1. epda.h

```
//define task number
//Notice you have to write all task names except DMB and MP3 with lowercase letters
#define CALL                0
#define CONVERSATION        1
#define SMS                 2
#define PBOOK               3
#define MP3                 4
#define DMB                 5
#define CAMERA              6
#define WEB                 7
#define NETWORK     8
#define MAIN                9

#define TASKS               10

//message code
#define REQUEST             0
#define RESPONSE    1
#define START               2
#define FINISH              3

#define     FLOG            11
#define     CLOG            22
#define     CFLOG           33
```

```
//Message format: type(=DestinationTaskNumber)+text(SourceTaskNumber+code+body(.. ))
//  Note that each ePDA service requires its own text format
struct MessageFormat{
        long int type;                  //equal to DestinationTaskNumber
        char text[BUFSIZ];      //Details are defined by TextFormat
};
struct SMSTextFormat{ //text format for the sms sending service
        int SourceTaskNumber;
        int code;
        char RecipientName[50];
        char RecipientNumber[50];
        char SenderName[50];
        char SenderNumber[50];
        char data[BUFSIZ-8-200];        //remaining date
};
struct MovieTextFormat{ //text format for the movie relay service
        int SourceTaskNumber;
        int code;
        char url[50];                   //destination site URL including path and filename as well
        char quality[50];       //source site URL including path and filename as well
        char fifo[50];                  //the name of named pipe, that is, FIFO
        char data[BUFSIZ-8-150-4];//remaining date
};
```

```
int mqid;                    //keep the ID of the message queue created by main task
int LogFlag=CFLOG; //default log flag meaning both console and file logging
int nrmessage=0,nsmessage=0; //numbers of internal messages received and sent,respectively
FILE *flog;                  //file pointer for log file

//logging both sent and received messages which are identified by the term in the prefix parameter
void MessageLog(FILE *flog, char prefix[50], struct MessageFormat message) {
        struct SMSTextFormat *text;     //SMSTextFormat is enough because only
destination/source task numbers and code are logged
        text=(struct SMSTextFormat *) message.text;
        if(LogFlag==CLOG || LogFlag==CFLOG) fprintf(stderr, "%s: Destination/Source Tasks = %d/
%d, code = %d\n", prefix, message.type, text->SourceTaskNumber, text->code);
        if(LogFlag==FLOG || LogFlag==CFLOG) fprintf(flog, "%s: Destination/Source Tasks = %d/%d,
code = %d\n", prefix, message.type, text->SourceTaskNumber, text->code);
        fflush(flog);                   //write immediately the logging data in disk file
}

//Copy a structured data block "b" of size "c" to "a" in which their stuctures may differ
void StructCopy(char *a,char *b, int c)
{
        int i;
        for (i=0;i<BUFSIZ;i++) *(a+i)=*(b+i);
}
```

2. main.c

```
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/msg.h>
#include <termios.h>
#include <term.h>
#include <curses.h>

static struct termios initial_settings, new_settings;

#include "epda.h"

#define TASKS 10

//Each record of task table has two fields of process id and task name
struct tasktable {
        int ProcessId;
        char TaskName[50];
        int Pipe2Task;          //pipe to child process
        int Pipe2Main;          //pipe from child process
};

struct tasktable tb[TASKS];

void init_keyboard(); void close_keyboard(); int kbhit(); int readch(); //for asynchronous(unblock) MMC

main()
{
        //Define the task table tb[i] of size TASKS, where i is task number
        int button,i;
        char c[10],cmqid[20],pipeid[4][50];
        key_t mqkey;            //define key to identify message queue. Notice that the key is a kind of
message queue name
        int Pipe2Main[TASKS][2],Pipe2Task[TASKS][2];
        struct msqid_ds *buf;

        if((flog=fopen("main.log","w"))==NULL) {fprintf(stderr,"Fail to oepn main.log file");exit(-1);}
        if((mqkey = ftok(".",'m'))==-1) {; //obtain a key needed to create a message queue. The key is
similar to file name needed to create a file
                if((mqkey = ftok(".",'e'))==-1) {fprintf(stderr, "Fail to obtain mkey !!");exit(-1);}
        }
        //create a message queue with key for message communication among TASKS tasks in ePDA
which is handled by message queue ID (mqid)
        //If alreay message queue, recreate message queue after remove the current message queue
```

```
which is to clear unexpected garbage up
        if((     0666|IPC_CREAT|IPC_EXCL))==-1) {
                (mqid, IPC_RMID, buf); //remove the message queue of mqid
                if((mqid=msgget(mqkey, 0666|IPC_CREAT))==-1) {fprintf(stderr, "Fail to obtain mqid at main!!");exit(-1);}
        }

        //Initialize pipe channels between parent and all child processes
        for(i=0;i<TASKS;i++) {
                if(pipe(Pipe2Main[i])<0) fprintf(stderr, "pipe open error");if(pipe(Pipe2Task[i])<0) fprintf(stderr, "pipe open error");;
//              close(Pipe2Main[i][1]);close(Pipe2Task[i][0]);//make a pipe channel
                tb[i].Pipe2Task=Pipe2Task[i][1];// PipeOut[i][1] : pipe to task i
                tb[i].Pipe2Main=Pipe2Main[i][0];//PipeIn[i][0] : pipe from task i
        }

        sprintf(cmqid,"%d",mqid);       //convert to string which is to put mqid in argv[] array

        //initialize tasktable
        for(i=0;i<TASKS;i++) tb[i].ProcessId=0;
        strcpy(tb[CALL].TaskName,"call");
        strcpy(tb[CONVERSATION].TaskName,"conversation");
        strcpy(tb[SMS].TaskName,"sms");
        strcpy(tb[PBOOK].TaskName,"pbook");
        strcpy(tb[MP3].TaskName,"MP3");
        strcpy(tb[DMB].TaskName,"DMB");
        strcpy(tb[CAMERA].TaskName,"camera");
        strcpy(tb[WEB].TaskName,"web");
        strcpy(tb[NETWORK].TaskName,"network");


while(1) {
        //Ask what kinds of action does user want to do
        printf("Button: 0(task ON),1(task OFF),2(sms), 3(camera),9(esc),10(stop)\n");
        scanf("%s",&c);
        if(c!="s") button=atoi(c);
        else fprintf(stderr, "remove some possible garbage occurred when type in 's' to stop a camera service \n");
//      scanf("%d",&button);
        switch (button) {
        case 0:
                //read task number that ePDA user pushes ON
                printf("ON: 0(call),1(conversation),2(sms),3(pbook),4(MP3),5(DMB),6(camera),7(web),8(network),9(esc)\n");
                scanf("%d",&button);
                if(button !=9 && button >=0 && button<=8) {
                        //Check whether the corresponding process is alreay running or not
                        //Notice that the ProcessId of 0 in task table means there is no running process for the task

                        if(tb[button].ProcessId==0) {
                                pid_t pid = fork();//fork new process
                                if(pid==0) {
                                        //if the new child process forked just before,
```

```
prepare parameters such as mqid and pipe IDs to handover the child process in string form
                                        fprintf(stderr, "      Success to fork the task of %s\n",tb[button].TaskName);
                                        sprintf(pipeid[0],"%d",Pipe2Main[button][0]);
                                        sprintf(pipeid[1],"%d",Pipe2Main[button][1]);
                                        sprintf(pipeid[2],"%d",Pipe2Task[button][0]);
                                        sprintf(pipeid[3],"%d",Pipe2Task[button][1]);
                                        char *argv[]={tb[button].TaskName,cmqid,pipeid[0],pipeid[1],pipeid[2],pipeid[3],NULL};//put all parameters to handover to the new process in argv
                                        //and then execute the corresponding task image file with argc and argv[]
                                        execv(argv[0],argv);
                                }else if(pid<0) fprintf(stderr, "   Fail to fork the task of %s\n",tb[button].TaskName);
                                else {
                                        tb[button].ProcessId=pid;       //the return value from fork(), i.e., pid is the id of the new process forked just before
                                        //make two one-way pipes of Pipe2Main and Pipe2Task
                                        close(  );close(  );//make a pipe channel
                                }
                        }else fprintf(stderr, "  The task of %s is already running\n",tb[button].TaskName);
                }
                break;
        case 1:
                //read task number that ePDA user pushes OFF
                printf("OFF: 0(call),1(conversation),2(sms),3(pbook),4(MP3),5(DMB),6(camera),7(web),8(network),9(esc)\n");
                scanf("%d",&button);
                if(button !=9 && button >=0 && button<=8) {
                        //Check whether the corresponding process is running now or not
                        //Notice that the ProcessId of non-zero in task table means there is a running process for the task
                        if(tb[button].ProcessId != 0) {
                                kill(tb[button].ProcessId,SIGKILL);// kill the process of the task to be OFF. Notice that the process id can be obtained from the task table in which the process's pid was registered when forked
                                tb[button].ProcessId=0;         //reset the corresponding ProcessId field in the task table
                                fprintf(stderr, "      Success to kill the task of %s\n",tb[button].TaskName);
                        }else fprintf(stderr, "  The task of %s is already OFF\n",tb[button].TaskName);
                }
                break;
        case 2:
                sms();
                break;
        case 3:
                camera();
        case 4:
        case 5:
```

```
		case 6:
		case 7:
		case 8:
		case 9:
			break;
		case 10: //if stop, kill all child processes and then exit
			for(i=0;i<TASKS;i++)  if(tb[i].ProcessId != 0)  kill(tb[i].ProcessId,SIGKILL);
			exit(0);
		}
}
		fclose(flog);
}


//MMC(Man-Machine Communication) routine for the sms task
sms() {
		struct MessageFormat smessage;struct SMSTextFormat stext;
		char name[20],mbuf[100];
		int button;

		printf("\nButton: 0(send new message),1(exit)\n");
		scanf("%d",&button);
		if(button==0) {
			printf("What is the receiver's name?\n");
			scanf("%s",&name);
			printf("Type the message that you'd like to send (enter enter Key to finish)\n");
			scanf("%s",&mbuf);
			//send the SMS sending request message to the sms task
			smessage.type=SMS;
			stext.SourceTaskNumber=MAIN;
			stext.code=REQUEST;
			strcpy(stext.RecipientName,name);
			strcpy(stext.data,mbuf);
			StructCopy(smessage.text,&stext,BUFSIZ);
			//Send the prepared message
			if(msgsnd(mqid,(void *)&smessage,BUFSIZ,IPC_NOWAIT)==-1) {fprintf(stderr,
"Fail to send message !!");exit(-1);}
			MessageLog(flog,"TX",smessage);nsmessage++;

		}
}

//MMC(Man-Machine Communication) routine for the camera task
camera() {
		struct MessageFormat smessage;struct MovieTextFormat stext;
		struct MessageFormat rmessage;struct MovieTextFormat *rtext;
		char ch,url[50],quality[50],buf[10];
		int button,nc,flag=0;

		printf("\nButton: 0(movie relay),1(exit)\n");
		scanf("%d",&button);
		if(button==0) {
			printf("What is the destination URL?\n");
```

```
scanf("%s",&url);
printf("What about the quality of the movie ? (resolution/fps)\n");
scanf("%s",&quality);
printf("OK ! Press 's' key to stop the movice relay service\n");

//send the movie relay request message to the camera task
smessage.type=CAMERA;
stext.SourceTaskNumber=MAIN;
stext.code=REQUEST;
strcpy(stext.url,url);
StructCopy(smessage.text,&stext,BUFSIZ);
//Send the prepared message
if(                ,(void  *)&smessage,BUFSIZ,0)==-1)  {fprintf(stderr,  "Fail  to  send
message !!");exit(-1);}
MessageLog(flog,"TX",smessage);nsmessage++;

//wait the movie relay start message from camera
if(     , (void *)&rmessage,BUFSIZ,MAIN,0)==-1) {fprintf(stderr, "Fail to receive the
movie relay start message at main !!");exit(-1);}
MessageLog(flog,"RX",rmessage);nrmessage++;
rtext=(struct MovieTextFormat *)rmessage.text;

//if receives the movie relay start message, dump it on display window and monitor
the stop key using the kbhit routines
init_keyboard();
if(rtext->SourceTaskNumber==CAMERA && rtext->code==START){
        while(1) {
moive relay service ****\n");flag=1;}

                                                //
service stops now\n");

camera task\n");

the camera task

                                        }
                }
                close_keyboard();
                init_keyboard();
                close_keyboard();
..." repeatly

                while(1) {
nc=         ,buf,sizeof(buf));
if(nc>0) {
        if(flag==0)  {fprintf(stderr,"\n***  Monitoring  the

        fprintf(stderr,"%s",buf);
}
if any key, send a signal to the camera task
if(kbhit()) {
        ch=readch();
        if(ch =='s'){
                fprintf(stderr,"\nThe  movie  relay

                fprintf(flog,"Sent the user signal 1 to

                kill(          ); //send the user signal 1 to

                break;
        }
}

//The following two lines are to fix temporary a problem that
// that the console seldom displays the message of "Button: 0
```

```
                    nc=read(tb[CAMERA].Pipe2Main,buf,sizeof(buf));
                    //if receives the movie relay stop message from camera task, stop it
                    if(msgrcv(mqid, (void *)&rmessage,BUFSIZ,MAIN,0)==-1)
{fprintf(stderr, "Fail to receive the movie relay start message at main !!");exit(-1);}
                    MessageLog(flog,"RX",rmessage); nrmessage++;
                    rtext=(struct MovieTextFormat *)rmessage.text;
                    if(rtext->SourceTaskNumber==CAMERA && rtext->code==FINISH)
break;
                }

        }
}

//The following four functions of init_keyboard(), close_keyboard(), kbhit() and readch()
// are to implement unblocking keyboard input function,
// which is mandatory to stop the camera's movie realy during the console monitors the movie relay

static struct termios initial_settings, new_settings;
static int peek_character = -1;

void init_keyboard()
{
   tcgetattr(0,&initial_settings);
   new_settings = initial_settings;
   new_settings.c_lflag &= ~ICANON;
   new_settings.c_lflag &= ~ECHO;
   new_settings.c_lflag &= ~ISIG;
   new_settings.c_cc[VMIN] = 1;
   new_settings.c_cc[VTIME] = 0;
   tcsetattr(0, TCSANOW, &new_settings);
}

void close_keyboard()
{
   tcsetattr(0, TCSANOW, &initial_settings);
}

int kbhit()
{
   char ch;
   int nread;

   if(peek_character != -1)
      return 1;
   new_settings.c_cc[VMIN]=0;
   tcsetattr(0, TCSANOW, &new_settings);
   nread = read(0,&ch,1);
   new_settings.c_cc[VMIN]=1;
   tcsetattr(0, TCSANOW, &new_settings);

   if(nread == 1) {
      peek_character = ch;
      return 1;
   }
```

```
    return 0;
}

int readch()
{
    char ch;

    if(peek_character != -1) {
        ch = peek_character;
        peek_character = -1;
        return ch;
    }
    read(0,&ch,1);
    return ch;
}
```

3. camera.c

```
#include <X11/Xlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <signal.h>

#include "epda.h"

#define RELAY           00
#define IDLE            01

int CameraState=IDLE;



//If receives the user signal 1 from camera task, log it and change CameraState
void relayoff(int sig) {
        fprintf(flog,"Receiced the user signal 1\n");
        CameraState=IDLE;
}

long long  calclock();
main(int argc, char *argv[]) {
        //Some Definitions for X-window
    Display *d ;
    Window   w, root ;
        Font f;
        GC gc;
        XSetWindowAttributes xswa;

        char msg0[100],msg1[100], msg2[100],msg3[100],mbf1[100],mbf2[100];
        struct   timeval  start_time, pre_time, cur_time, end_time;
        long long elapsedtime;
        int ctime1, ctime2,i;
        char cmd[100];
        double stime,a;
        FILE *fo;
        int Pipe2Main[2],Pipe2Me[2];



        //Set envirionments for X-window
    d = XOpenDisplay(NULL) ;
    root = XDefaultRootWindow (d);
    w = XCreateSimpleWindow ( d, root, 10, 110, 250, 100,
                2, BlackPixel (d,0), WhitePixel(d,0) );
        XChangeWindowAttributes (d, w, CWOverrideRedirect, &xswa);
```

```
    XMapWindow (d, w);
        gc=XCreateGC(d,w,0L, (XGCValues *) NULL);
        f=XLoadFont (d,"fixed");
        XSetFont (d,gc,f);

            (    ,relayoff); //register the service routine for the user signal 1

        gettimeofday(&start_time,NULL); //set the starting time

        if((flog=fopen("camera.log","w"))==NULL) { perror("file open error");exit(-1);} //open log file

        //get the id of the message queue created by main task
        mqid=atoi(argv[1]);
        //get pipe IDs and make two one way pipe
        Pipe2Main[0]=atoi(argv[2]);Pipe2Main[1]=atoi(argv[3]);
        Pipe2Me[0]=atoi(argv[4]);Pipe2Me[1]=atoi(argv[5]);
        close(      ); close(Pipe2Me[1]);
//      fprintf(stderr,"Pipe2Main[0]=%d, Pipe2Main[1]=%d\n",Pipe2Main[0],Pipe2Main[1]);

        while(1) {
                for(i=0;i<100;i++)
msg0[i]=msg1[i]=msg2[i]=msg3[i]=mbf1[i]=mbf2[i]='\0';//initizlize message buffers
                sprintf(msg0,"camera task: ");

                task(Pipe2Main[1]);

                gettimeofday(&cur_time,NULL);

                //*************************************************************************
                //obtain accumulated CPU time which is the sum of the 42 and 43'th fields of
/proc/ppid/stat
                //*************************************************************************
                //input the result of "cat /proc/ppid/stat" to "awk '{printf $42}'"
                sprintf(cmd,"cat /proc/%d/stat |awk '{printf $42}'",getpid());
                //input the result of "awk '{printf $42}'",i.e., the 42'th field, to ctime1 by means of
pipe
                fo=popen(cmd,"r"); fscanf(fo,"%d",&ctime1); fclose(fo);
                //input the result of "cat /proc/ppid/stat" to "awk '{printf $43}'"
                sprintf(cmd,"cat /proc/%d/stat |awk '{printf $43}'",getpid());
                //input the result of "awk '{printf $42}'",i.e., the 42'th field, to ctime1 by means of
pipe
                fo=popen(cmd,"r"); fscanf(fo,"%d",&ctime2); fclose(fo);
                //obtain accumulated CPU time through adding two fields and normalizing it in
terms of second unit
                a=ctime1;a+=ctime2;a=a/100;

                //obtain accumulated service time of the differece between the starting and current
times, and normalize it in terms of second unit
                stime=calclock(start_time, cur_time);stime=stime/1000000;

                //printf("service time = %1.3f [sec], CPU time =%1.3f [Sec]\n", stime,a);

                //make three messages to display on the window
```

```
                sprintf(mbf1,"%d/%d,",getpid(),getppid());
                sprintf(mbf2," %1.1f/%1.1f",a,stime);
                strcat(msg0,mbf1);strcat(msg0,mbf2);
                sprintf(msg1,"Received messages = %d",nrmessage);
                sprintf(msg2,"Sent messages = %d",nsmessage);
                sprintf(msg3,"Charge = ");

                //After clear the window, display four messages on the window
                XClearWindow(d,w);
                XDrawString(d,w,gc,5,20,msg0,sizeof(msg0));
                XDrawString(d,w,gc,15,45,msg1,sizeof(msg1));
                XDrawString(d,w,gc,15,65,msg2,sizeof(msg2));
                XDrawString(d,w,gc,15,85,msg3,sizeof(msg3));
                XFlush (d);

        }
        //close procedure for X-window
        XUnloadFont (d,f);
        XFreeGC(d,gc);
        XDestroyWindow(d,w);
    XCloseDisplay (d);
}


//task specific module
task(int Pipe2Main) {
    struct MessageFormat rmessage;struct MovieTextFormat *rtext;
    struct MessageFormat smessage;struct MovieTextFormat stext;
    struct tm t;
    char buf[100],fifo_file[20];
    FILE *fs,*fo;

    //If there is no message to receive, return immediately
    if(msgrcv(mqid, (void *)&rmessage,BUFSIZ,CAMERA,IPC_NOWAIT) == -1 ) return;

    MessageLog(flog,"RX",rmessage); nrmessage++;
    rtext=(struct MovieTextFormat *) rmessage.text;

    if(rtext->SourceTaskNumber==MAIN && rtext->code==REQUEST) {

            //send the movie relay start message to the main task
            smessage.type=MAIN;
            stext.SourceTaskNumber=CAMERA;
            stext.code=START;
            strcpy(stext.url,rtext->url);
            strcpy(stext.quality,rtext->quality);
            StructCopy(smessage.text,&stext,BUFSIZ);
            //Send the prepared message
            if(msgsnd(mqid,(void *)&smessage,BUFSIZ,0)==-1) {fprintf(stderr, "Fail to send
message !!");exit(-1);}
            MessageLog(flog,"TX",smessage);nsmessage++;

            //send the movie relay start message to the network task
```

```
                smessage.type=NETWORK;
                strcpy(fifo_file,"/tmp/camera2network");//set the file name for named pipe
                (     , S_IFIFO|0666,0);              //make named pipe of camera2network
                strcpy(stext.fifo,fifo_file);
                StructCopy(smessage.text,&stext,BUFSIZ);
                if(msgsnd(mqid,(void  *)&smessage,BUFSIZ,0)==-1)  {fprintf(stderr,  "Fail  to  send
message !!");exit(-1);}
                MessageLog(flog,"TX",smessage);nsmessage++;

                //emulate  the  movice  relay  service  with  dumping  repeatly  the  source  file  of
camera.c
                CameraState=RELAY;
                fs=fopen("camera.c","r");
                fo=fopen(fifo_file,"w");
                while (CameraState==RELAY) {
                        sleep(1);
                        if(fscanf(fs,"%s",&buf)==EOF) {fclose(fs);fs=fopen("camera.c","r");}
                        //fprintf(stderr,"%s",buf);
                            (     ,buf,sizeof(buf));
                        fputs(buf,fo);
                }
                fclose(fs);
                fclose(fo);

                //send the movie relay finish message to the main task
                smessage.type=MAIN;
                stext.SourceTaskNumber=     ;
                stext.code=FINISH;
                StructCopy(smessage.text,      ,BUFSIZ);
                //Send the prepared message
                if(msgsnd(mqid,(void  *)&smessage,BUFSIZ,0)==-1)  {fprintf(stderr,  "Fail  to  send
message !!");exit(-1);}
                MessageLog(flog,"TX",smessage);nsmessage++;

                //send the movie relay finish message to the network task
                smessage.type=      ;
                StructCopy(smessage.text,&stext,BUFSIZ);
                if(msgsnd(mqid,(void  *)&smessage,BUFSIZ,0)==-1)  {fprintf(stderr,  "Fail  to  send
message !!");exit(-1);}
                MessageLog(flog,"TX",smessage);nsmessage++;

        }
}

long long  calclock( struct timeval clock_a, struct timeval clock_b ) {
long long timedelay, temp, temp_n;
if (clock_a.tv_usec >= clock_a.tv_usec) {
temp = clock_b.tv_sec - clock_a.tv_sec;
temp_n = clock_b.tv_usec - clock_a.tv_usec;
timedelay = 1000000 * temp + temp_n;
} else {
temp = clock_b.tv_sec - clock_a.tv_sec - 1;
temp_n = 1000000 + clock_b.tv_usec - clock_a.tv_usec;
timedelay = 1000000 * temp + temp_n;
```

```
}
return timedelay;
}
```

4. network.c

```
#include <X11/Xlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include "epda.h"

long long  calclock();
main(int argc, char *argv[]) {
        //Some Definitions for X-window
    Display *d ;
    Window   w, root ;
        Font f;
        GC gc;
        XSetWindowAttributes xswa;

        char msg0[100],msg1[100], msg2[100],msg3[100],mbf1[100],mbf2[100];
        struct   timeval  start_time, pre_time, cur_time, end_time;
        long long elapsedtime;
        int ctime1, ctime2,i;
        char cmd[100];
        double stime,a;
        FILE *fo;
        int Pipe2Main[2],Pipe2Me[2];



        //Set envirionments for X-window
d = XOpenDisplay(NULL) ;
root = XDefaultRootWindow (d);
w = XCreateSimpleWindow ( d, root, 10, 110, 250, 100,
                2, BlackPixel (d,0), WhitePixel(d,0) );
        XChangeWindowAttributes (d, w, CWOverrideRedirect, &xswa);
XMapWindow (d, w);
        gc=XCreateGC(d,w,0L, (XGCValues *) NULL);
        f=XLoadFont (d,"fixed");
        XSetFont (d,gc,f);

        gettimeofday(&start_time,NULL); //set the starting time

        if((flog=fopen("network.log","w"))==NULL) { perror("file open error");exit(-1);} //open log file

        //get the id of the message queue created by main task
        mqid=atoi(argv[1]);

        //get pipe IDs and make two one way pipe
        Pipe2Main[0]=atoi(argv[2]);Pipe2Main[1]=atoi(argv[3]);
        Pipe2Me[0]=atoi(argv[4]);Pipe2Me[1]=atoi(argv[5]);
```

```
        close(Pipe2Main[0]); close(Pipe2Me[1]);

        while(1) {
                for(i=0;i<100;i++)
msg0[i]=msg1[i]=msg2[i]=msg3[i]=mbf1[i]=mbf2[i]='\0';//initizlize message buffers
                sprintf(msg0,"network task: ");

                task();


                gettimeofday(&cur_time,NULL);

                //*********************************************************************************
                //obtain accumulated CPU time which is the sum of the 42 and 43'th fields of
/proc/ppid/stat
                //*********************************************************************************
                //input the result of "cat /proc/ppid/stat" to "awk '{printf $42}'"
                sprintf(cmd,"cat /proc/%d/stat |awk '{printf $42}'",getpid());
                //input the result of "awk '{printf $42}'",i.e., the 42'th field, to ctime1 by means of
pipe
                fo=popen(cmd,"r"); fscanf(fo,"%d",&ctime1); fclose(fo);
                //input the result of "cat /proc/ppid/stat" to "awk '{printf $43}'"
                sprintf(cmd,"cat /proc/%d/stat |awk '{printf $43}'",getpid());
                //input the result of "awk '{printf $42}'",i.e., the 42'th field, to ctime1 by means of
pipe
                fo=popen(cmd,"r"); fscanf(fo,"%d",&ctime2); fclose(fo);
                //obtain accumulated CPU time through adding two fields and normalizing it in
terms of second unit
                a=ctime1;a+=ctime2;a=a/100;

                //obtain accumulated service time of the differece between the starting and current
times, and normalize it in terms of second unit
                stime=calclock(start_time, cur_time);stime=stime/1000000;

                //printf("service time = %1.3f [sec], CPU time =%1.3f [Sec]\n", stime,a);

                //make three messages to display on the window
                sprintf(mbf1,"%d/%d,",getpid(),getppid());
                sprintf(mbf2," %1.1f/%1.1f",a,stime);
                strcat(msg0,mbf1);strcat(msg0,mbf2);
                sprintf(msg1,"Received messages = %d",nrmessage);
                sprintf(msg2,"Sent messages = %d",nsmessage);
                sprintf(msg3,"Charge = ");

                //After clear the window, display four messages on the window
                XClearWindow(d,w);
                XDrawString(d,w,gc,5,20,msg0,sizeof(msg0));
                XDrawString(d,w,gc,15,45,msg1,sizeof(msg1));
                XDrawString(d,w,gc,15,65,msg2,sizeof(msg2));
                XDrawString(d,w,gc,15,85,msg3,sizeof(msg3));
                XFlush (d);

        }
        //close procedure for X-window
```

```
        XUnloadFont (d,f);
        XFreeGC(d,gc);
        XDestroyWindow(d,w);
    XCloseDisplay (d);
}



//task specific module
task() {
        struct MessageFormat rmessage;struct MovieTextFormat *rtext;
        struct tm t;
        FILE *fo;
        int nc;
        char buf[100];

        //If there is no message to receive, return
        if(msgrcv(mqid, (void *)&rmessage,BUFSIZ,     ,IPC_NOWAIT) == -1 ) return;
        MessageLog(flog,"RX",rmessage); nrmessage++;


        rtext=(struct MovieTextFormat *) rmessage.text;

        //if receives the movie relay start message from camera, treats it
        if(rtext->SourceTaskNumber==     && rtext->code==     ) {
                fo=fopen(      ,"r");       //open named pipe

                //emulate the movie relay service by dump it into the log file instead of
transmitting to network
                fprintf(flog, "%s",asctime(&t));
                fprintf(flog, "Destination = %s \n",rtext->url);
                while(1) {
                        fgets(buf,sizeof(buf),    );
                        fprintf(flog,"%s", buf);

                        //check if the message of the Movie relay serice finish arrives
                        if(msgrcv(mqid, (void *)&rmessage,BUFSIZ,     ,IPC_NOWAIT) !=-1) {
                                MessageLog(flog,"RX",rmessage); nrmessage++;
                                //if receives the message, stop
                                rtext=(struct          *) rmessage.text;
                                if(rtext->SourceTaskNumber==CAMERA          &&          rtext-
>code==FINISH) break;
                        }
                }
                fclose(fo);
        }
}

long long  calclock( struct timeval clock_a, struct timeval clock_b ) {
long long timedelay, temp, temp_n;
if (clock_a.tv_usec >= clock_a.tv_usec) {
temp = clock_b.tv_sec - clock_a.tv_sec;
temp_n = clock_b.tv_usec - clock_a.tv_usec;
timedelay = 1000000 * temp + temp_n;
```

```
} else {
temp = clock_b.tv_sec - clock_a.tv_sec - 1;
temp_n = 1000000 + clock_b.tv_usec - clock_a.tv_usec;
timedelay = 1000000 * temp + temp_n;
}
return timedelay;
}
```