PROJECT 3

ICSI 500 – OPERATING SYSTEMS

Report by

Bhanu Pravallika Pothuru

UAlbany ID: 001555471

Net ID: BP885811

**Table of Contents**

[1. SYSTEM DOCUMENTATION 3](#_Toc127878176)

[1.1. HIGH LEVEL DATA FLOW DIAGRAM FOR THE SYSTEM 3](#_Toc127878177)

[1.2. LIST OF ROUTINES AND THEIR BRIEF DESCRIPTION 4](#_Toc127878178)

[1.3. IMPLEMENTATION DETAILS](#_Toc127878179) 5

[2. TEST DOCUMENTATION .](#_Toc127878180)5

[2.1. HOW YOU TESTED YOUR PROGRAM .](#_Toc127878181)5

[2.2. LIST OF YOUR TEST SETS INCLUDING THE RESULTS OBTAINED BY EACH SET 6](#_Toc127878182)

[3. USER DOCUMENTATION .](#_Toc127878183)7

[3.1. HOW TO RUN YOUR PROGRAM .](#_Toc127878184)7

**System Documentation**

1. High level data flow diagram for the system

The high-level data flow diagram for the system is as follows:

Diagram

Description automatically generated

1. List of routines and their brief description

Routines used in the program are listed below:

* **fork()**: The fork() system call creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.
* **pipe()**: The pipe() system call creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe.
* **execlp()**: execlp() function duplicate the actions of the shell in searching for an executable.
* **close()**: The close() system call closes a file descriptor, so that it no longer refers to any file and may be reused.
* **fopen()**: The fopen() function opens the filename pointed to, by filename using the given mode.
* **fclose()**:The fclose() function closes an open file pointer.
* **perror()**: The perror() function produces a message on standard error describing the last error encountered during a call to a system or library function.
* **strlen()**: The strlen() function calculates the length of the string, excluding the terminating null character.
* **strcasecmp()**: The strcasecmp() function compares string1 and string2 without sensitivity to case. All alphabetic characters in string1 and string2 are converted to lowercase before comparison.
* **wait()**: The wait() system call blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent **continues**its execution after wait system call instruction.
* **dup2()**: The **dup2()** system function is used to create a copy of an existing file descriptor.
* **encode()**: This function is used to execute all the tasks involved in encoding data.
* **decode()**: This function is used to execute all the tasks involved in decoding data.
* **removeParity()**: This function is used to check and remove parity bit from binary data.
* **addParity()**: This function is used to add parity bit to the 7-bit binary data.
* **binaryToCharacter()**: This function is used to convert 7-bit binary data to character.
* **characterToBinary()**: This function is used to convert a character into binary.
* **frameData()**: This function is used to frame the data i.e, adds SYN and length characters to the data.
* **deframeData()**: This function is used to deframe the data i.e, removes SYN and length characters from the data.
* **physicalLayer()**: This function is used to either encode or decode the data.
* **dataLinkLayer()**: This function is used to either frame or deframe the data.
* **applicationLayer()**: This function is used to either read data from input files or write data to the output files.
* **charA()**: This function replaces all a’s in data with A’s.
* **charE()**: This function replaces all e’s in data with E’s.
* **charI()**: This function replaces all i’s in data with I’s.
* **charO()**: This function replaces all o’s in data with O’s.
* **charU()**: This function replaces all u’s in data with U’s.
* **digit()**: This function computes the sum of all the digits in the data
* **writer():** This function is used to share converted data with the serverEncoder process
* **pthread\_create()**: This function is used to created threads.
* **pthread\_join**(): This function waits for the thread specified by thread to terminate.
* **sem\_destroy**(): This fuction destroys the unnamed semaphore at the address pointed to by sem.
* **sem\_wait**(): This function decrements (locks) the semaphore pointed to by sem.
* **sem\_init**(): This function initializes the unnamed semaphore at the address pointed to by sem.
* **crc32CheckSum():** This function calculates the crc 32 checksum value

1. Implementation details

The program is developed using C and is implemented as follows:

* On running the program, pair of sockets (server and client).
* Client program encodes the data from the input file intext.txt and shares the encoded data with the Server using socket.
* The encoded data is then converted according to the requirements by the 7 threads created by the Server.
* Server then decodes the data converted and passes it with client through socket.
* The client encodes the data again and stores the output in the file result.txt.

Test documentation

1. How you tested your program

To test the program compile and build the source code i.e, server.c, client.c and run the command executable file i.e, ./server and ./client on two different terminals. For server program port number is given as a command line input. For client program hostname and port number are given as command line inputs. On successful execution of the program result.txt will be generated. Tester can verify the accuracy of the output by comparing the contents of result.txt with to the uppercase version of vowels in the input, appended by the sum of the all digits. If error is generated and detected by CRC program terminates, without producing result.txt.

intext.txt is considered as input file for testing the program.

USER DOCUMENTATION

1. How to run your program

* Compiler: GNU C
* System: Linux OS – Ubuntu

Follow the below instructions to run the program:

Step 1: Open terminal.

Step 2: Compile and build supporting c files.

Command: gcc -o physicalLayer physicalLayer.c

Command: gcc -o dataLinkLayer dataLinkLayer.c

Command: gcc -o applicationLayer applicationLayer.c

Command: gcc -o encodeDecode encodeDecode.c

Step 3: Compile and build source code.

Command 1: gcc -o server server.c

Command 2: gcc -o client client.c

Step 4: Run the executable/object file of server and client programs.

Command 1: ./server 54554

Command 2: ./client 127.0.0.1 54554

Output: result.txt file will be generated