Name: **Student Number:**

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Mock Lab Test One 2012

Prescription Number: ENCE360

Lab Test Course Title: ***Operating Systems***

Time allowed: 90 minutes

Number of pages: 13

- This exam is worth a total of 20 marks
- Contribution to final grade: 20 %
- Length: 3 questions
- Use this *exam paper* for answering *all* questions. If you need more space, use a separate *Answer Booklet* provided.
- *This is an open book test.* Notes, text books and online resources may be used.
- This open book test is supervised as a University of Canterbury exam. Therefore you cannot communicate with anyone other than the supervisors during the test. Anyone using email or other forms of communication with others will be removed and score zero for this test.
- Please answer *all* questions carefully and to the point. Check carefully the number of marks allocated to each question. This suggests the degree of detail required in each answer, and the amount of time you should spend on the question.

1 Instructions

First reboot your PC (to minimise problems during the test)
Now, write your name and student number at the top of the front page of this paper to avoid scoring 0**.**
This test is open book. You are given 90 minutes. To answer each question you need to write the answer on this exam paper. If you need extra space, you may request an answer book.
To help with your written answers, you may edit the files supplied to you. The examiner not look at the edited files. Your source code files are not marked.
Only this written test paper is marked.
So your source code files are NOT marked.
There are three questions, worth a total of 20 marks.

1.1 Preparation

Log in with your normal user code. At the beginning of the test, the source code files for this test will be available from Learn.
Before the test begins you may check that the files are available, you may not view or edit them until the test has begun.
You should now have the following files:

- "2012 mock lab test.pdf" – this lab test handout
- `one.c` – source code for question one
- `twoServer.c, twoClient.c` – source code for question two
- `threeServer.c, threeClient.c` – source code for question three

If you do not have all these files, then call over an exam supervisor promptly.
These source code files are not marked.

1.2 Comments and code layout

Your source code answers (hand-written on this exam paper) should always be neatly laid out and commented to make it clear to the examiner. **There are a total of FOUR MARKS across all three questions for commenting** in your written answers.
This test paper has semi-complete source code for which your task is to fill out all the gaps (empty boxes) and comment almost every line of the code – both the supplied code and your added lines of code.
Separate source code files are also supplied for each question. But any commenting in these separate source code files is not examined.
Only your answers in this written paper are marked.
Your edited source code files are NOT marked.

Do not turn to the next page until instructed.

Question 1: Mutex (4 Marks)

The child thread, ***`set_data()`*** is setting ***`gobal_data`*** to a random number with ***`rand()`***.
The child thread, ***`read_data()`*** is displaying the value of ***`global_data`***.
Use a mutex to protect the critical region of code accessing this global variable. To enable consistent results, place the `printf()` inside this critical region.
(Hint: use `pthread_mutex_lock()` and `pthread_mutex_unlock()`)

Your task for this question is to complete the code below and **comment almost every line of code**, including both existing and new lines of code.
Source code is in **one.c**
Compile one.c using: `gcc -o one one.c -lpthread`

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

int global_data = 0;

void read_data();
void set_data();

int main()
{
    pthread_t thread1, thread2;
    int thread_return1, thread_return2;
    int i;

    for(i = 0; i < 5; i++) {
            thread_return1 = pthread_create( &thread1, NULL, (void*)&set_data, NULL);
            thread_return2 = pthread_create( &thread2, NULL, (void*)&read_data, NULL);
      }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("exiting\n");
    exit(0);
}
```

```
void set_data()
{
	/* use: global_data = rand(); */
```

```

	pthread_mutex_lock( &mutex1 );
	printf("Setting data\t");
	global_data = rand();
	pthread_mutex_unlock( &mutex1 );

```

```
}

void read_data()
{
	int data;
```

```

	pthread_mutex_lock( &mutex1 );
	data = global_data;
	printf("Data: %d\n", data);
	pthread_mutex_unlock( &mutex1 );

```

```
}
```

//Run the program and write down the results displayed:

```
Setting data	Data: 1481765933
Setting data	Data: 1481765933
Setting data	Data: 1481765933
Setting data	Data: 1481765933
Setting data	Data: 1481765933
```

Question 2: Pipes (12 Marks)

Two named pipes are created for two way communication between a server (twoServer.c) and a client (twoClient.c). One is created by the server and the other is created by the client.

Two unnamed pipes are also created for two way communication between the server and a child process it forks.

Text typed into the client terminal is piped to the server and then to the child process which "compresses" the text by changing it to lower case and stripping out any vowels.

This compressed text is piped back to the parent process and then piped back to the client to be displayed on the client terminal.

enter text → client → server → child(compress text) → server → client → display text

Your task for this question is to complete the code below and **comment almost every line of code**, including both existing and new lines of code.

Source code files are in **twoClient.c** and **twoServer.c**
Compile twoServer.c using: `gcc twoServer.c -o twoServer`
Compile twoClient.c using: `gcc twoClient.c -o twoClient`

To run:
- open two terminals
- in one terminal type: **twoServer**
- in the other terminal type: **twoClient**

(Note: always run twoServer before twoClient)

```c
/****************************** twoClient.c ******************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#define PRE_FILE "PRE_PIPE"
#define POST_FILE "POST_PIPE"
#define BUFSIZE 80

int main(void)
{
      FILE* fp1; // file pointer to named pipe
      FILE* fp2; // file pointer to named pipe
      char buffer[BUFSIZE] = {0};
      char recieved[BUFSIZE] = {0};

      /* open an existing client to server named pipe: */

      fp1 = fopen(PRE_FILE, "w");

      printf("Enter a string to compress (no spaces): ");
      scanf("%s", buffer);
      fputs(buffer, fp1); // pipe text to the server
      fclose(fp1);

      /* create a server to client named pipe (if it does not exist): */

      mknod(POST_FILE, S_IFIFO | 0666, 0);
      fp2 = fopen(POST_FILE, "r");

      fgets(recieved, BUFSIZE, fp2); //read compressed text back from server
      printf("Recieved from server: %s\n", recieved);
      fclose(fp2);

      return 0;
}
```

```c
/***************************** twoServer.c ******************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/wait.h>

#define PRE_FILE "PRE_PIPE"
#define POST_FILE "POST_PIPE"
#define BUFSIZE 80

#define ASCII_A 97
#define ASCII_E 101
#define ASCII_I 105
#define ASCII_O 111
#define ASCII_U 117

int isNotVowel(char c)
{
      /* return 0 if the character is a vowel, otherwise return 1 */

      if (c != ASCII_A && c != ASCII_E && c != ASCII_I && c != ASCII_O && c != ASCII_U)
            return 1;
      else
            return 0;

}

void clearBuffer(char* buff)
{
      int i;
      for(i = 0; i < BUFSIZE; i++)
            buff[i] = '\0';
}
```

```
int main(void)
{
      FILE* fp1;
      FILE* fp2;
      char buffer[BUFSIZE] = {0};
      char modified[BUFSIZE] = {0};

      pid_t childPid;
      int child_status;
      int fd1[2];
      int fd2[2];

      int i,p;

      /* Create a client to server named pipe (if it does not exist): */
      umask(0);
      mknod(PRE_FILE, S_IFIFO | 0666, 0);

      while(1) {
            clearBuffer(buffer);
            clearBuffer(modified);

            fp1 = fopen(PRE_FILE, "r");
            /* read text from client via the named pipe: */

            fgets(buffer, BUFSIZE, fp1);

            fclose(fp1);

            /* create two unnamed pipes for communication between parent and child: */

            if (pipe(fd1) == -1) {
                  perror("pipe");
                  exit(1);
            }

            if (pipe(fd2) == -1) {
                  perror("pipe");
                  exit(1);
            }
```

```
        /* fork a child */
        childPid = fork();
        if(childPid == -1) {
                perror("fork");
                exit(2);
        }

        else if (childPid == 0) { /* in the child code */

                /* read text from a pipe from the parent */

                close(fd1[1]);
                if (read(fd1[0], modified, BUFSIZE) == -1) {
                        perror("Read");
                        exit(3);
                }

                /* compress text: convert all leters to lower case & remove vowels */
                p = 0;
                for(i = 0; i < strlen(modified); i++) {
                        modified[i] = tolower(modified[i]);
                        char c = modified[i];
                        if (isNotVowel(c)) {
                                modified[p] = c;
                                p++;
                        }
                }
                modified[p] = '\0';

                /* pipe compressed text back to the parent: */

                close(fd2[0]);
                if (write(fd2[1], modified, strlen(modified)) == -1) {
                        perror("Write");
                        exit(4);
                }
                /* child is finished so exit */
                exit(0);
        }
```

```
            else { /* in the parent code */

                  /* pipe text to the child: */

                  close(fd1[0]);
                  if (write(fd1[1], buffer, strlen(buffer)) == -1) {
                        perror("Write");
                        exit(5);
                  }

                  /* the child will convert the message...*/

                  /* read compressed text back from the child: */

                  close(fd2[1]);
                  if (read(fd2[0], modified, BUFSIZE) == -1) {
                        perror("Read");
                        exit(6);
                  }
            }
            wait(&child_status);

            /* pipe compressed text back to the client: */
            fp2 = fopen(POST_FILE, "w");

            fputs(modified, fp2);

            fclose(fp2);
      }

      return 0;
}
```

Question 3: Sockets (4 Marks)

Your task for this question is to add error checks to the code below and **comment almost every line of code**, including both existing and new lines of code. Source files are in **threeServer.c** and **threeClient.c**.
To compile: `gcc twoServer.c -o twoServer` and `gcc twoClient.c -o twoClient`
To run: In one terminal type **twoServer** and in the other terminal type **twoClient** (but always run twoServer before twoClient).

threeServer.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define PORT_NUMBER 1234
#define BUFFER_SIZE 512

int main() {
    char buffer[BUFFER_SIZE] = { '\0' };
    struct sockaddr_in server_address, client_address;

    int server_socket = socket(AF_INET, SOCK_STREAM, 0); /* create a socket */
    if (server_socket < 0) {
        printf( "error opening socket" );
        exit(-1);
        }
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT_NUMBER);
    int rc = bind(server_socket, (struct sockaddr *) &server_address,
                                  sizeof(struct sockaddr_in)); /* bind to PORT_NUMBER */
    if(rc == -1) { // Check for errors
        printf( "bind error" );
        exit(-1);
        }
    rc = listen(server_socket, 5);
    if(rc == -1) { // Check for errors
        printf( "listen error" );
        exit(-1);
        }
    socklen_t client_address_length = sizeof(struct sockaddr_in);
    int client_socket = accept(server_socket, (struct sockaddr *) &client_address,
                                   &client_address_length); /* accept client's socket */
    if (client_socket < 0) {
        printf( "error opening client socket" );
        close(server_socket)
        exit(-1);
        }
    int message_length = read(client_socket, buffer, BUFFER_SIZE-1); /* from client */
    if (message_length < 0) {
        printf( "read error" );
    close(client_socket)
    exit(-1);
    }
    printf("from client: %s\n", buffer);
    message_length = sprintf(buffer, "back to ya client");
    message_length = write(client_socket, buffer, message_length);   /* to client */
    if (message_length < 0) {
        printf( "read error" );
        close(client_socket)
        exit(-1);
        }
    close(client_socket);
    return 0;
}
```

threeClient.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define PORT_NUMBER "1234"
#define BUFFER_SIZE 512

int main()
{
    struct addrinfo hints;
      struct addrinfo *server_address = NULL;
    char buffer[BUFFER_SIZE] = {'\0'};

    int socket_fd = socket(AF_INET, SOCK_STREAM, 0); /* create a socket */
    if (socket_fd < 0) {
        printf( "error opening socket" );
        exit(-1);
        }
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    getaddrinfo("localhost", PORT_NUMBER, &hints, &server_address);
    int rc = connect(socket_fd, server_address->ai_addr, server_address->ai_addrlen);
    if (rc < 0) {
        printf( "connect error" );
        close(socket_fd)
        exit(-1);
        }
    int message_length = sprintf(buffer, "hello server");
    message_length = write(socket_fd, buffer, message_length); /* to server */
    if (message_length < 0) {
        printf( "write error" );
        close(socket_fd)
        exit(-1);
        }
    message_length = read(socket_fd, buffer, BUFFER_SIZE-1);   /* from server */
    if (message_length < 0) {
        printf( "read error" );
        close(socket_fd)
        exit(-1);
        }
    printf("from server: %s\n", buffer);
    close(socket_fd);
    return 0;
}
```

END OF PAPER