Started on	Wednesday, 21 August 2019, 11:23 AM
State	Finished
Completed on	Wednesday, 21 August 2019, 1:05 PM
Time taken	1 hour 41 mins
Grade	**18.33** out of 20.00 (**92**%)

Information

- This exam is worth a total of 20 marks
- Contribution to final grade: 20 %
- Length: 8 questions
- 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, Facebook 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.

Question **1**

Correct

Mark 3.00 out of 3.00

Question 1: Threads (3 Marks)

The child thread, ***set_data()*** is setting ***global_data*** to a random number with ***rand()*** .
The child thread, ***read_data()*** is displaying the value of ***global_data*** .
Use a semaphore to protect the critical region of code accessing this global variable.

You need to initialise the semaphores, ***read*** and ***write*** to their respective values, and then use ***sem_wait*** and ***sem_post*** to do reading and writing in the critical region.

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).

Source code is in one.c

You can use the following command to compile one.c

```
gcc one.c -o one -lpthread
```

This source code is exactly the same as provided in one.c, and is here for your reference.

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

#define NUM_THREADS 10

typedef struct {

        // Read and write semaphores for our channel
        sem_t read;
        sem_t write;

        // Global shared data
        int global_data;

} Channel;

void read_data();
void set_data();
void init_channel();

int main()
{
        pthread_t threads[NUM_THREADS];
        Channel channel;
        int i;

        srand(2018);
        init_channel(&channel);

        for(i = 0; i < NUM_THREADS; i += 2) {
                pthread_create(&threads[i], NULL, (void*)&set_data, &channel);
                pthread_create(&threads[i+1], NULL, (void*)&read_data, &channel);
        }

        // wait for threads to finish before continuing
        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////

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

void set_data(Channel *channel)
{
        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////
        assert(channel->global_data == 0);

        printf("Setting data\t");
        channel->global_data = rand();

        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////
}

void read_data(Channel *channel)
{
        int data;

        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////

        data = channel->global_data;
        channel->global_data = 0;
        printf("Data: %d\n", data);

        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////
}

void init_channel(Channel *channel)
{

        // Initialise count of the read semaphore to 0 (there's nothing to read yet)
        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////

        // Note the write semaphore is initialised to 1 (channel is empty, free for writing)
        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////

        channel->global_data = 0;

}
```

Answer: (penalty regime: 10, 20, ... %)

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

#define NUM_THREADS 10

typedef struct {

    // Read and write semaphores for our channel
    sem_t read;
    sem_t write;

    // Global shared data
    int global_data;

} Channel;

void read_data();
void set_data();
```

	Expected	Got	

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Question 2 - Semaphores (1 Mark)

In the context of Question 1, if ***global_data*** was to become an array that could hold **multiple elements**, and had a known fixed size of 10 elements (i.e. we know the array is **empty** at **0**, and the array is **full** at **10** elements), and we can only **read** or **write one** element at a time, do we need **one set** (read & write) of semaphores, or do we need **ten sets** (10x read & write) semaphores to organise concurrency?

Penalties are set at 33.33% per submission.

Select one:

- [✓ a. We only need one set of semaphores. We can sem_post the read semaphore until it hits 10, and sem_wait the write semaphore above 0. Semaphores are not mutexes and can take any number.] ✓
- b. We need ten sets of semaphores. Semaphores act exactly like mutexes and can only hold 0 or 1, so we cannot increment a single semaphore to 10. We just use semaphores to ensure one action (like writing) happens before another action (like reading).
- c. We only need one single semaphore for this task. The semaphore can be incremented and decremented more than once. Say, if there are 6 writes to the array, the semaphore is at 6, and can be read from 6 times. The semaphore can be a maximum of 10 and minimum of 0.
- d. We only need ten single semaphores here. If semaphore #1 is 1, then data can be read from the array at index 0, and if semaphore #1 is 0, then data can be written to the array at position 0.

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 5.00 out of 5.00

Question 3: Pipes (5 Marks)

Here, two processes are communicating both ways by using two pipes. One process reads input from the user and handles it. The other process performs some translation of the input and hands it back to the first process for printing.

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).

Source code is in two.c

You can use the following command to compile two.c

```
gcc two.c -o two
```

This source code is exactly the same as provided in two.c, and is here for your reference.

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


void translator(int input_pipe[], int output_pipe[])
{
        int c;
        char ch;
        int rc;

        /* first, close unnecessary file descriptors */
        // place your code between the lines of //
        ///////////////////////////////////////////////


        ///////////////////////////////////////////////

        /* enter a loop of reading from the user_handler's pipe, translating */
        /* the character, and writing back to the user handler.              */
        while (read(input_pipe[0], &ch, 1) > 0) {
                c = ch;
                if (isascii(c) && isupper(c))
                        c = tolower(c);
                ch = c;

                /* write translated character back to user_handler. */
                // place your code between the lines of //
                ///////////////////////////////////////////////


                ///////////////////////////////////////////////

                if (rc == -1) {
                        perror("translator: write");
                        close(input_pipe[0]);
                        close(output_pipe[1]);
                        exit(1);
                }
        }

        close(input_pipe[0]);
        close(output_pipe[1]);
        exit(0);
}



void user_handler(int input_pipe[], int output_pipe[])
{
        int c;    /* user input - must be 'int', to recognize EOF (= -1). */
        char ch;  /* the same - as a char. */
        int rc;   /* return values of functions. */

        /* first, close unnecessary file descriptors */
        ///////////////////////////////////////////////
        // place your code between the lines of //


        ///////////////////////////////////////////////

        printf("Enter text to translate:\n");
        /* loop: read input from user, send via one pipe to the translator, */
        /* read via other pipe what the translator returned, and write to   */
        /* stdout. exit on EOF from user.                                   */
        while ((c = getchar()) > 0) {
                ch = (char)c;

                /* write to translator */
                ///////////////////////////////////////////////
                // place your code between the lines of //


                ///////////////////////////////////////////////

                if (rc == -1) { /* write failed - notify the user and exit. */
                        perror("user_handler: write");
                        close(input_pipe[0]);
                        close(output_pipe[1]);
                        exit(1);
                }

                /* read back from translator */
                ///////////////////////////////////////////////
                // place your code between the lines of //


                ///////////////////////////////////////////////

                c = (int)ch;
                if (rc <= 0) {
                        perror("user_handler: read");
                        close(input_pipe[0]);
                        close(output_pipe[1]);
                        exit(1);
                }

                putchar(c);
                if (c=='\n' || c==EOF) break;
        }

        close(input_pipe[0]);
        close(output_pipe[1]);
        exit(0);
}




int main(int argc, char* argv[])
{
        int user_to_translator[2];
        int translator_to_user[2];
        int pid;
        int rc;

        rc = pipe(user_to_translator);
        if (rc == -1) {
                perror("main: pipe user_to_translator");
                exit(1);
        }

        rc = pipe(translator_to_user);
        if (rc == -1) {
                perror("main: pipe translator_to_user");
                exit(1);
        }

        pid = fork();

        switch (pid) {
        case -1:
                perror("main: fork");
                exit(1);
        case 0:
                translator(user_to_translator, translator_to_user); /* line 'A' */
                exit(0);
        default:
                user_handler(translator_to_user, user_to_translator); /* line 'B' */
        }

        return 0;
}
```

For example:

Input	Result

`I wish YOU a HAPPY new YEAR`	`Enter text to translate:`
	`i wish you a happy new year`

Answer: (penalty regime: 10, 20, ... %)

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

```
#include <unistd.h>
#include <ctype.h>


void translator(int input_pipe[], int output_pipe[])
{
    int c;
    char ch;
    int rc;

    /* first, close unnecessary file descriptors */
    // place your code between the lines of //
    //////////////////////////////////////////////
    close(input_pipe[1]);
    close(output_pipe[0]);

    //////////////////////////////////////////////

    /* enter a loop of reading from the user_handler's pipe, translating */
    /* the character, and writing back to the user handler.              */
```

	Input	Expected	Got	
✓	I wish YOU a HAPPY new YEAR	Enter text to translate: i wish you a happy new year	Enter text to translate: i wish you a happy new year	✓

Passed all tests! ✓

Correct

Marks for this submission: 5.00/5.00.

Question **4**

Correct

Mark 0.00 out of 1.00

Question 4 - Pipes (1 Mark)

If we are using a pipe to communicate between a parent and a child process, with the parent writing data for the child to read, why must we always close the ends of the pipe that we are not using?

Penalty is 33.33% per submission.

Select one:

[✓ a. Processes read from a pipe until EOF (End of File) is received, and since the file descriptors are duplicated between the parent and the child, if the child does not close the write end, EOF will never be sent, and the child will read forever.] ✓

b. The kernel buffers data from the parent to the child, and since the file descriptors are duplicated between the parent and the child, the kernel will not know what process the data is meant for, and neither process will be able to read data.

c. The pipe command creates a one way pipe, and data can only flow in one direction. However, C is a flexible programming language, which enables us to decide what way the data will flow. We tell the kernel the direction by closing ends of the pipe.

d. Pipes are virtual file descriptors which must be opened and closed to use. We open the end we want to use, and close the other to signal the kernel the channel data flows across.

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.00/1.00**.

Question **5**

Correct

Mark 4.00 out of 4.00

Question 5: Sockets - Server (4 Marks)

Question 5 and Question 7 are about implementing a server and client program which communicate over sockets.

Question 5 relates to completing a socket server program. Only submit the socket server in this question.

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).

Source code is in threeServer.c

You can use the following command to compile threeServer.c

```
gcc threeServer.c -o threeServer
```

You can use the following commands to run:

- open two terminals
- in one terminal type **./threeServer 1234**
- in the other terminal type **./fourClient localhost 1234**

Note to test, you need to also complete the client from Question 7. Always run threeServer first before fourClient.

This source code is exactly the same as provided in threeServer.c, and is here for your reference.

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

#define MAXDATASIZE 1024


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

        if (argc != 2) {
                fprintf(stderr,"usage: threeServer port-number\n");
                exit(1);
        }

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        struct sockaddr_in sa, caller;
        sa.sin_family = AF_INET;
        sa.sin_addr.s_addr = INADDR_ANY;
        sa.sin_port = htons(atoi(argv[1]));

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        //////////////////////////////////////////////////////

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        socklen_t length = sizeof(caller);
        // place your code between the lines of //
        /////////////////////////////////////////////////////////////


        /////////////////////////////////////////////////////////////

        char message[MAXDATASIZE] = "congrats you successfully connected to the server!";
        while (strlen(message) > 0)
        {
                int numbytes; // number of bytes of data read from socket

                // send data to the client and then get data back from the client:
                // place your code between the lines of //
                ///////////////////////////////////////////////////////


                ///////////////////////////////////////////////////////

                message[numbytes - 1] = '\0';
        }

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        exit (0);
}
```

Answer: (penalty regime: 10, 20, ... %)

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

```
#include <unistd.h>

#define MAXDATASIZE 1024


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

    if (argc != 2) {
        fprintf(stderr,"usage: threeServer port-number\n");
        exit(1);
    }

    // place your code between the lines of //
    ///////////////////////////////////////////////////////
    int s = socket(AF_INET, SOCK_STREAM, 0);                       // Create socket

    ///////////////////////////////////////////////////////
```

	Expected	Got	

Passed all tests! ✓

Correct

Marks for this submission: 4.00/4.00.

Question **6**

Correct

Mark 1.00 out of 1.00

Question 6 - Sockets - Server (1 Mark)

What function call lets the server know that a client is connecting? Is it **listen()** or **accept()**?

Penalty is 33.33% per submission.

Select one:

- a. Neither. listen() is used to tell the kernel to start listening for connections, but doesn't return anything so it cannot tell us if a client is connecting. accept() only works if we know there is a connection to be made, as it only processes the last stage of communication by opening a file descriptor.
- b. listen() and accept() can both do this. Both listen() and accept() can determine when a client is connecting, and can let the process know to open a new file descriptor for communication.
- [✓ c. accept(). accept() blocks until a client connects to the server, and then returns a new file descriptor which represents the new connection made. listen() simply enables the process to start listening for new connections, but does not actually tell the process that an connection attempt has been made.] ✓
- d. listen(). listen() tells the kernel that we are interested in connections to this socket, and is the first to know when a connection is incoming, so we can call accept() to accept the connection and get a new file descriptor.

Correct

Marks for this submission: 1.00/1.00.

Question **7**

Correct

Mark 4.00 out of 4.00

Question 7: Sockets - Client (4 Marks)

Question 5 and Question 7 are about implementing a server and client program which communicate over sockets.

Question 7 relates to completing a socket client program. Only submit the socket client in this question.

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).

Source code is in fourClient.c

You can use the following command to compile fourClient.c

```
gcc fourClient.c -o fourClient
```

You can use the following commands to run:

- open two terminals
- in one terminal type **./threeServer 1234**
- in the other terminal type **./fourClient localhost 1234**

Note to test, you need to also complete the server from Question 5. Always run threeServer first before fourClient.

This source code is exactly the same as provided in fourClient.c, and is here for your reference.

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

#define MAXDATASIZE 1024


int main(int argc, char *argv[])
{
        if (argc != 3) {
                fprintf(stderr,"usage: fourClient hostname port-number\n");
                exit(1);
        }

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        struct addrinfo their_addrinfo; // server address info
        struct addrinfo *their_addr = NULL; // connector's address information
        memset(&their_addrinfo, 0, sizeof(struct addrinfo));
        their_addrinfo.ai_family = AF_INET;        /* communicate using internet address */
        their_addrinfo.ai_socktype = SOCK_STREAM;  /* use TCP rather than datagram */
        getaddrinfo(argv[1], argv[2], &their_addrinfo, &their_addr); /* get IP address info for the
hostname (argv[1]) and port (argv[2]) */

        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        char buffer[MAXDATASIZE]; //buffer contains data from/to server
        int numbytes; // number of bytes of data read from socket
        // get data from the server:
        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        while (numbytes > 0)
        {
                buffer[numbytes-1] = '\0';
                printf("%s\n", buffer); //print out what you received

                // send data to the server and then get data back from the server:
                // place your code between the lines of //
                ///////////////////////////////////////////////////////


                ///////////////////////////////////////////////////////
        }
        // place your code between the lines of //
        ///////////////////////////////////////////////////////


        ///////////////////////////////////////////////////////

        return 0;
}
```

Answer: (penalty regime: 10, 20, ... %)

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

#define MAXDATASIZE 1024


int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr,"usage: fourClient hostname port-number\n");
        exit(1);
    }

    // place your code between the lines of //
    ///////////////////////////////////////////////////////
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);                    // Create the socket

    ///////////////////////////////////////////////////////

```

Expected	Got

Passed all tests! ✓

Correct

Marks for this submission: 4.00/4.00.

Question **8**

Correct

Mark 0.33 out of 1.00

Question 8 - Sockets - Client (1 Mark)

We see that one of the arguments to **connect()** is the server port number. But hang on, we never set any of the arguments to the clients port number. How does the server know what port to connect back to the client on?

Penalty is 33.33% per submission.

Select one:

- a. The client simply uses the same port number as the server port to receive connections back from the server.
- [✓ b. The system assigns its own port number itself when connect() is called, and simply selects a port which isn't being used.] ✓
- c. The client opens a communication channel to the server using the server port, and the server simply replies over this channel. The client never has to bind any sockets at all to communicate, so there is no need to worry about this.
- d. We choose a port number by writing code to bind() and listen() before any calls to connect() are made. We then pass this information to the server with a call to accept() to show what port we want to use.

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.33/1.00**.

◄ For fun quiz   Jump to...   ENCE360 Mid Term Test 2019 ►