Birzeit University - Faculty of Information Technology
Computer Systems Engineering Department – ENCS531
1st semester - 2012/13 - first hour exam – duration (80 minutes)

Real-Time Applications and Embedded Systems

Instructor: Dr. Ahmad Afaneh

Q1. Select the most correct answer. Write down your final answers in the following table. (35 points)

1	2	3	4	5	6	7
B	A	B	C	B	B	C

1) The difference between the process and the program is

a. They are the same
b. The process includes the program
c. The program includes the process
d. none of the above

2) How many essential IPC communication models are there

a. 2 b. 5 c. 8 d. none of the above

3) When a signal is masked (blocked) it will

a. never be delivered
b. be delivered once it is unblocked
c. still cause an interrupt
d. none of the above

4) Which statement is false regarding Named Pipes

a. exist as device special files
b. managed by the OS
c. they only work for parent child communication

5) Mailslot is used for two way inter-process communications.

a. True
b. False

6) Shared memory is the fastest form of RPC

a. True
b. False

7) Marshalling in RPC is performed by

a. Client code
b. server code
c. stubs
d. all of the above

Q2. (65 points) In today's world, applications are quickly moving towards a simple, all-encompassing distribution model. Web applications are gaining popularity because of their scalability and ease of deployment, and desktop applications are becoming less common. This holds both positive and negative consequences - mainly with functionality and user experience. Most applications need to ensure the best user experience possible for any given situation. In many cases, a web site meets the needs of both the developer and the consumer. However, some applications are better suited as a client-side, distributed application. For these applications, the need arises for an easy and reliable method of deployment that allows the application the flexibility for any scenario. It should gracefully handle **update**s to the application, and be easily managed remotely. *For this reason, a simple client/server framework to update client software automatically is needed.*

You task is to implement the client/ server update framework

The update protocol:

1. Client Side
   - The client main function calls the CheckForUpdate() function
   - The CheckForUpdate function will
     i. Connect to the update server ("update.birzeit.edu", port 2345) **[5 points]**
     ii. Send the current version of the software to the server as an integer**[5points]**
     iii. If the server reply is that a newer version exists then client will download the update file and *run it* otherwise the function will return**[15 points]**
   - Assume the client has a function *int getCurrentVersion()* that will return the software version as an integer (you don't have to write this function)

2. Server Side
   - Server will start and wait for connections **[10 points]**
   - Once a client connection is established
     i. The server will wait for the client to send its current version **[5 points]**
     ii. The server will compare the clients version to the latest version
     iii. The server will let the client know if it needs an update
     iv. If the client needs update the server will send the update file to the client otherwise the server will close the connection **[10 points]**
   - Assume the server has a function *int IsUpToDate(int)* that will take the client's version and then return 0 if an update is available and 1 if the client is up to date
   - The server should handle multiple clients simultaneously (i.e. no client should wait for any other client). **[15 points for using threads]**

a) Implement both the client and the server (under either Linux or Win32)
b) You should use sockets for client/server communication
c) Include some comments in your code to indicate the main tasks in your code

Good Luck

//////////////////// ***server.c***//

```
Int main()
{
        int srvfSoc;
        struct sockaddr_in srv;        /* used by bind() */
        /* create socket */
        if((srvfSoc = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(1);
        }
        srv.sin_family = AF_INET; /* use the Internet addr family */
        srv.sin_port = htons(2345); /* bind socket 'fd' to port 2345*/
        /* bind: a client may connect to any of my addresses */
        srv.sin_addr.s_addr = htonl(INADDR_ANY);
        if(bind(srvfSoc, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
                perror("bind"); exit(1);
        }
        if(listen(srvfSoc, 5) < 0) {
                perror("listen");
                exit(1);
        }
```

While(1)

```
{/* accept new client*/
        struct sockaddr_in cli;          /* used by accept() */
        int* newfd;                      /* returned by accept() */
        int cli_len = sizeof(cli);
        pthread_t f_thread;
        void *ProcessClient(int*);
        newfd= (int*)malloc(sizeof(int));
        *newfd = accept(fd, (struct sockaddr*) &cli, &cli_len);
        if(*newfd < 0) {
        perror("accept");       exit(1);
        }
        /* create new thread to handle each client*/
        pthread_create(&f_thread,NULL, ProcessClient, newfd);



}
```

} // end of main

Void* ProcessClient (int* fd)

```
{
    int clientfd=*fd;
    free (fd);
    int ver;
    int nbytes;
    int reply;
    /* read client version*/
    if((nbytes = read(clientfd, &ver, sizeof(ver))) < 0) {
        perror("read"); exit(1);
    }
```

```
        /* check if client needs update*/
        if(reply =IsUptoDate(ver)) {
                /* client is up to date */
                  if((nbytes = write(clientfd, &reply, sizeof(reply))) < 0)          perror("write");
                  close(clientfd);
                  pthread_exit(0);

        }
        else
        {       /* client needs an  update */
                        if((nbytes = write(clientfd, &reply, sizeof(reply))) < 0) {
                           perror("write"); }
                  int filed;
                  /* open update file*/
                  filed=int open(UPDATE_FILE, O_RDONLY);
                 /* stream the file*/
                  int buf[BUF_SIZE];
                 while (1)
                 {
                        // read from file
                        if((nbytes = read(filed, buf, BUF_SIZE)) < 0) {
                        perror("read"); exit(1);}
                        //write to client
                        write(clientfd, buf,nbytes);
                        if (nbytes <BUF_SIZE) // if end of file
                        {

                                close(filed);
                                close(clientfd);

                                pthread_exit(0);

                        }


                 }

        }
}
```

//////////////////// ***client.c***

```
void CheckForUpdate()
{     /* resolve server address*/
          struct hostent *hp; /*ptr to host info for remote*/
          struct sockaddr_in srv;
          char *name = “update.birzeit.edu”;
          srv.sin_family = AF_INET;
          hp = gethostbyname(name)
          srv.sin_addr.s_addr = ((struct in_addr*)(hp->h_addr))->s_addr;
           srv.sin_port = htons(2345);
          /* connect to server*/
          if(connect(fd, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
                    perror(”connect"); exit(1);
          }
          int ver;
          int reply;
          int nbytes;
          int buf[BUF_SIZE];
          vet=getCurrentVersion();
          /* send current version to the server */

          if((nbytes = write(fd, &ver, sizeof(ver))) < 0) {

                              perror(“write”); exit(1);}

          /* read server reply 0: need update , 1: no update */

          if((nbytes = read(fd, &reply, sizeof(reply))) < 0) {

                              perror(“write”);exit(1); }

          if(reply) // no update exit the function

          {close (fd); return; }

          // there is an update

          /* create and download update file from the server */

          Int filed;

          filed= open(UPDATE_FILE, O_CREAT| O_WRONLY);


          /* download file*/
```

```
        while (1)
        {       /*read from server */
                if((nbytes = read(fd, buf, BUF_SIZE)) < 0) {
                perror("read"); exit(1);}
                        //write to file
                        write(filed, buf,nbytes);
                        if (nbytes <BUF_SIZE) // if end of file
                        {
                                close(filed);
                                close(fd);

                                return;
                        }

        }

}

Int main ()
{
        CheckForUpdate();
         //…. Client code ……..…..//
        return 0;

}
```