Homework #5

[ECE30021/ITP30002] Operating Systems

Mission

- Solve problem 1 and 2
  - Solve the problems on Ubuntu Linux (VirtualBox or UTM) using vim and gcc.
  - Problem 1 is a group assignment but Problem 2 is not.
- Submission
  - Submit a .tgz file containing hw5_1.c and hw5_2.c on LMS.
    "tar cvfz hw5_<*student_id*>.tgz hw5_1.c hw5_2.c"
    (Do not copy&past but type the above command.)
  - After compression, please check the .tgz file by decompressing in an empty directory.
    "tar xvfz hw5_<*student_id*>.tgz"
- Due date: PM 11:00:00, May 9th

Honor Code Guidelines

- “과제”
  - 과제는 교과과정의 내용을 소화하여 실질적인 활용 능력을 갖추기 위한 교육활동이다. 학생은 모든 과제를 정직하고 성실하게 수행함으로써 과제에 의도된 지식과 기술을 얻기 위해 최선을 다해야 한다.
  - 제출된 과제물은 성적 평가에 반영되므로 공식적으로 허용되지 않은 자료나 도움을 획득, 활용, 요구, 제공하는 것을 포함하여 평가의 공정성에 영향을 미치는 모든 형태의 부정행위는 단호히 거부해야 한다.
  - 수업 내용, 공지된 지식 및 정보, 또는 과제의 요구를 이해하기 위하여 동료의 도움을 받는 것은 부정행위에 포함되지 않는다. 그러나, 과제를 해결하기 위한 모든 과정은 반드시 스스로의 힘으로 수행해야 한다.
  - 담당교수가 명시적으로 허락한 경우를 제외하고 다른 사람이 작성하였거나 인터넷 등에서 획득한 과제물, 또는 프로그램 코드의 일부, 또는 전체를 이용하는 것은 부정행위에 해당한다.
  - 자신의 과제물을 타인에게 보여주거나 빌려주는 것은 공정한 평가를 방해하고, 해당 학생의 학업 성취를 저해하는 부정행위에 해당한다.
  - 팀 과제가 아닌 경우 두 명 이상이 함께 과제를 수행하여 이를 개별적으로 제출하는 것은 부정행위에 해당한다.
  - 스스로 많은 노력을 한 후에도 버그나 문제점을 파악하지 못하여 동료의 도움을 받는 경우도 단순한 문법적 오류에 그쳐야 한다. 과제가 요구하는 design, logic, algorithm의 작성에 있어서 담당교수, TA, tutor 이외에 다른 사람의 도움을 받는 것은 부정행위에 해당한다.
  - 서로 다른 학생이 제출한 제출물간 유사도가 통상적으로 발생할 수 있는 정도를 크게 넘어서는 경우, 또는 자신이 제출한 과제물에 대하여 구체적인 설명을 하지 못하는 경우에는 부정행위로 의심받거나 판정될 수 있다.

Problem 1

- Mission: peer review of midterm problems as a group
  - Sign up study group
    - Section 01: https://docs.google.com/spreadsheets/d/1fIRP_15xVKxahA6bGDeifoBvW2upYjgkyi6eU0ch4MQ/edit?usp=sharing
    - Section 02: https://docs.google.com/spreadsheets/d/1JovWOmVhmVbF2A74NX_T7uXxEZrC-oRXlgHeK9r53tQ/edit?usp=sharing
  - Have a meeting to review midterm1, midterm2, and midterm3
    - Review the solutions of other members and share comments to each other.
    - Each member update her/his solution to midterm2 individually applying the comments.
      - Mark the modified parts by comments as the next page.
      - If you believe your previous solution was perfect, submit it again and put a comment “My previous solution was perfect, so I didn't modify it!” in the first line.
    - Review all midterm problems but submit only the updated solution to midterm2.

Problem 1

- Example of updated solution.

```
int main()
{
	int i = 0;

	// previous code
	/*
	<old code>

	*/

	// updated code
	<new code>


	return 0;
}
```

Problem 2

- Search the Internet for the following functions and learn how to use socket in C.
  - struct sockaddr
  - struct sockaddr_in

  - int socket(int domain, int type, int protocol);
  - int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  - int listen(int sockfd, int backlog);
  - int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

  - uint32_t htonl(uint32_t hostlong);
  - uint16_t htons(uint16_t hostshort);
  - uint32_t ntohl(uint32_t netlong);
  - uint16_t ntohs(uint16_t netshort);

Problem 2

- Write a couple of programs that reads digit strings and converts them into alphabet strings.
  - socket_server.c: repeat the followings until it receives "quit"
    - Reads a string from socket_client through socket.
      - if the string is "quit", break the loop
    - Converts the digit string into the corresponding alphabet string.
      e.g., "135246" ➔ "one three five two four six"
    - Send the alphabet string to the client

    Usage: ./socket_server <server-port>

  - socket_client.c: repeat the followings until the user types "quit" instead of a digit string
    - Read a digit string from the user
    - Send the digit string to socket_server through socket
    - Read the conversion result from socket_server
    - Display the input string and the conversion result

    Usage: ./socket_client <server-ip> <server-port> <client-port>

  - Write the two programs by completing socket_server_problem.c and socket_client_problem.c following the instructions

Problem 2 Example

- Server

```
$ ./socket_server 1234
port = 1234
creating socket...
Done.
socket binding...
Done.
listening socket...
Done.
Waiting for connect request...
Connected.

input = [1234]
output = [one two three four]
Sending message...
Done.

input = [872]
output = [eight seven two]
Sending message...
Done.
```

- Client

```
$ ./socket_client 127.0.0.1 1234 4321

Connecting to 127.0.0.1:1234
Done.
Input a number: 1234

1234 ==> one two three four
Input a number: 872

872 ==> eight seven two
```

Problem 2 Example (cont'd)

- Server

```
input = [1004]
output = [one zero zero four]
Sending message...
Done.

input = [1829401]
output = [one eight two nine four zero
one]
Sending message...
Done.


Closing sockets
Done.
```

- Client

```
Input a number: 1004

1004 ==> one zero zero four

Input a number: 1829401

1829401 ==> one eight two nine four
zero one

Input a number: quit
quit ==>
Closing sockets
Done.
```