**프로젝트 보고서**

**UNIX 프로그래밍 과제2**

**2분반**

**컴퓨터과학과**

**201411168 김동민**

**목차**

1. 프로젝트 개요
2. 팀 소개
3. 주제 및 개요
4. 기능
5. 프로젝트 계획서 및 요구사항 명세서
6. 프로젝트 계획서
7. 요구사항 명세서
8. 프로그램 설명 및 테스트 화면
9. 텍스트 전송 및 실행 로그
10. 파일 전송 및 실행 로그
11. 프로그램 종료
12. 소스코드
13. **프로젝트 개요**

팀 소개

팀명: 미소켓 (미디어소프트웨어 + 소켓 프로그래밍)

팀장: 김동민 (1인 프로젝트)

주제 및 개요

주제: 소켓을 이용하여 파일 및 텍스트를 통신한다.

개요: 소켓 프로그래밍을 이용하여 클라이언트와 서버가 파일, 텍스트를 주고받는다. 실행 로그는 로그파일에 저장된다.

기능

1. 소켓을 생성하여 클라이언트와 서버 간의 통신으로 터미널에 입력되는 텍스트를 송수신한다.
2. 소켓을 생성하여 클라이언트와 서버 간의 통신으로 파일(.txt , C source 파일 등등)을 송수신한다.
3. 프로세스 시스템 콜을 이용하여 클라이언트 측에서 파일 수신 전 후로 파일의 수신 여부를 확인한다.
4. 프로그램 실행 로그를 클라이언트와 서버에 각각 따로 프로그램 실행 시간과 함께 로그 파일(log.txt)에 저장한다.
5. 명령어를 통해 프로그램을 종료시킨다.
6. **프로젝트 계획서 및 요구사항 명세서**

프로젝트 계획서

1. 주요일정

|  |  |  |
| --- | --- | --- |
| **단계** | **일정** | **산출물 및 일정내용** |
| 주제 선정 및 설계 | 2019.11.26~2019.11.26 | 주제 선정 |
| 구현 및 개발 | 2019.11.27~2019.12.01 | C source file, 실행파일 및 텍스트 파일 |
| 산출물 및 발표자료 준비 | 2019.12.02~2019.12.08 | 프로젝트 보고서, 발표 PPT |
| 발표 및 demo | 2019.12.09 | 발표 및 demo |

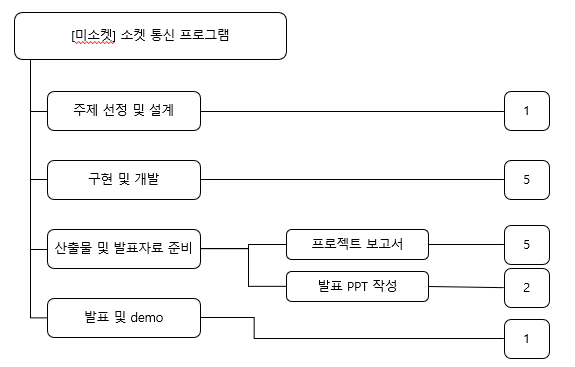
1. 역할 및 책임

|  |  |  |
| --- | --- | --- |
| **팀원** | **역할** | **책임** |
| 김동민 | PM | 주제 선정, 구현 및 개발, 산출물 작성, 발표 및 demo |

1. 도구

|  |  |
| --- | --- |
| **도구** | **설명** |
| Oracle VM Virtual Box (Ubuntu) | 컴퓨팅 환경을 소프트웨어로 구현한 것으로써 컴퓨터를 에뮬레이션하는 소프트웨어다.  가상 머신 상에서 운영 체제나 응용 프로그램을 설치 및 실행할 수 있다 |
| Microsoft PowerPoint, Word | 각종 문서들을 작성하기 위한 도구로써 가독성을 높이고 효율적으로 작업할 수 있게 해주는 도구이다. 프로젝트 보고서, 발표 PPT 등에 이용하였다. |

1. WBS



1. Pert Chart



요구사항 명세서

1. 기능 요구사항

|  |  |  |
| --- | --- | --- |
| **구분** | **상세** | **비고** |
| 텍스트 송수신 | 소켓을 생성하여 클라이언트와 서버 간의 통신으로 터미널에 입력하는 텍스트를 송수신한다 |  |
| 파일 송수신 | 소켓을 생성하여 클라이언트와 서버 간의 통신으로 ‘d’ 명령을 통해 파일(.txt , C source 파일 등등)을 송수신한다 |  |
| 파일 수신확인 | 프로세스 시스템 콜을 이용하여 클라이언트 측에서 파일 수신 전 후로 파일의 수신 여부를 확인한다 |  |
| 실행 로그 | 프로그램 실행 시간과 로그를 클라이언트와 서버에 각각 따로 로그 파일(log.txt)에 저장한다 |  |
| 프로그램 종료 | ‘:quit’ 명령을 입력하여 프로그램을 종료 시킨다. |  |

1. 비기능 요구사항

|  |  |
| --- | --- |
| **요구항목** | **설명** |
| 신뢰성 | 오류 없이 정확한 정보를 계속해서 제공할 수 있어야 한다. 프로그램 실행 도중 오류가 발생한다면 오류의 정보에 대한 메시지를 출력해야 한다. |
| 응답 시간 | 사용자가 요청한 데이터에 대하여 짧은 응답시간을 보장한다. |

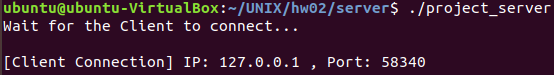
1. **프로그램 설명 및 테스트 화면**
2. 텍스트 전송 및 실행 로그



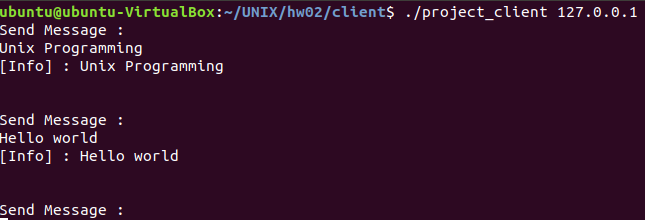
* 서버를 실행



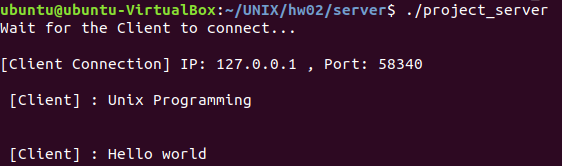
* 클라이언트를 로컬 IP로 실행한다.



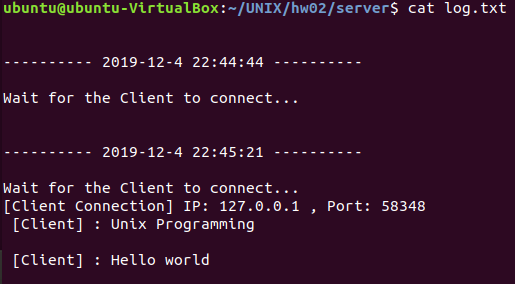
* 클라이언트가 접속되면 서버 측에는 위와 같은 커넥션 정보가 뜬다.



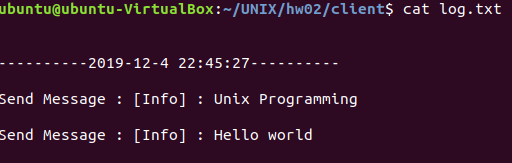
* 클라이언트에서 메시지를 입력한다.



* 서버 측에서는 클라이언트가 보낸 텍스트를 수신한다.

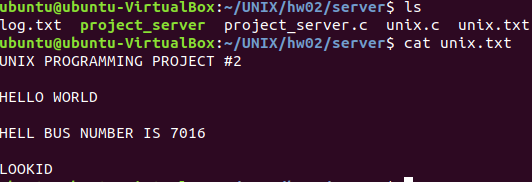


* 서버 측에 생긴 log.txt 를 cat으로 열어보면 실행 시간과 실행 로그가 남아있다.

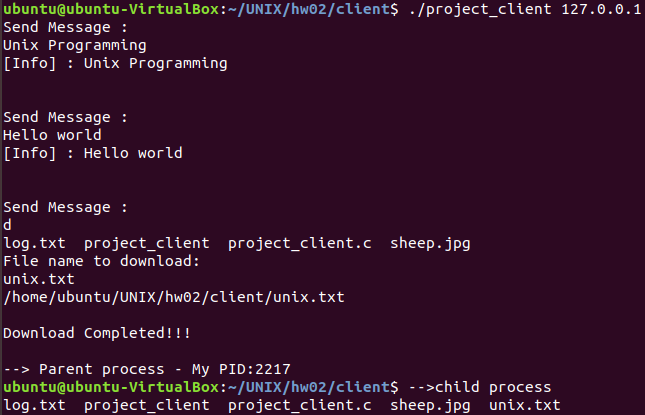


* 클라이언트 측에 생긴 log.txt 를 열어보면 실행 시간과 실행 로그가 남아있다.

1. 파일 전송 및 실행 로그



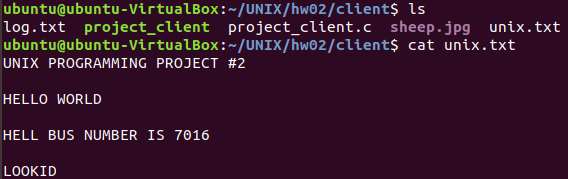
* 서버 측에 위와 같은 내용의 unix.txt 를 만들었다.



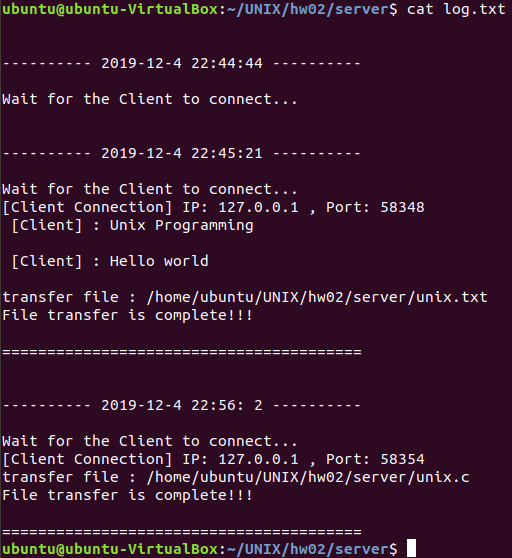
* 서버와 클라이언트를 연결하여 클라이언트 측에서 메시지 입력 터미널에 ‘d’ 명령어를 입력한다.
* 프로세스 시스템콜에 의하여 현재 클라이언트 측에 저장 되어있는 파일들이 출력되고 File name to download: 메시지가 뜬다.
* Unix.txt를 입력하면 서버로부터 수신될 unix.txt 파일이 클라이언트 측에서 저장될 경로를 출력하고 서버에 unix.txt 파일이 존재하면 성공적으로 다운로드를 했다는 Download Completed!!! 메시지가 출력된다.
* 프로세스 시스템콜에 의하여 서버로부터 수신된 unix.txt 파일을 포함하여 현재 클라이언트 측에 저장 되어있는 파일들이 출력된다.



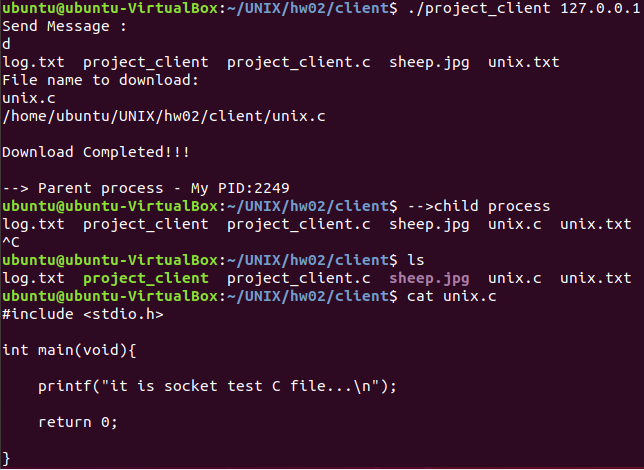
* 서버 측에서도 클라이언트로 전송된 파일의 기존 경로와 파일이 성공적으로 전송되었다는 메시지가 출력된다.



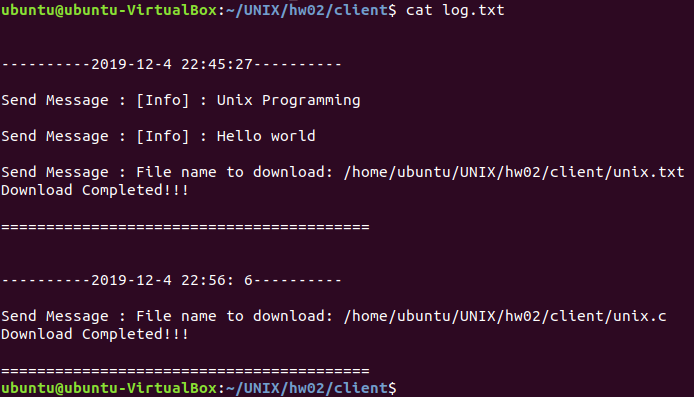
* 클라이언트로 전송된 파일을 cat으로 출력



* 프로그램 실행 시간과 파일 전송 로그도 log.txt에 정상적으로 입력된다.



* Unix.c (C source file) 파일도 정상적으로 송수신된다.

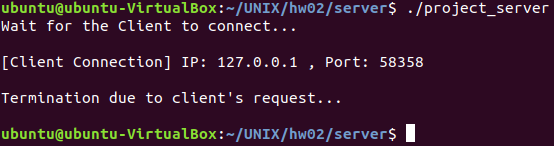


* Log.txt에 로그가 정상적으로 작성되었음을 알 수 있다.

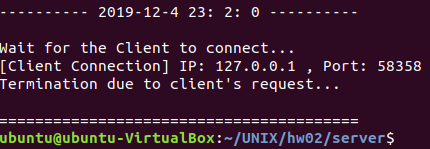
1. 프로그램 종료



* 클라이언트 측에서 ‘:quit’ 명령어를 입력한다.



* 서버 측에서 Termination due to client’s request 메시지를 출력하고 프로그램이 종료된다.



* 마찬가지로 프로그램의 실행 시간과 로그가 정상적으로 작성된다.

1. **소스코드**

**Project\_server.c (서버)**

1 #include <sys/socket.h>  
  2 #include <netinet/in.h>  
  3 #include <arpa/inet.h>  
  4 #include <stdio.h>  
  5 #include <stdlib.h>  
  6 #include <unistd.h>  
  7 #include <string.h>  
  8 #include <sys/types.h>  
  9 #include <time.h>  
 10 #include <fcntl.h>  
 11 #include <time.h>  
 12  
 13 #define PORT 7016   // port number  
 14 #define MAXPENDING 5  
 15 #define BUFSIZE 1024  
 16  
 17 /\* image \*/  
 18 void receiveImage(int socket){  
 19     int c;  
 20  
 21     FILE \*rfp;  
 22     FILE \*stream;  
 23  
 24     rfp = fdopen(socket,"r");  
 25     stream = fopen("sheep.jpg","wb");  
 26  
 27     while((c=fgetc(rfp)) != EOF){  
 28         fputc(c,stream);  
 29     }  
 30     fclose(rfp);  
 31     fclose(stream);  
 32 }  
 33  
 34  
 35 /\* log stamp \*/  
 36 void stamp\_log(char logbuf[]){  
 37  
 38     FILE \*lfp;  // log file pointer  
 39     char buf[256];  
 40  
 41     int n;  
 42  
 43     lfp = fopen("log.txt","a"); // append  
 44  
 45     int wordlen = 0;    // length of word  
 46  
 47     /\* count length of word \*/  
 48     for(int i = 0; i < sizeof(buf); i++){  
 49  
 50         if(logbuf[i] >= 0 && logbuf[i] <= 127){ // in ASCII  
 51             wordlen++;  
 52         }  
 53         else{  
 54             break;  
 55         }  
 56     }  
 57  
 58     strcpy(buf,logbuf);  
 59     buf[wordlen] ='\0';  
 60  
 61     fputs(buf,lfp); // put string in log.txt  
 62  
 63     fclose(lfp);  
 64 }  
 65  
 66 int main(int argc, char \*argv[])  
 67 {  
 68     int sockfd, servsockfd;  
 69     struct sockaddr\_in serv\_addr;  
 70     int servlen,recvlen;  
 71     char recvbuff[1024];    // receive buffer  
 72     time\_t ticks;  
 73     size\_t fpsize=0, fsize=0;  
 74  
 75     /\* FILE function \*/  
 76     char buff[1024];  
 77     char path[]="/home/ubuntu/UNIX/hw02/server/";   // location of file for transfer  
 78     FILE \*fp = NULL;  
 79  
 80     time\_t t = time(NULL);         // log stamp time  
 81     struct tm tm = \*localtime(&t); // log stamp time  
 82  
 83     char console[256];             // log stamp buffer  
 84  
 85     int filelen = 0; // length of filename  
 86  
 87     sprintf(console,"\n\n---------- %d-%d-%d %2d:%2d:%2d ----------\n\n",tm.tm\_year+1900, tm.tm\_mon+1, tm.tm\_mday, tm.tm\_hour, tm.tm\_min, tm.tm\_sec);  
 88     stamp\_log(console);  
 89  
 90     /\* create socket \*/  
 91     if((sockfd=socket(AF\_INET, SOCK\_STREAM, 0)) == -1)  
 92     {  
 93         perror("socket failed\n");  
 94         close(1);  
 95     }  
 96  
 97     serv\_addr.sin\_family = AF\_INET;  
 98     serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  // htonl  
 99     serv\_addr.sin\_port = htons(PORT);               // htons  
100  
101     /\* bind \*/  
102     if(bind(sockfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) == -1) {  
103         perror("bind failed\n");  
104         close(1);  
105     }  
106  
107     /\* listen \*/  
108     if(listen(sockfd,MAXPENDING) == -1 ){  
109         perror("listen failed\n");  
110         close(1);  
111     }  
112  
113     sprintf(console,"Wait for the Client to connect...\n");  
114     stamp\_log(console);  
115     puts(console);  
116  
117     servlen = sizeof(serv\_addr);  
118  
119     /\* accept \*/  
120     if((servsockfd=accept(sockfd, (struct sockaddr\*)&serv\_addr, &servlen)) == -1) {  
121         perror("accept failed\n");  
122         close(1);  
123     }  
124  
125     /\* Get a local Socket information \*/  
126     getpeername(servsockfd, (struct sockaddr \*)&serv\_addr, &servlen);  
127  
128     sprintf(console,"[Client Connection] IP: %s , Port: %d\n",inet\_ntoa(serv\_addr.sin\_addr), ntohs(serv\_addr.sin\_port));  
129     stamp\_log(console);  
130     puts(console);  
131  
132     while(1)  
133     {  
134         /\* receive \*/  
135         if((recvlen=recv(servsockfd, recvbuff, BUFSIZE-1, 0)) == -1) {  
136             perror("recv failed\n");  
137             break;  
138         }  
139  
140         /\* If has receive data \*/  
141         if(recvlen > 0)  
142         {  
143             recvbuff[recvlen] = 0x00;  
144  
145             /\* image \*/  
146             if(strlen(recvbuff)==2 && !strncmp(recvbuff,"i",1)){  
147  
148                 receiveImage(servsockfd);  
149                 sprintf(console,"receive Image complete!\n");  
150                 stamp\_log(console);  
151                 puts(console);  
152  
153             }  
154  
155             /\* If data is 'd' \*/  
156             if(strlen(recvbuff)== 2  && !strncmp(recvbuff,"d",1))  
157             {  
158                 /\* receive filename \*/  
159                 if((recvlen=recv(servsockfd, recvbuff, BUFSIZE-1, 0)) == -1){  
160                     perror("recv failed\n");  
161                     break;  
162                 }  
163  
164                 /\* compute length of file's words \*/  
165                 for(int i = 0; i < recvlen; i++){  
166                     if(recvbuff[i] >= 0 && recvbuff[i] <= 127){  
167                         filelen++;  
168                     }  
169                 }  
170  
171                 recvbuff[filelen] = '\0';   // null  
172  
173                 strcpy(path+30, recvbuff);  
174             //    path[strlen(path)-1]=0x00;  
175                 printf("File Path : %s\n",  path);  
176  
177                 stamp\_log("transfer file : ");  
178                 stamp\_log(path);  
179                 stamp\_log("\n");  
180  
181                 fp=fopen(path, "r");    // file open ; read  
182  
183                 if(fp == NULL)  
184                 {  
185                     perror("File does not exist...\n");  
186                     break;  
187                 }  
188  
189                 while(( fpsize=fread(buff,1,BUFSIZE-1,fp))>0){  // fread  
190                     send(servsockfd,buff,BUFSIZE-1,1);          // send; file transfer  
191                 }  
192  
193                 sprintf(console,"File transfer is complete!!!\n");  
194                 stamp\_log(console);  
195                 puts(console);  
196  
197                 strcpy(buff, "f");  
198                 sleep(5);  
199                 fclose(fp);  
200                 break;  
201             }  
202             if(strncmp(recvbuff, ":quit",5)==0) // receive ':quit' from client  
203             {  
204                 // termination  
205                 sprintf(console, "Termination due to client's request...\n");  
206                 stamp\_log(console);  
207                 puts(console);  
208  
209                 break;  
210             }  
211  
212             sprintf(console," [Client] : %s\n", recvbuff);  
213             stamp\_log(console);  
214             puts(console);  
215  
216             if(send(servsockfd, recvbuff, strlen(recvbuff),0) == -1){  
217                 perror("send failed\n");  
218                 break;  
219             }  
220  
221         }  
222  
223     }  
224     sprintf(console,"\n========================================\n");  
225     stamp\_log(console);  
226  
227     close(sockfd);  
228     close(servsockfd);  
229     sleep(1);  
230  
231 }

**Project\_client.c (클라이언트)**

  1 #include <sys/socket.h>  
  2 #include <sys/types.h>  
  3 #include <netinet/in.h>  
  4 #include <netdb.h>  
  5 #include <stdio.h>  
  6 #include <string.h>  
  7 #include <stdlib.h>  
  8 #include <unistd.h>  
  9 #include <arpa/inet.h>  
 10 #include <fcntl.h>  
 11 #include <time.h>  
 12  
 13 #define PORT 7016   // port number  
 14 #define BUFFER 1024  
 15  
 16 /\* image \*/  
 17 void sendImage(int socket){  
 18     int c;  
 19  
 20     FILE\* wfp;  
 21     FILE\* stream;  
 22  
 23     wfp = fdopen(socket,"w");  
 24     stream = fopen("sheep.jpg","rb");  
 25  
 26     while((c=fgetc(stream)) != EOF){  
 27         fputc(c,wfp);  
 28     }  
 29     fclose(wfp);  
 30     fclose(stream);  
 31 }  
 32  
 33  
 34 /\* log stamp \*/  
 35 void stamp\_log(char logbuf[]){  
 36  
 37     FILE \*lfp;  // log file pointer  
 38     char buf[256];  
 39  
 40     int n;  
 41  
 42     lfp = fopen("log.txt","a"); // append  
 43  
 44     int wordlen = 0;    // length of word  
 45  
 46     /\* count length of word \*/  
 47     for(int i = 0; i < sizeof(buf); i++){  
 48  
 49         if(logbuf[i] >= 0 && logbuf[i] <= 127){ // in ASCII  
 50             wordlen++;  
 51         }  
 52         else{  
 53             break;  
 54         }  
 55     }  
 56  
 57     strcpy(buf,logbuf);  
 58     buf[wordlen] = '\0';  
 59  
 60     fputs(buf,lfp); // put string in log.txt  
 61  
 62     fclose(lfp);  
 63 }  
 64  
 65 /\* argv[1] IP address 127.0.0.1 \*/  
 66 int main(int argc, char \*argv[])  
 67 {  
 68     int sockfd = 0, n = 0;  
 69     int recvlen;            // receive data length  
 70     char sendbuff[BUFFER];  // send buffer  
 71     char recvbuff[BUFFER];  // receive buffer  
 72     char buffer[BUFFER];  
 73     char path[]="/home/ubuntu/UNIX/hw02/client/";   // save location of file  
 74     struct sockaddr\_in serv\_addr;  
 75     char filename[100];     // file name  
 76     FILE \*fp=NULL;  
 77     char tmp[128];  
 78  
79     int wordlen = 0; // length of file's word  
 80  
 81     time\_t t = time(NULL);         // log stamp time  
 82     struct tm tm = \*localtime(&t); // log stamp time  
 83     char console[256];             // log stamp buffer  
 84  
 85     pid\_t pid;  // for process function  
 86  
 87     sprintf(console,"\n\n----------%d-%d-%d %2d:%2d:%2d----------\n\n",tm.tm\_year+1900, tm.tm\_mon+1, tm.tm\_mday,tm.tm\_hour, tm.tm\_min, tm.tm\_sec);  
 88     stamp\_log(console);  
 89  
 90     // argc != 2 -> error  
 91     if(argc != 2)  
 92     {  
 93         sprintf(console,"Usage: %s <ip of server> \n",argv[0]);  
 94         stamp\_log(console);  
 95         puts(console);  
 96         close(1);  
 97     }  
 98  
 99     /\* create socket \*/  
100     if((sockfd=socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {  
101  
102         perror("socket failed\n");  
103         close(1);  
104     }  
105  
106     serv\_addr.sin\_family = AF\_INET;  
107     serv\_addr.sin\_port = htons(PORT);   // htons  
108  
109     if(inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr) == -1)  
110     {  
111         sprintf(console,"inet\_pton error\n");  
112         puts(console);  
113         stamp\_log(console);  
114  
115         close(1);  
116     }  
117  
118     /\* connect \*/  
119     if(connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) == -1)  
120     {  
121         perror("connect failed\n");  
122         close(1);  
123     }  
124     while(1)  
125     {  
126         sprintf(console,"Send Message : ");  
127         stamp\_log(console);  
128         puts(console);  
129  
130         fgets(buffer, BUFFER,stdin);  
131  
132         /\* send \*/  
133         send(sockfd, buffer, strlen(buffer),0);  
134         sleep(1);  
135  
136         /\* termination on input ":quit" \*/  
137         if(strncmp(buffer, ":quit",5)==0)  
138         {  
139             break;  
140  
141         }  
142  
143         /\* image \*/  
144         else if(!strncmp(buffer,"i",1) && strlen(buffer) ==2){  
145  
146  
147             sprintf(console,"Image file name to download: ");  
148             stamp\_log(console);  
149             puts(console);  
150  
151             sendImage(sockfd);  
152  
153             sprintf(console,"Download complete!!\n");  
154             stamp\_log(console);  
155             puts(console);  
156             break;  
157         }  
158  
159         /\* Download file on input "d" \*/  
160         else if(!strncmp(buffer,"d",1) && strlen(buffer)==2)  
161         {  
162  
163             system("ls");  
164  
165             sprintf(console,"File name to download: ");  
166             stamp\_log(console);  
167             puts(console);  
168  
169             scanf("%s", filename);  
170             send(sockfd, filename, strlen(filename),0);  
171  
172             strcpy(path+30, filename);  
173             sprintf(console, "%s\n", path);  
174             puts(console);  
175             stamp\_log(console);  
176  
177             fp=fopen(path, "w");  
178  
179             /\* receive \*/  
180             while((recvlen=recv(sockfd, recvbuff, BUFFER-1, 0)) > 0){  
181                 recvbuff[recvlen]=0x00;  
182  
183                 /\* compute length of file's words \*/  
184                 for(int i = 0; i < recvlen; i++){  
185  
186                     if(recvbuff[i] >= 0 && recvbuff[i] <= 127){ // in ASCII  
187                         wordlen++;  
188                     }  
189                     else{  
190                         break;  
191                     }  
192                 }  
193  
194                 fwrite(recvbuff, 1,wordlen,fp); // fwrite  
195             }  
196             sprintf(console,"Download Completed!!!\n");  
197             stamp\_log(console);  
198             puts(console);  
199  
200             /\* ls ; execlp \*/  
201             switch (pid = fork()){  
202                 case -1: /\* fork failed \*/  
203                     perror("fork");  
204                     break;  
205                 case 0: /\* child process \*/  
206                     printf("-->child process\n");  
207                     if(execlp("ls", "ls", (char \*)NULL) == -1){ // execlp  
208                         perror("execlp");  
209                         exit(1);  
210                     }  
211                     break;  
212                 default :   /\* parent process \*/  
213                     printf("--> Parent process - My PID:%d\n",(int)getpid());  
214                     break;  
215             }  
216  
217             fclose(fp);  
218             break;  
219         }  
220  
221         else if((recvlen=recv(sockfd, recvbuff, BUFFER-1, 0)) == -1)  {  
222             perror("recv failed\n");  
223             break;  
224         }  
225         recvbuff[recvlen] = '\0';  
226         sprintf(console,"[Info] : %s\n", recvbuff);  
227         stamp\_log(console);  
228         puts(console);  
229  
230     }  
231  
232     sprintf(console,"\n=========================================\n");  
233     stamp\_log(console);  
234  
235     close(sockfd);  
236     return 0;  
237 }