2023

Project 1

Jeremy Varghese

jv852856

Contents

[System Documentation 2](#_Toc146925871)

[1. Data Flow Diagram 2](#_Toc146925872)

[2. Routines 3](#_Toc146925873)

[3. Implementation Details 4](#_Toc146925874)

[Test Documentation 6](#_Toc146925875)

[1. Method of Testing 6](#_Toc146925876)

[2. Test Sets 6](#_Toc146925877)

[User Documentation 8](#_Toc146925878)

[How to run 8](#_Toc146925879)

[Parameters 8](#_Toc146925880)

# System Documentation

## Data Flow Diagram

A diagram of a flowchart

Description automatically generated

## Routines

Application Layer:-

* 1. producer():- Parent process that reads input file and sends data in chunks of 64 characters to be encoded. Producer also reads capitalized encoded frames from Pipe In, and sends those frames to be deframe for further processing. Producer also forks and creates consumer child.
  2. consumer(char\* fdOut, char\* fdIn) - Child of producer. Reads encoded input data, and sends it to deframe for further processing. Takes in file descriptors for the outPipe to read from, and the inPipe for appropriate service to write to.

Physical Layer:-

* 1. int encode(char \*inData, char \*len, char\* fdOut\_One) – encodes incoming data frame of <=64 characters to 7 bits of binary for each character. Number of characters is encoded and added to frame.
  2. void addParity(char \*inData, char\* fdOut\_One) – adds parity to the encoded frame with number of characters and the characters.
  3. int checkRemoveParity(char\*\* characters) – checks parity bits for all characters and removes them
  4. int decode(char\*\* blocks) – converts encoded binary characters to ascii values. Writes to data.done if called by producer, or calls toUpper if called by consumer.

Data Link Layer:-

* 1. void buildFrame(char \*inData, char\* fdOut\_One) – adds SYN characters (hardcoded) to front of frame and writes to file descriptor of pipe passed as argument. (Writes to both inPipe and outPipe)
  2. int deframe(char \*inData, char\* fdIn\_One) – checks and removes SYN characters, and sends deframed data

Additional:-

* 1. int toUpper(char\* str, char\* len, char\* fdIn\_One) – converts lower case characters to upper case, and sends frame to encoder. Writes frames to data.outf
  2. int error(char \*inData, char\* len, char\* fdOut\_One) – adds error to the 3rd frame. Flips one bit using srand(pid) to determine index of bit.

## Implementation Details

I will describe the data flow.

The producer creates 2 pipes, an outPipe (sending information from the producer to the consumer), and an inPipe (c->p)

The producer forks and the child closes the appropriate pipe fd’s and execs consumer, passing it the 2 open file descriptors.

The producer closes its appropriate pipe fd’s, opens the data.inpf file, steps through every character, and saves them to a buffer. It counts the number of characters read, and every 64 characters, it forks and sends the buffer, the length of the frame, the file descriptor of the outPipe, and “0” as isCap to indicate which data flow is currently running to encoderService in the child process by calling exec.

If there are <64 characters, the producer sends the remaining characters, the length, outPipe fd, and “0” to encoderService via an exec.

The producer also keeps track of how many frames have been sent. The third frame is sent to the errorService instead of the encoderService, to generate a random bit flip, and continue with the data flow.

The processor then reads from the inPipe until the consumer function call closes at the other end. It receives an encoded bit string from the consumer function call and sends the string to the deframer for further processing.

Once all frames have been sent, and all child processes have returned, the producer ends, terminating the program.

The encoder converts the length of the string into its 7-character binary representation, and all the characters passed in its argument to their ASCII value’s 7-character binary representations. It forks and sends this encoded information, along with the pipe fd to write to and “0”, to the parityAddService via an exec.

The errorService does the same thing as the encoder, except it uses the process id as the seed for srand, and generates a random number, mods it with the length of the string passed, to generate a random index whose bit is to be flipped. It flips this bit and then sends the resulting string to parityAddService via an exec.

The parityAddService calculates the odd parity for each of the 7 character blocks and converts each of them into 8 character blocks. It forks and sends this encoded data, the pipe fd and “0”, to the buildFrameService via an exec.

The buildFrameService creates one long string, with parity bit binary encoded SYN characters (2222) in the front, followed by the data sent as an argument. It then writes to the pipe fd from the argument. Please note, the buildFrameService is used by both the producer and consumer, it does not care who calls it, it writes to the file descriptor that has been passed as an argument. In this instance, since the producer flow has called the buildFrameService, it will write to the outPipe. Depending on if “0” or “1” is passed as isCap, it writes to either data.binf (“0” = not capitalized, i.e. producer is ultimate parent), or data.chck (“1” = capitalized, i.e. consumer is ultimate parent).

The consumer is constantly reading from the outPipe until the producer closes the write end after all frames have been processed. When the consumer receives a fully encoded frame, it forks, and the child execs deframe with the received data, and the file descriptor to write to. Once the outPipe has been closed, and all the frames sent for further processing, the consumer closes its pipe file descriptors.

The deframe service receives a fully encoded frame. It checks the first 2 8 character blocks, and compares them to the SYN encoded string. If they match, then there is no issue in transmission and the service removes those 16 characters, and sends the rest of the frame and the file descriptor to write to, to the checkParityRemoveService via an exec. Please note, this service is called by both the Producer, and the Consumer. The producer passes the frame read from the inPipe, and -1 as the file descriptor argument to indicate writing to data.done. The consumer passes the frame read from the outPipe, and the inPipe file descriptor to write to. These fd values are passed down to subsequent service calls.

The checkParityRemoveService checks each 8 character block for its odd parity, and compares it to the Most Significant Bit. If they match, the data isn’t corrupted. This service grabs the length of the string passed in the encoded message, and loops through the string and extracts 8 character blocks. It checks those blocks for their parity and removes the bit. All blocks are now 7 characters long. This service then forks and exec calls the decoder service, passing the raw encoded data and the file descriptor to write to.

The decoder service loops through the 7 character blocks and converts them into their ASCII values. These values are stored in a char pointer. Once all the information has been decoded, the service checks the fd. If the fd value is -1, it writes the decoded frame to the data.done file, frees the memory and returns. If the fd is anything else, it forks and exec calls the toUpperService and passes the decoded string and the fd to write to.

The toUpperService simply converts all characters >=’a’ and <=’z’ to its upper case value. It writes this transformed string to the data.outf file. It then forks and exec calls encoderService with this changed string, its length, the inPipe fd, and “1” as isCap.

The encoder service then encodes the upper case string, passes it on to the parityAddService, with the inPipe fd, and “1” as isCap.

The parityAddService adds the parity bits and passes it on to the buildFrameService, with the inPipe fd, and “1” as isCap.

The buildFrameService builds the full frame with the encoded SYN chars, and writes the frame to the inPipe. It also writes this encoded frame to the data.chck file since isCap is “1”.

The producer reads this frame from the inPipe, and sends it to the deframe service, passing -1 as the pipe file descriptor.

The deframe service breaks the frame down into 8 character blocks and passes them on to the checkRemoveParityService, with -1 as the fd argument.

The checkRemoveParityService checks and removes the parity bits, and passes on the 7 character blocks to the decoder, with -1 as the fd argument.

The decoder converts the binary encoded chunks into their ASCII values, and on seeing -1 as the fd argument, writes the decoded string to data.done, and returns.

Once all frames have gone through this sequence, the input file has successfully been converted as desired, and all processes end.

The data.done file will have an error somewhere in the 3rd frame. On running ./producer, the console will display the index chosen, and the character being flipped. You can run diff data.done data.compare to check for the error.

# Test Documentation

## Method of Testing

Initial testing began with <64 characters. BCCA was used to get basic process functionalities in place.

bcca was then used to test data flow with toUpper service implemented.

When bcca was correctly processed, the alphabet characters were used to reach >64 characters, and functionalities were tested and tweaked to correctly process the data.

When frames were correctly implemented, the quote given in the Project 1 document was tested. Process flow had bugs that were rectified.

Error in input file was detected, and professor was notified.

Error service was implemented and Data.compare was created to diff with Data.done

Diff correctly identifies area of difference.

The first scene of Jurassic Park’s script was also tested and correctly converted with an error included in the 3rd frame.

## Test Sets

* 1. Data.inpf – input data. Contains the quote in the problem statement. Can be swapped for any text.
  2. Data.binf – binary, parity encoded data frames, ready for sending through outPipe
  3. Data.outf – capitalized version of .inpf before encoding and sending through inPipe
  4. Data.chck – binary, parity encoded data frames, ready for sending through inPipe
  5. Data.done – fully capitalized output version of input file. Contains error in frame 3
  6. Data.compare – Correct fully capitalized output version of input file.
  7. jurassicPark.inpf – input file containing the first scene from Jurassic Park’s script.
  8. jurassicPark.compare – Correct fully capitalized output version of Jurassic Park’s script.

# User Documentation

## How to run

Simply navigate to folder with the source code, input file, data.compare and compiled services.

Type ./producer in console. The Test Set files, along with the data.done files should be created.

Run diff data.done data.compare to see differences in the two files.

Copy and paste the contents of jurassicPark.inpf into Data.inpf and run ./producer.

Run diff data.done jurassicPark.compare to check the error.

## Parameters

No parameters to be used.