UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 6 Examination in Engineering: September 2023

Module Number: EE6206 **Module Name: Operating Systems Programming (C-18)**

[Three Hours]

[Answer **all questions**, each question carries 10 marks.]

(Q1)

a) Briefly explain the purpose of following Linux terminal commands?

(i)
```
sudo init 0
```
(ii)
```
sudo init 6
```
(iii)
```
gnome-session-quit
```
(iv)
```
cat Doc1.txt>>Doc2.txt
cat Doc2.txt
```
(v)
```
cat Doc1.txt>Doc2.txt
cat Doc2.txt
```
(vi)
```
cat Doc1.txt Doc2.txt
```
(vii)
```
cat >Doc1.txt
ctrl+d
cat >Doc2.txt
ctrl+d
```
(viii)
```
gedit &
```
(ix)
```
sudo adduser user1
```
(x)
```
sudo apt-get install build-essential
```

[5 marks]

b) Write Linux console-based commands for the following sequence of operations?

(i) Go to user's **home** directory. Change the directory to **Documents**. Display the current working directory. Note that **Documents** is a directory inside the home directory.

(ii) Assuming there is a text file known as **txt1.text** inside **Documents** directory, create an exact copy of **txt1.text** as **txt2.text** in the previous (**home**) directory.

(iii) Go back to the **home** directory and copy the **Downloads** directory to inside the **Desktop** directory as **Downloads_copy**. Note that **Downloads and Desktop** are directories inside the home directory.

(iv) Delete **Downloads_copy**, go to the **Home** directory and list the details of **txt2.text** file in long format.

(v) Remove the write privilege of the other users (others) for **txt2.text** and add read privilege to users in File's group.

(vi) Create a directory named **mydir2** inside **Downloads** directory. Go to that directory. After that create two empty text files as **Doc1.txt, Doc2.txt** inside **mydir2** using commands

(I) **touch** command

(II) **cat** command

(vii) Assuming both **Doc1.txt** and **Doc2.txt** has content inside them, copy the content of **Doc1.txt** to **Doc2.txt** and display **Doc2.txt**.

(viii) Observe the statistics of **Doc1.txt.** Change the access time of **Doc1.txt.** Change the modification time of **Doc1.txt**

(ix) Assume there is a C program file called **program_1.C**. Write instructions to compile and run an object file with same name as **program_1.C**. Note that **program_1.C** contains header files for mathematical computations and multi-threading.

[5 marks]

(Q2)

a)

(i) State the meaning of a recursive function in programming.

[1 mark]

(ii) State two (2) differences between structures and unions in C language.

[1 mark]

(iii) Consider the code snippet given in Figure Q2(a). State whether **`function1`** implements pass by reference or pass by value parameter passing.

[1 mark]

(iv) Write a C program to print 10 double precision floating point numbers from the console using arrays and loops without using functions. Include all header files in your program.

[2 marks]

b)

(i) State the corresponding function used for creating a new file or open an existing file in each of Low level I/O and High level I/O models.

[1 mark]

(ii) An incomplete code segment to read from 15th byte to 30th byte and 40th byte to 50th byte from a doc1.txt and write to doc2.txt using low level file I/O is given in Figure Q2(b). Complete the missing lines of the code A to J.

[4 marks]

(Q3)

a)

(i) State the 5 segments in virtual memory.

[1 mark]

(ii) State two (2) differences and two (2) similarities between pipes and message queues.

[1 mark]

(iii) State the meaning of the following statement.
`waitpid(pid1, NULL, 0);`

[1 mark]

(iv) Define the structure of a message in a message queue

[1 mark]

(v) The function to receive a message in a message queue is defined as follows.

```
ssize_t  msgrcv (int msqid, const void * msgp, size_t
msgsz, long msgtype, int msgflag)
```

Explain the retrieval modes of the message queue under following cases for the msgtype: positive, zero, and negative.

[1 mark]

(vi) Pipe function is defined as follows.

```
int pipe(int * file_des)
```

Briefly explain the purpose of the argument *file_des*

[1 mark]

b) Complete the missing lines A-J in the C program given in Figure Q3(b) to implement the following scenario.

Use shared memory to transfer the message "**message from parent P1**" from parent process to child process **C1**. **C1** has a Child **C2**. When message from P1 is received to **C1**; **C1** will respond P1 with "**response from child C1**". At the same time C1 must send "**message from parent C1**" to **C2**. When **C2** receives that message, it should respond to **C1** with "**response from child C2**". **P1** and **C1** must exit after responses from child are heard. C2 must terminate after sending response to **C1**.

[4 marks]

(Q4)

a)

(i) Define what is a thread and a process separately.

[1 mark]

(ii) Specify two (2) differences and two (2) similarities between threads and processes.

[1 mark]

(iii) State 3 ways that can be used for Inter process communication and 2 techniques of thread synchronization.

[1 mark]

(iv) Define simultaneous multi-threading.

[1 mark]

(v) Briefly explain what is meant by race conditions in hyper-threading.

[1 mark]

(vi) Briefly explain how a Mutex can be used to avoid race conditions

[1 mark]

b) Complete the missing lines A-J in the incomplete C program given in Figure Q4(b) to implement the following scenario.

Create a hyper-threaded process with two (2) simultaneous threads. One thread should write as "**T1**" repeatedly forty (40) times and the other thread should write as "**T2**" repeatedly forty (40) times. Thread synchronization should be as follows. When **Thread1** executes, it should print the word "**T1**" ten (10) times and after that wait until the other thread prints "**T2**" five (5) times before **Thread1** prints again. After **Thread1** has finished printing; **Thread2** can print continuously. Use Mutex and condition variables for this program.

[4 marks]

(Q5)

a)

(i) State the two types of socket domains that can be used for inter process communication.

[1 mark]

(ii) List down the 7 attributes of a socket.

[1 mark]

(iii) State and briefly explain the two types of byte orders.

[1 mark]

(iv) The connect( ) function called in the client is defined as follows.

```
int  connect(int  socket_fd, const struct sockaddr *
sockptr, int   addrlen_sockptr)
```

Briefly explain the connect function using the arguments given.

[1 mark]

(v) Briefly explain the meaning of the following two commands,

(I) cat /etc/hosts

(II) ifconfig

[1 mark]

(vi) State and briefly explain the server only functions in the correct sequence which they should be called in a server machine in client-server communication.

[1 mark]

b)

(i) Complete the missing lines A-J in the incomplete C program given in Figure Q5(b) to implement the following scenario.

Create a client program to connect to the **echo** service of a remote machine to send a user specified string of user specified size and display the returned string on screen. Include error handling for sending only. The dotted decimal notation of the IPv4 address of the remote machine is **134.56.90.12**. The socket's domain is internet and its service type is stream. After receiving the reply from the remote machine, the client program must make sure that the socket is disabled for reading only for any process which has been connected to it.

[4 marks]

```
void function1(int * x, int * y)
{
        *x = *x + *y;

}

int main( )
{
        function1(&val1,&val2);
        return (0);
}
```

Figure Q2(a): Code snippet to implement a function

```
#include <stdio.h>
          (A)
-----------------
#include <fcntl.h>
#include <sys/stat.h>
          (B)
-----------------
#include <stdlib.h>
int main()
```

```
{
        int fd1;
              C
        ---------------------
        if(fd1 == -1)
        {
                          D
                --------------------------
                printf("error %d occured while opening file",errno),
                exit(0);
        }
        else
        {
                           E
                ----------------------------------------
                if(fd2 == -1)
                {
                                 F
                        --------------------------
                        printf("error %d occured while opening file",errno),
                        exit(1);
                }
                else
                {
                        char buf1[16];
                        char buf2[11];
                        char buf3[3] = "   ";
                                 G
                        --------------------------
                        read(fd1,buf1,16);
                                 H
                        --------------------------
                                 I
                        --------------------------
                        write(fd2,buf1,16);
                        write(fd2,buf3,3);
                                  J
                ---------------------
            }
        }
        return(0);
}
```

Figure Q2(b): Incomplete code snippet to read from 15th byte to 30th byte and 40th byte to 50th byte from a doc1.txt and write to doc2.txt using low level file I/O

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/msg.h>
          (A)
---------------------
          (B)
---------------------
#include <sys/stat.h>
int main()
{
        key_t key_for_SM1 = ftok("file1.text", 56);
                          (C)
        -------------------------------------
        int SMID1 = shmget(key_for_SM1,4096,IPC_CREAT | 0666);
                          (D)
        -------------------------------------
        pid_t PID1;
        switch(PID1 = fork())
        {
                case -1:
                {
                        perror("Child1 error");
        exit(0);
        break;
    }
    case 0:
    {
        pid_t PID2;
        switch(PID2 = fork())
        {
            case -1:
            {
                perror("Child2 error");
                exit(0);
                break;
            }
            case 0:
            {
                char bufc2[50];
                char * shmptrc2;
                            (E)
                ----------------------------
                strcpy(bufc2,shmptrc2);
                printf("\n C2 received: %s", shmptrc2);
                char msgc2[] = "Response from child C2";
                            (F)
                -----------------------------
                printf("\n C2 sent: %s", shmptrc2);
                exit(0);
                break;
            }
            default:
            {
                    char * shmptrc1a;

                    char * shmptrc1b;

                    char bufc1a[50];
                              G
                    ----------------------------------

                    shmptrc1b =
                (char*)shmat(SMID2,NULL,SHM_W|SHM_R);

                    strcpy(bufc1a, shmptrc1a);

                    printf("\n C1 received: %s", shmptrc1a);

                    char msgc1a[] = "Response from child C1";

                    char msgc1b[] = "Message from parent C1";
                              H
                    ----------------------------------

                    strcpy(shmptrc1b,msgc1b);
                    printf("\n C1 sent: %s", shmptrc1a);
                    printf("\n C1 sent: %s", shmptrc1b);
                    waitpid(PID2,NULL,0);
                    char bufc1b[50];
                    strcpy(bufc1b,shmptrc1b);
                    printf("\n C1 received: %s", shmptrc1b);
                    exit(0);
                    break;

                }

            break;

            }

        default: {

            getpid();

            char msgp1[] = "message from parent p1";

            char * shmptrp1;
                        I
            ------------------------------------------
                        J
            ------------------------------------------

            printf("\n P1 sent: %s ", shmptrp1);

            sleep(0.01);
```

```
                    char bufp1[50];

                    strcpy(bufp1,shmptrp1);

                    printf("\n P1 received: %s",shmptrp1);

                    exit(0);

                    break;

              }

        return(0); }
```

Figure Q3(b): Incomplete code for inter-process communication using shared memory.

```
#include <stdio.h>
--------Ⓐ--------

void * thread_function1(void * argumes);
void * thread_function2(void * argumes);
-----------Ⓑ---------------------------
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;


int main()
{

        pthread_t tid1,tid2;
        -----------Ⓒ--------------
        pthread_create(&tid2,NULL,thread_function2, NULL);

        pthread_join(tid1,NULL);
        ------------Ⓓ-------
        pthread_mutex_destroy(&mutex1);
        return(0);
}


void * thread_function1(void * argumes)
{
        for(int i=0;i<4;i++)
        {
                pthread_mutex_lock(&mutex1);
                if (i != 0)
                {
                        ----------E----------
                }
                for(int j=0; j<10; j++)
                {
                        ----------F----------

                }
                ---------------G---------------

                sleep(1);

                pthread_cond_signal(&cond1);
        }
        pthread_exit(NULL);

}


void * thread_function2(void * argumes)
{
        for(int i=0; i<8; i++)
        {
                ---------------H---------------
                if(i<4)
                {
                        pthread_cond_wait(&cond1,&mutex1);

                }

                for(int j=0; j<5; j++)
                {
                        ----------I----------

                }

                pthread_mutex_unlock(&mutex1);
                sleep(1);
                ---------------J---------------
        }
        pthread_exit(NULL);

}
```

Figure Q4(b): Incomplete code for thread synchronization.

```
#include <stdio.h>

#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
          Ⓐ
-----------------------
#include <sys/types.h>
#include <netinet/in.h>
          Ⓑ
-----------------------

#include <string.h>

int main()
{
        int socket_fd;
                        Ⓒ
        ------------------------------
        struct sockaddr_in remote_ad;
        remote_ad.sin_family = AF_INET;
        remote_ad.sin_port = htons(7);
                        Ⓓ
        ------------------------------;
        memset(&remote_ad.sin_zero,'\0',sizeof(remote_ad.sin_zero));
                        Ⓔ
        ----------------------------------------------;
        int size;
        printf("Enter size of message:");
                        Ⓕ
        --------------------
        char buf[size];
        printf("Enter your message:");
        scanf("%s",buf);
        int x;
                    Ⓖ
        ---------------------
        if(x > 0)
        {
                sleep(4);
                        Ⓗ
                ------------------
                read(socket_fd, buf2, sizeof(buf2));
                printf("\nReceived: %s",buf2);
        }
        if(x == -1)
        {
                        Ⓘ
                ----------------------
                printf("%d",errno);
                exit(0);

        }
        if(x == 0){
                printf("nothing sent");

        }_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                        (J)
        exit(0);
        return(0);

}
```
Figure Q5(b): Incomplete code to echo a message