84

Name: Ameer Nasr-Allah

Student ID: 1090338

Birzeit University - Faculty of Information Technology

Computer Systems Engineering Department – ENCS531
2nd semester - 2012/13 - first hour exam – duration (80 minutes)- 19 MAR 2013

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. (30 points)

25

1	2	3
b	b	1. Shared memory 2. Message Queue
4	5	6
c	b	In windows you can't change maximum value of semaphores (gives error when greater), In Linux it just ignores it and continues.

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) **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

3) **The essential IPC communication models are**

1. Shared Memory
2. Message Queue

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) **What is the main difference in semaphore implementation between Linux and Windows ?**

Name: Ameer Nasrallah

Student ID: 1090338

Q2. (70 points) In preparations for the next code competition we decided to write our own automated judging system (a system that will decide if the code is correct or not). The system contains a client, server and other processes. Your task is to just write the server. The server job is to receive the source code, make sure it is safe to run, compile it and test it using predefined tests. The following are the main steps

1. *The server is initialized and ready to accept connections*
2. *The client connects to the server*
3. *The client sends the source code to the server*
4. *The server sends the file name to another process on the server (policyCheck) using any IPC*
5. *policyCheck then sends either 0 or 1 as a response to the server (0: code is not safe the server sends an error to the client, 1: the code is safe the server continues*
6. *the server compiles the code (hint: it calls gcc)*
7. *the server has test files input.txt and output.txt, the server runs the code with input.txt as input and compares the output with output.txt (hint use < and > to redirect input and output)*
8. *the server has a function checkOutput (char* file1,char* file2) that returns true if the two files match 0 otherwise. (Don't write this function assume it exists already)*
9. *the server sends the results report to the client*
10. *connection closed*

your task is to write the sever process , please keep in mind

- ***the server should support multiple clients***
- ***the details of messages and communication between the processes is not strict***
- ***the result report format is not strict***

//Define message Queue structure
struct msgbuf {
    Long mtype;
    Char fileName[256];
    int response; };

59

server.c

Name: Ameer Nasrallah
Student ID: 1090338

int main() {
    int fd; //start step 1
    struct sockaddr_in srv;
    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket error"); exit(1);
    }
    srv.sin_family = AF_INET;
    srv.sin_port = ntohs(1234); //port number (receiving so use ntohs)
    srv.sin_addr.s_addr = ntohl(INADDR_ANY); // server address
    if(bind(fd, (struct sockaddr *) &srv, sizeof(srv)) == -1) {
        perror("bind error"); exit(1);
    }
    if(listen(fd, 5) == -1) { perror("listen error"); exit(1); } // max 5 queue size
    while (1) { // start accepting clients
        struct sockaddr_in cli;
        int *newfd = (int *) malloc(sizeof(int));
        *newfd = accept(fd, (struct sockaddr *) &cli, sizeof(cli)); //end of step 1
//start of 3 ← if (*newfd == -1) { perror("error Client"); }
        int codeFile = open("./code.c", O_WRONLY); // create code.c
        char buf[80]; // buffer to read from client and write to codeFile
        while (read(*newfd, buf, 80) != 0) // while not end of client socket
        { if(write(codeFile, buf, 80) < 0) { perror("write error"); exit(1); }
        }
        // finished reading from client socket and writing to code.c
//start of 4 ← close(codeFile); // close code.c
        int msqid = msgget(ftok(".", 'm'), IPC_CREAT | 0666);
        struct msgbuf msg; // message to communicate with policyCheck
        msg.mtype = 1; // policyCheck type
        strcpy(msg.fileName, "./code.c");
        msg.response = 0; //initial, not important here
        int n = sizeof(struct msgbuf) - sizeof(long);
        if(msgsnd(msqid, &msg, n, 0) == -1) { perror("send error"); exit(1); }

//step 5) server type = 2
if(msgrcv(msqid, &msg, n, 2, 0) == -1) { perror("receive error"); exit(1); }
if(msg.response == 0) { // error in code
    if(write(*newfd, "error in code", 20) < 0){ {
        perror("code error"); exit(1); }
    close(*newfd); // close client socket and exit
    free(newfd);
    exit(0);
}

//step 6, compile code using execlp, command path = "/bin/usr/gcc" (not sure about gcc path)
assume code.c in same path
execlp("/bin/usr/gcc", "gcc", "-o code", "code.c");
// result is: gcc -o code code.c → code executable
// step 7, execute code < input.txt > output.txt, assuming all files and terminal are in current directory
execlp("./", "code", "< input.txt > output.txt");
// step 8, read output file from client and write it to client out.txt
int clientOut = open("./clientout.txt", O_WRONLY);
while(read(*newfd, buf, 80) != 0) {
    if (write(clientOut, buf, 80) < 0) { perror("write error"); exit(1); }
}
// now send the two files: output.txt and clientout.txt to method
int result = CheckOutput("output.txt", "clientout.txt");
char *status;
if (result) {
    strcpy(status, "Good code, success!");
}
else {
    strcpy(status, "Bad code, failure!"); }
// step 9
write(*newfd, status, strlen(status));
// step 10
    close(*newfd); free(newfd);
} // end of while
return 0;
} // end of server.c

Notes
All steps done assuming synchronization with client ~~of client~~ ~~code work~~