Name: **Student Number:**

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Lab Test 2016

Prescription Number: **ENCE360**

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

Time allowed: 90 minutes

Number of pages: 14

Question	Marks
1	
2	
3	
Total:	

- 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 separate *Answer Sheets* provided – with your name and student number writing at the top of the page.
- *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 separate *Lab Test Answer Sheets*.
To help with your written answers, you may edit the files supplied to you. The examiner will 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 download the files from Learn and 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:

- "2016 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 that you understand the code and concepts**.

This test paper has semi-complete source code for which your task is to fill in all the gaps (empty boxes). Add the minimum code possible – so do not add error checking code. (You are being tested on your understanding of concepts and usage of functions, not error checking skills.)

There are a total of THREE MARKS across all three questions for commenting in your written answers. So 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: Threads 1 (4 marks)

If the main parent thread creates a child thread and then this main parent thread finishes before its child thread ends, does this automatically kill the child thread – or does that child thread continue regardless of when the main parent thread ends?
This is the question that one.c seeks to answer by initially having a one second delay in the main parent thread, during which time you **print "Hello World"** inside the child thread. Then you introduce a **five second delay** in the child thread, before printing in the child thread and observe the difference.
(Hint: there is a big difference between waiting 5 seconds versus waiting 1 second for a result.)

Your task for this question is to complete the code below (in the empty boxes) using as few lines of code as possible (e.g. no error checking). Comment almost every line of code; including both existing and new lines of code. 1 of the 4 marks is for commenting.

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>

void *print_message_function( void *ptr );

int main()
{
```

```
        sleep(1);
        return(0);
}

void *print_message_function( void *ptr )
{
        printf("child thread\n");
```

```
        pthread_exit(NULL);
}
```

Run the program, with and without a 5 second delay in the child thread - write down the results displayed and then answer the question:
"If the main parent thread finishes before a child thread, does that child thread still continue?"

Question 2: Pipes (10 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 (in the empty boxes) using as few lines of code as possible (e.g. no error checking). Comment almost every line of code; including both existing and new lines of code. 1 of the 10 marks is for commenting.

Source code files are **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>
#include <string.h>

#define PRE_FILE "PRE_PIPE" // client to server pipe
#define POST_FILE "POST_PIPE" // server to client pipe
#define BUFSIZE 80

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

	/* open an existing client-to-server named pipe (already created in server): */
	fp1 = fopen(PRE_FILE, "w");

	printf("Enter a string to compress (no spaces): ");
	scanf("%s", buffer);
	// pipe this entered text to the server via "PRE_PIPE"
```

```
```

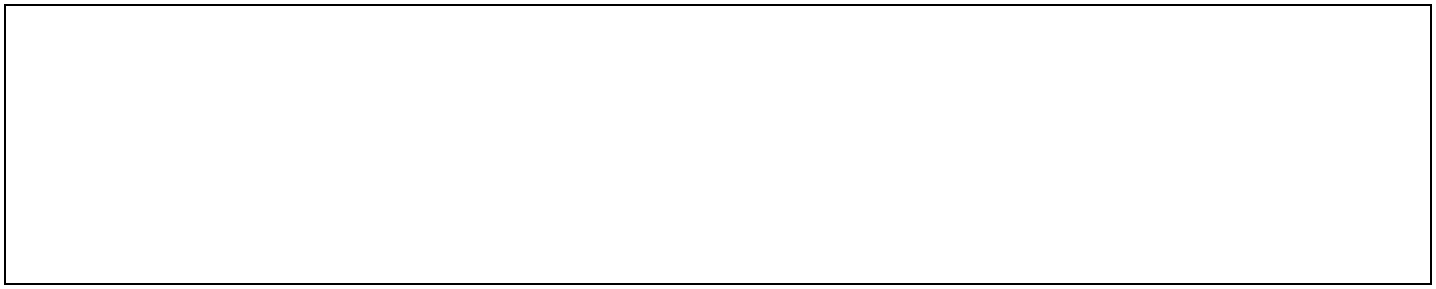
```c
	fclose(fp1);

	/* create a server to client named pipe: */
	mknod(POST_FILE, S_IFIFO | 0666, 0);
	fp2 = fopen(POST_FILE, "r");
	// read compressed text back from server via “POST_PIPE”
```

```
```

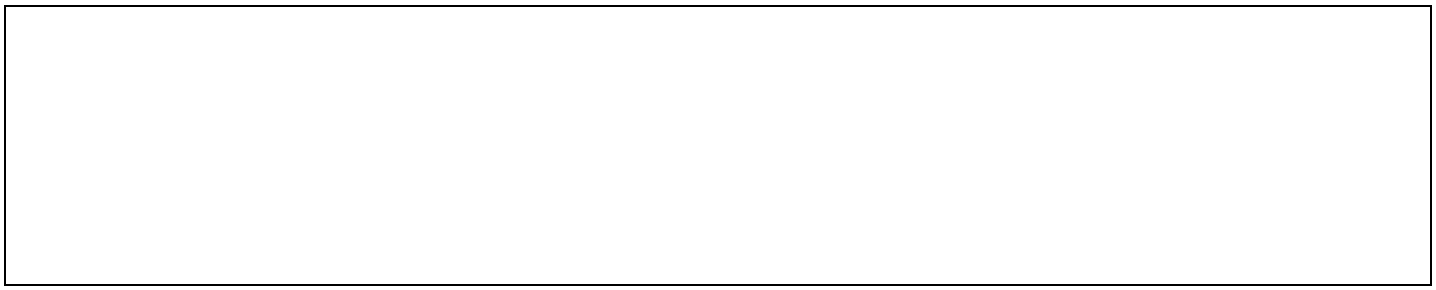
```c
	printf("Recieved from server: %s\n", recieved);
	fclose(fp2);

	return 0;
}
```

```c
/***************************** twoServer.c ******************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

#define PRE_FILE "PRE_PIPE" // client to server pipe
#define POST_FILE "POST_PIPE" // server to client 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 main(void)
{
	FILE* fp1; // file pointer to named pipe, "PRE_PIPE" (client to server)
	FILE* fp2; // file pointer to named pipe, "POST_PIPE" (server to client)

	char buffer[BUFSIZE] = {0};
	char modified[BUFSIZE] = {0};

	pid_t childPid;
	int child_status;
	int fd1[2]; // parent-to-child unamed pipe
	int fd2[2]; // child-to-parent unamed pipe
	int i,p;

	/* 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 */
```

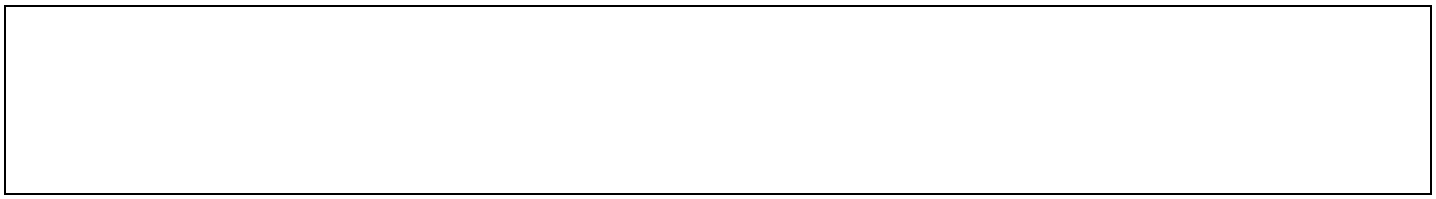


```
      /* compress text: convert all leters to lower case & remove vowels */
      for(p=0, i=0; i < strlen(modified); i++) {
            char c = tolower(modified[i]);
            if (c != ASCII_A && c != ASCII_E && c != ASCII_I && c != ASCII_O && c != ASCII_U)
                  modified[p++] = c; // not a vowel
      }
      modified[p] = '\0';

      /* write compressed text back to a pipe to the parent: */
```

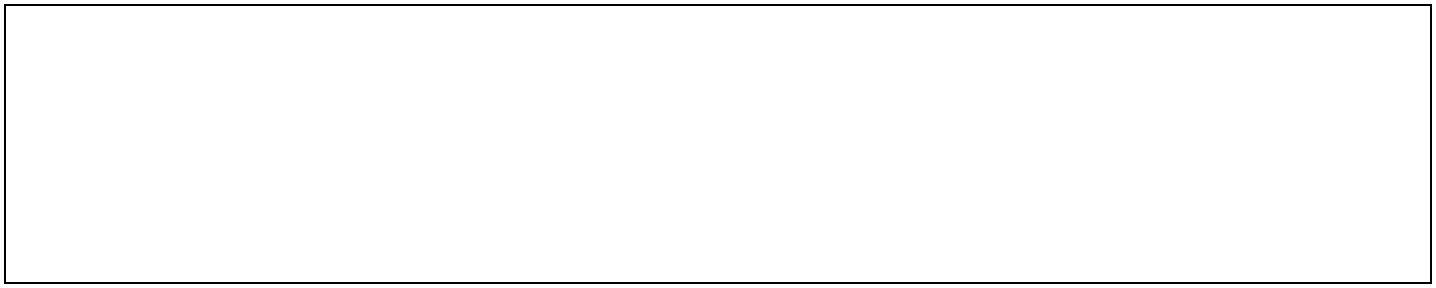


```
      /* child is finished so exit */
      exit(0);
}
```

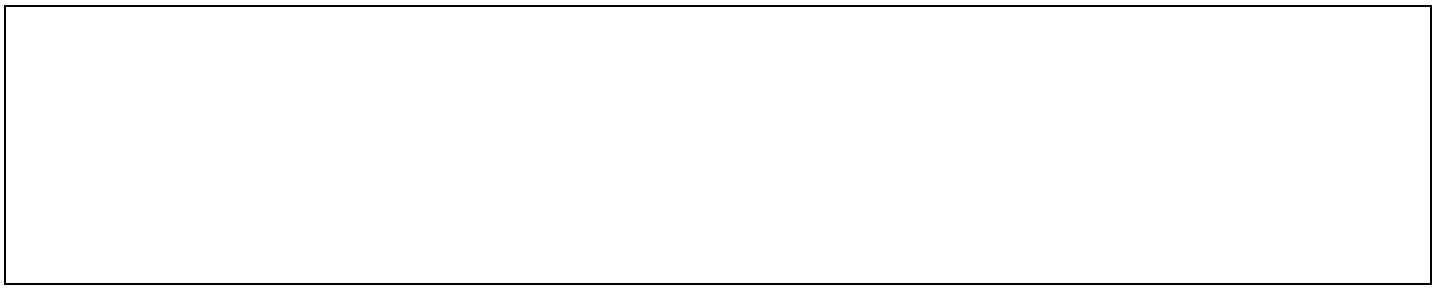
```
        else { /* in the parent code */

            /* Create and open a client to server named pipe: */
            mknod(PRE_FILE, S_IFIFO | 0666, 0);
            fp1 = fopen(PRE_FILE, "r");
            /* read text from client via the named pipe: */
```



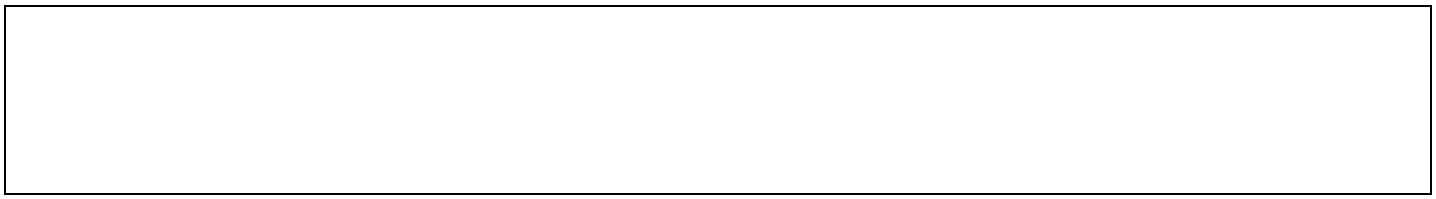
```
            /* pipe text to the child: */
```



```
            /* the child will convert the message...*/
            /* read compressed text back from the child: */
```



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



```
        }

    return 0;
}
```

Question 3: Sockets (6 Marks)

Below is the code for a socket server program and a socket client program. To simplify the main programs, socket()→bind()→listen()→accept() have been merged into a single "server_socket()" function and socket()→connect() have been merged into a single "client_socket()" function.

Your task for this question is to complete the code below (in the empty boxes) using as few lines of code as possible (e.g. no error checking). Comment almost every line of code, including both existing and new lines of code (1 of the 6 marks is for commenting).

Source code files are **threeClient.c** and **threeServer.c**
Compile threeServer.c using: `gcc threeServer.c -o threeServer`
Compile threeClient.c using: `gcc threeClient.c -o threeClient`
To run:

- open two terminals
- in one terminal type: **threeServer 1234**
- in the other terminal type: **threeClient localhost 1234**

(Note: always run threeServer before threeClient)

---

threeServer.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>

int server_socket(char *port);


int main(int argc, char *argv[])  {
    if (argc != 2) {
            fprintf(stderr,"usage: threeClient hostname port-number\n");
            exit(1);
    }
    printf("\nThis is the server with pid %d listening on port %s\n",getpid(),argv[1]);

    int sockfd = server_socket(argv[1]);

    char* message = "congrats you successfully connected to the server!\n";
    write (sockfd, message, strlen(message));

    close(sockfd);
    exit (0);
}
```

```
int server_socket(char *port)

// execute socket(), bind(), listen(), accept() and return the new socket from accept()

{
```

```
}
```

threeClient.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>

#define MAXDATASIZE 1024

int client_socket(char *hostname, char *port);



int main(int argc, char *argv[])
{
    char buffer[MAXDATASIZE];

    if (argc != 3) {
            fprintf(stderr,"usage: threeClient hostname port-number\n");
            exit(1);
    }

    // connect client socket to server:
   int sockfd = client_socket(argv[1], argv[2]);

    int numbytes = read(sockfd, buffer, MAXDATASIZE-1);

    buffer[numbytes] = '\0';
    printf("\nReceived from server: %s\n\n",buffer);

    close(sockfd);

    return 0;
}
```

```
int client_socket(char *hostname, char *port)
// execute socket() and connect() and return the socket ID
{

}
```

END OF PAPER