Network-Centric Programming – Spring 2021
Mid-Term Exam
April 1th (Thursday) 10:00AM—1:00PM

Instructions

1. Please use your computer to complete the Midterm problems. You are not allowed to access any Internet resource or part of in-class code examples during the exam.

2. Login to Webex: https://rutgers.webex.com/meet/bh439 at 10:00AM. Open your camera and mute your microphone.

3. Go to Sakai assignments and download Midterm questions at 10:00AM.

4. Provide the solution to each problem in a separate file, for example:
   prob1_read.c, prob1_write.c
   prob2_server.c, prob2_client.c
   prob3.c
   prob4.c

5. Provide *Makefile* for each problem and include a *README* file to describe how to compile/test the program.

6. You have 180 minutes to answer the questions. After you are done please upload your solutions to the Midterm assignment in Sakai. You could upload either one compressed file or multiple original files.

Problem 1 – Basic File Read/Write

Implement a program that can read from the keyboard and write to the file. First, create a file and write any sentence into the file. Then, read the sentence from the file and display it on the screen. You need to implement this problem with both *systemcall* and *standard I/O* functions.

Make your code readable and robust and output standard Linux error messages if an error occurs. For example, if a file is failed to be opened (i.e., file descriptor <0), you can use *perror("Opening File Failed")* to output error messages.

You may need to include the following header files:

```
#include <sys/types.h>
#include <sys/uio.h>
```

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
```

Problem 2 – Socket

Implement a basic socket server and client using the port 8080. After the connection is established, the client sends the following C string to the server.
"Network-Centric Programming – Spring 2021 Midterm"

Once the server receives the message from the client, the server will print it out on the screen.

Make your code readable and robust and output standard Linux error messages if an error occurs. For example, if a socket is failed to be created, you can use *perror("Socket Creation Failed")* to output error messages.

You may need to include the following header files:

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
```

Problem 3 – Process

Implement a program that uses fork() to create a child process. In the parent process, please print out the C string "I'm in the parent process". And in the child process, please print out the string "I'm in the child process".

Make your code readable and robust and output standard Linux error messages if an error occurs. For example, if the child process is failed to be created, you can use *perror("Process Creation Failed")* to output error messages.

You may need to include the following header files:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

Problem 4 – Thread

Implement a program that uses *pthread* to create two threads. In the main thread and two created peer threads, please print out the following C strings, respectively.

"This is main thread.\n"
"This is the first created thread.\n"
"This is the second created thread.\n"

Make your code readable and robust and output standard Linux error messages if an error occurs. For example, if a thread is failed to be created, you can use *perror("Thread Creation Failed")* to output error messages.

You may need to include the following header files:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```