: 9
```

```
04/08/21 22:59:13.402649233 UTC: Main: Adding New Client to the Job queue...
```

```
04/08/21 22:59:13.402685486 UTC: Main: Added New Client to the Job queue
```

```
04/08/21 22:59:14.873625625 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: get test.mp3
```

```
04/08/21 22:59:14.876993890 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent Header Information
```

```
04/08/21 22:59:19.427418989 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent: get test.mp3
```

```
04/08/21 22:59:26.053774100 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: get mserver.config
```

```
04/08/21 22:59:26.055197809 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent Header Information
```

```
04/08/21 22:59:26.055366579 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent: get mserver.config
```

```
04/08/21 22:59:29.736352327 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: list
```

```
04/08/21 22:59:39.421926201 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: get server.c
```

```
04/08/21 22:59:39.425904323 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent Header Information
```

```
04/08/21 22:59:39.427146236 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent: get server.c
```

```
04/08/21 22:59:41.607423644 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: get server
```

```
04/08/21 22:59:41.608999194 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent Header Information
```

```
04/08/21 22:59:41.609245782 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent: get server
```

```
04/08/21 22:59:46.570303654 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: list
```

```
04/08/21 22:59:49.725497193 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: exit
```

```
04/08/21 22:59:49.725558550 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Closed connection with client: 7
```

```
04/08/21 22:59:49.725577533 UTC: Watch Request: Thead 139803203073792: Handling client 8
```

```
04/08/21 22:59:49.725593795 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 1234
```

```
04/08/21 22:59:49.725621605 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: list
```

```
04/08/21 22:59:54.867078018 UTC: Handle Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Command Recevied string: get client.html
```

```
04/08/21 22:59:54.870120226 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :
```

```
Sent Header Information
```

```
04/08/21 22:59:54.870384944 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get client.html  
04/08/21 23:00:02.102061001 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: get send.txt  
04/08/21 23:00:02.105633550 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent Header Information  
04/08/21 23:00:02.106337243 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get send.txt  
04/08/21 23:00:07.115006958 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: list  
04/08/21 23:00:12.831200944 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: exit  
04/08/21 23:00:12.831258081 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Closed connection with client: 8  
04/08/21 23:00:12.831274178 UTC: Watch Request: Thead 139803203073792: Handling client 9  
04/08/21 23:00:12.831321460 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
04/08/21 23:00:12.831340957 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: list  
04/08/21 23:00:13.370999540 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: list  
04/08/21 23:00:14.203767607 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: list  
04/08/21 23:00:17.461355339 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: list  
04/08/21 23:00:26.618958765 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: get test.mp3  
04/08/21 23:00:26.620562047 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent Header Information  
04/08/21 23:00:31.196703986 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: get song.mp3  
04/08/21 23:00:31.199712802 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent Header Information  
04/08/21 23:00:34.469675185 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: get song.mp3  
04/08/21 23:00:34.471211582 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent Header Information  
04/08/21 23:00:40.016987833 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get test.mp3  
04/08/21 23:01:00.954741987 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: get song.mp3  
04/08/21 23:01:00.956951114 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent Header Information  
04/08/21 23:01:11.119970980 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get song.mp3  
04/08/21 23:01:42.662720004 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get song.mp3  
04/08/21 23:01:43.022612346 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Sent: get song.mp3  
04/08/21 23:01:48.257015878 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: exit  
04/08/21 23:01:48.257076075 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Closed connection with client: 9  
04/08/21 23:02:09.298075087 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: exit  
04/08/21 23:02:09.298114272 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Closed connection with client: 5  
04/08/21 23:02:12.003400642 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Command Received string: exit  
04/08/21 23:02:12.003425265 UTC: Handle_Request: Client IP: 10.234.136.55 Ephemeral Port: 1234 :  
Closed connection with client: 6  
^C  
[daxpate@in-csci-rrpc01 Project-3]$ exit  
exit  
Script done, file is server-load.script  
[daxpate@in-csci-rrpc01 Project-3]$ git status
```



```

Script started on Thu 08 Apr 2021 06:17:06 PM EDT
[pakpatel@in-csci-rrpc04 Project-3]$ ./client in-csci-rrpc01:1234
h_length = 4
Connected: server's address is in-csci-rrpc01
04/08/21 22:17:41.439841515 UTC: TX: list
04/08/21 22:17:41.439841515 UTC: Sent Command: list

04/08/21 22:22:10.130516495 UTC: Header Response Received
04/08/21 22:22:10.130577486 UTC: Status:100 Host:10.234.136.55:1234 Length:602 Type:Text
    Size      Name
    4096      .
    50        ..
    4096      .git
    473       .gitignore
    39        clientrc
    1125      media_transfer.h
    113       send.txt
    9733309   song.mp3
    445       Makefile
    2353      client.c
    2706      media_transfer.c
    8911      old.c
    2615      parser.h
    1961      queue.h
    9733309   r.mp3
    9733309   received.mp3
    9114      parser.c
    17355     ansi2html.sh
    20596     client.html
    34304     client.script
    9733309   get_receive.mp3
    22157     server-exe.html
    8192      server.script
    9733309   test.mp3
    12520     media_transfer.o
    18848     parser.o
    25928     server.o
    43216     server
    13312     client.o
    35728     client
    168       mserver.config
    12479     server.c
    0         dax-tesla.script

04/08/21 22:22:10.132477897 UTC: File Listing Received
04/08/21 22:17:41.439841515 UTC: TX: get r.mp3
04/08/21 22:17:41.439841515 UTC: Sent Command: get r.mp3

04/08/21 22:22:56.504876181 UTC: Header Response Received
04/08/21 22:22:56.504908514 UTC: Status:100 Host:10.234.136.55:1234 Length:9733309 Type:mp3
04/08/21 22:22:56.504908514 UTC: Name of the file to put data received from server to: receive-
parth.mp3
04/08/21 22:23:22.304170322 UTC: Media Received and Downloaded
04/08/21 22:17:41.439841515 UTC: TX: exit
[pakpatel@in-csci-rrpc04 Project-3]$ ./client in-csci-rrpc01:1234
h_length = 4
Connected: server's address is in-csci-rrpc01
04/08/21 22:25:17.481820634 UTC: TX: exit
[pakpatel@in-csci-rrpc04 Project-3]$ stop
bash: stop: command not found...
Similar command is: 'top'
[pakpatel@in-csci-rrpc04 Project-3]$ exit
exit

Script done on Thu 08 Apr 2021 06:27:32 PM EDT

```

```

Script started on Thu 08 Apr 2021 06:28:36 PM EDT
[kirbycm@in-csci-rrpc03 Project-3]$ ./client in-csci-rrpc01:1234
h_length = 4
Connected: server's address is in-csci-rrpc01
04/08/21 22:28:50.369668596 UTC: TX: list
04/08/21 22:28:50.369668596 UTC: Sent Command: list

04/08/21 22:28:57.223193773 UTC: Header Response Received
04/08/21 22:28:57.223245403 UTC: Status:100 Host:10.234.136.55:1234 Length:694 Type:Text
Size      Name
4096      .
50        ..
4096      .git
473       .gitignore
39        clientrc
1125      media_transfer.h
113       send.txt
9733309   song.mp3
445       Makefile
2353      client.c
2706      media_transfer.c
8911      old.c
2615      parser.h
1961      queue.h
9733309   r.mp3
9733309   received.mp3
9114      parser.c
17355     ansi2html.sh
20596     client.html
34304     client.script
9733309   get_receive.mp3
22157     server-exe.html
8192      server.script
9733309   test.mp3
12520     media_transfer.o
18848     parser.o
25928     server.o
43216     server
13312     client.o
35728     client
168       mserver.config
12479     server.c
5352      dax-tesla.script
9733309   receive-dax.mp3
194       typescript
0         server-load.script
0         dax-tesla-2.script

04/08/21 22:28:57.225295958 UTC: File Listing Received
04/08/21 22:28:50.369668596 UTC: TX: get mserver.config
04/08/21 22:28:50.369668596 UTC: Sent Command: get mserver.config

04/08/21 22:29:14.729834567 UTC: Header Response Received
04/08/21 22:29:14.729867361 UTC: Status:100 Host:10.234.136.55:1234 Length:168 Type:config
04/08/21 22:29:14.729867361 UTC: Name of the file to put data received from server to: caleb-
mserver.config
04/08/21 22:29:20.777601080 UTC: Media Received and Downloaded
04/08/21 22:28:50.369668596 UTC: TX: get server
04/08/21 22:28:50.369668596 UTC: Sent Command: get server

04/08/21 22:29:30.283687258 UTC: Header Response Received
04/08/21 22:29:30.283749220 UTC: Status:100 Host:10.234.136.55:1234 Length:43216 Type:Unknown
04/08/21 22:29:30.283749220 UTC: Name of the file to put data received from server to: caleb-server
04/08/21 22:29:32.562844448 UTC: Media Received and Downloaded
04/08/21 22:28:50.369668596 UTC: TX: list
04/08/21 22:28:50.369668596 UTC: Sent Command: list

04/08/21 22:30:06.138348621 UTC: Header Response Received
04/08/21 22:30:06.138382598 UTC: Status:100 Host:10.234.136.55:1234 Length:716 Type:Text
Size      Name
4096      .
50        ..
4096      .git
473       .gitignore
39        clientrc
1125      media_transfer.h
113       send.txt
9733309   song.mp3
445       Makefile
2353      client.c

```

```

2706      media_transfer.c
8911      old.c_
2615      parser.h
1961      queue.h
9733309   r.mp3
9733309   received.mp3
9114      parser.c
17355     ansi2html.sh
20596     client.html
34304     client.script
9733309   get_receive.mp3
22157     server-exe.html
8192      server.script
9733309   test.mp3
12520     media_transfer.o
18848     parser.o
25928     server.o
43216     server
13312     client.o
35728     client
168       mserver.config
12479     server.c
5352      dax-tesla.script
9733309   receive-dax.mp3
194       typescript
0         server-load.script
0         dax-tesla-2.script
12479     server-copy.c

```

04/08/21 22:30:06.140622295 UTC: File Listing Received

04/08/21 22:28:50.369668596 UTC: TX: get song.mp3

04/08/21 22:28:50.369668596 UTC: Sent Command: get song.mp3

04/08/21 22:30:19.874155579 UTC: Header Response Received

04/08/21 22:30:19.874186833 UTC: Status:100 Host:10.234.136.55:1234 Length:9733309 Type:mp3

04/08/21 22:30:19.874186833 UTC: Name of the file to put data received from server to: caleb-song.mp3

04/08/21 22:31:32.267920644 UTC: Media Received and Downloaded

04/08/21 22:28:50.369668596 UTC: TX: exit

[kirbycm@in-csci-rrpc03 Project-3]\$ exit

exit

Script done on Thu 08 Apr 2021 06:32:27 PM EDT