Yang J. Ren

Daniel Su

NCP - Final Project

Introduction:

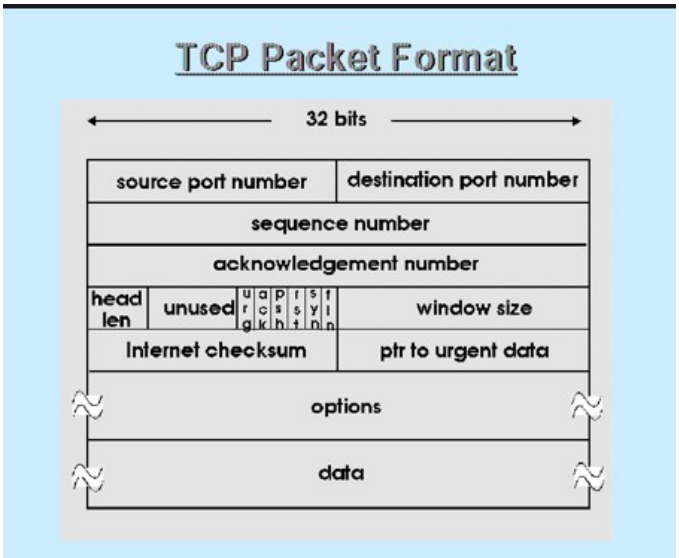
Our project was to create a server that would provide live streaming video to a HTTP client. We accomplished this by listening for an HTTP request, then responding to the request by reading a video file stream that is generated by FFMPEG.

Initiating the Connection:

The server first does the standard socket create, bind, and listen. We do this on port 8000. Then we loop and wait for an HTTP GET request to be initiated. This GET request is generated by the client web browser when attempting the initial connection to our server. The server responds with the server description of the file it is sending, and then responds with the index.html. This index.html contains an embedded tag that links to the web stream on our server. When the client browser sees this tag, it sends another request for the video stream based on the location referred to in the video tag. Since the handling of html5 video differs between clients, to ensure that the video stream loaded correctly we had to use Firefox to stream the video. Chrome was tested and found to be unable to actually playback the video, and as such was unsuitable for our needs.

Protocol:

The HTTP requests and data are sent through the Transmission Control Protocol. The TCP packet contains the source port and the destination port.



These identify the sockets that are open at the respective locations. The other fields of the TCP packet are not handled manually by the server in our implementation. The implementation that we use processes the server and client destination addresses, and generates a pointer to them within the client file descriptor, and the server file descriptor. The TCP struct encapsulates the data that we send/receive. The only HTTP request that we service on the server is GET. We service HTTP GET requests by sending the initial response detailing the format of the following data and then sending data in the packets that follow. Ex.

Once the initial GET request is received, we then sent back a 200 OK reply in the data segment of the TCP Packet.

HTTP/1.0 200 OK

Content-type: text/html

Then we send index.html in next packet. Once the client receives index.html, the client will send another GET request for out.ogg, which is our webcam stream. Once we receive this packet, we proceed to initialize the webcam stream, and start the transfer.

Capturing Video:

For capturing the content we wish to stream, we decided to use FFMPEG. FFMPEG is initiated to capture a 640x480 video stream of the webcam device that is located on /dev/video0 . The output stream is stored as out.ogg. This out.ogg is the file that is being served to the client. We perform a delayed buffering to send the initial video data to the client. In addition, since the transmission speed is faster than the generation of the video stream, the data retrieved from the webcam is sent as soon as it becomes available.

User Interface:

Our system contains no user interface, however, the system writes to the output stream, the results of its read and write operations.

Concepts:

The system that was implemented has no instances in which the buffer may be overflowed, and due to that construction, the server is relatively secure. Our server took the concepts of HTTP requests learned from the first proxy server and applied those concepts towards the construction of the webcam streaming.

Appendix:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<fcntl.h>
#include<signal.h>
#include<unistd.h>
#define MAXBUF 8196

char * parse(char request[MAXBUF]);
void indexhtml(int clientfd);
void webstream(int clientfd);

int main(int argc, char** argv){

struct sockaddr_in sockserv,sockclient;
int socketfd,clientfd;
socklen_t clientsocklen;
char request[MAXBUF];
signal(SIGPIPE,SIG_IGN);
//make socket
socketfd = socket(AF_INET,SOCK_STREAM,0);
printf("Socket Created: %s\n",strerror(errno));
int opt=1;
bzero(&sockserv,sizeof(sockserv));
sockserv.sin_family = AF_INET;
sockserv.sin_addr.s_addr = INADDR_ANY;
sockserv.sin_port = htons(8001);

//Reusable Sockets
setsockopt(socketfd,SOL_SOCKET,SO_REUSEADDR,(const char *)&opt,sizeof(int));
//Bind socket
if(bind(socketfd,(struct sockaddr *)&sockserv,sizeof(sockserv))<0){
perror("Bind Error\n");
exit(1);
}
printf("Socket Bind: %s\n",strerror(errno));
//Listen for requests
if(listen(socketfd,10)<0){
perror("Listen error\n");
exit(1);
```

```
}
printf("Socket Listen: %s\n",strerror(errno));

// Loop to continuously Accept new streaming requests after the last one is closed.
while (1){
clientfd = accept(socketfd,(struct sockaddr*)&sockclient,&clientsocklen);

memset(request,0, MAXBUF);
read(clientfd,request,MAXBUF);
/*Do not handle any non GET requests*/
if(strncmp(request, "GET", 3)!=0){
close(clientfd);
continue;
}

char *address;
address = (char *)malloc(sizeof(parse(request)));
memset(address, 0, sizeof(address));
//parse request
address = parse(request);
//request address comparison
if ((!strncmp(address,"/index.html", strlen(address)))&&strlen(address)==11){
indexhtml(clientfd);
}
else if ((!strncmp(address, "/", strlen(address)))&&strlen(address)==1){
indexhtml(clientfd);
}
else if ((!strncmp(address, "/out.ogg", strlen(address)))&&strlen(address)==8){
webstream(clientfd);
}

close(clientfd);
}
return 0;
}

//parse the request for the target address
char * parse(char request[MAXBUF]){
char * target;
target = strtok (request," ");
target = strtok (NULL, " ");
return target;
}

//Handle index GET request
void indexhtml(int clientfd){
```

```
char ihtml[MAXBUF];
memset(ihtml, 0, MAXBUF);
strcpy(ihtml,"HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n\r\n");
send(clientfd, ihtml, strlen(ihtml), 0);
int filefd, filesize;
filefd = open("index.html", O_RDONLY);
memset(ihtml, 0, MAXBUF);
filesize = read(filefd, ihtml, MAXBUF);
send(clientfd, ihtml, strlen(ihtml), 0);
}

//Handle WebCam Get request
void webstream(int clientfd){
// Remove Last Streamed Video
remove("out.ogg");
char video[MAXBUF];
pid_t record;
int bytesread;
int bytessent;
int filesize;
int closecounter = 0;
char * argv[11];
int videostream;
// Send initial Video Descriptor
memset(video, 0, MAXBUF);
strcpy(video,"HTTP/1.0 200 OK\r\n\r\nConnection: close\r\nCache-control: public\r\nContent-
type: video/ogg\r\n\r\n\r\n");
send(clientfd, video, strlen(video), 0);
// Fork new Process to handle video Processing
record = fork();
if (record == 0){
argv[0]="./ffmpeg";
argv[1]= "-f";
argv[2]= "video4linux2";
argv[3]= "-r";
argv[4]="30";
argv[5]= "-s";
argv[6]="640x480";
argv[7]="-i";
argv[8]="/dev/video0";
argv[9]="out.ogg";
argv[10]=(char*)0;
//run video processing
execvp(argv[0],argv);
}
else{
```

```
//delay timer before feeding to live stream
sleep(5);
errno = 0;
//wait until video feed is ready
while((videostream = open("out.ogg",O_RDONLY))==-1){
;
}

sleep(1);
while(1){
//read from video feed continuously
bytesread = read(videostream,video,1024);
if(strcmp(strerror(errno), "Broken pipe")==0){
close(clientfd);
close(videostream);
break;
}
filesize = filesize + bytesread;
if(bytesread <= 0){
sleep(1);
//if video feed has not been sending new data for a while close the feed
closecounter++;
if (closecounter == 100){
close(clientfd);
close(videostream);
break;
}
}
else {
closecounter = 0;
//if client closed connection end the feed
if(strcmp(strerror(errno), "Broken pipe")==0){
close(clientfd);
close(videostream);
break;
}
errno = 0;
bytessent = send(clientfd, video,bytesread,0);
printf("Read %d bytes from the file:Total %i\n",bytesread,filesize);
printf("Data written = %d bytes: %s\n", bytessent,strerror(errno));
}
}
//after the client closes the connection kill the recording process
kill(record,SIGTERM);     } }s
```