Multi-Threading

Submitted to Prof. Jackson Marques de Carvalho
for Project 2 in CSI 500 Operating Systems

Collaborators:

Jinyu Tian
AlbanyID: [redacted]

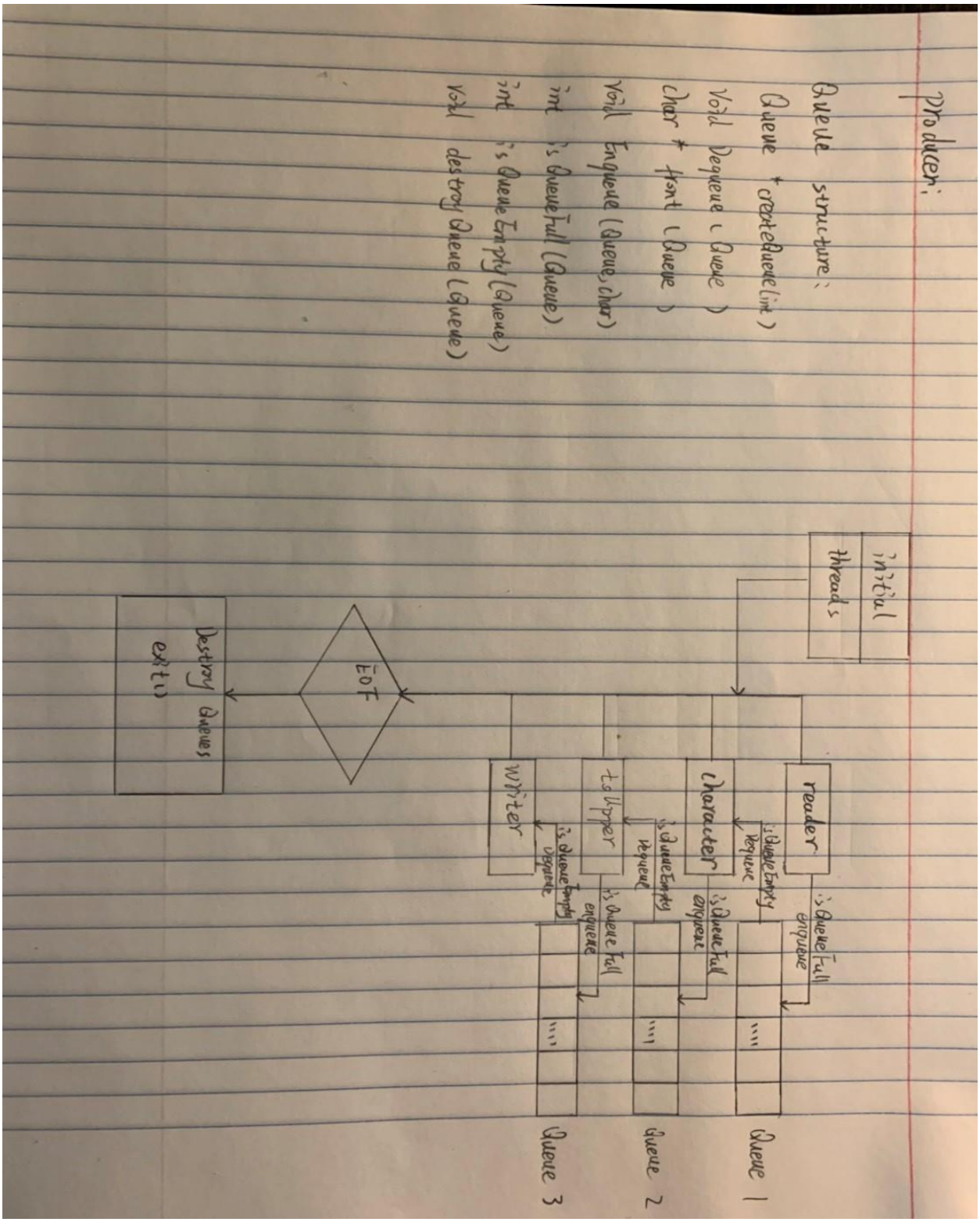
Santhosh Ranganathan
AlbanyID: [redacted]

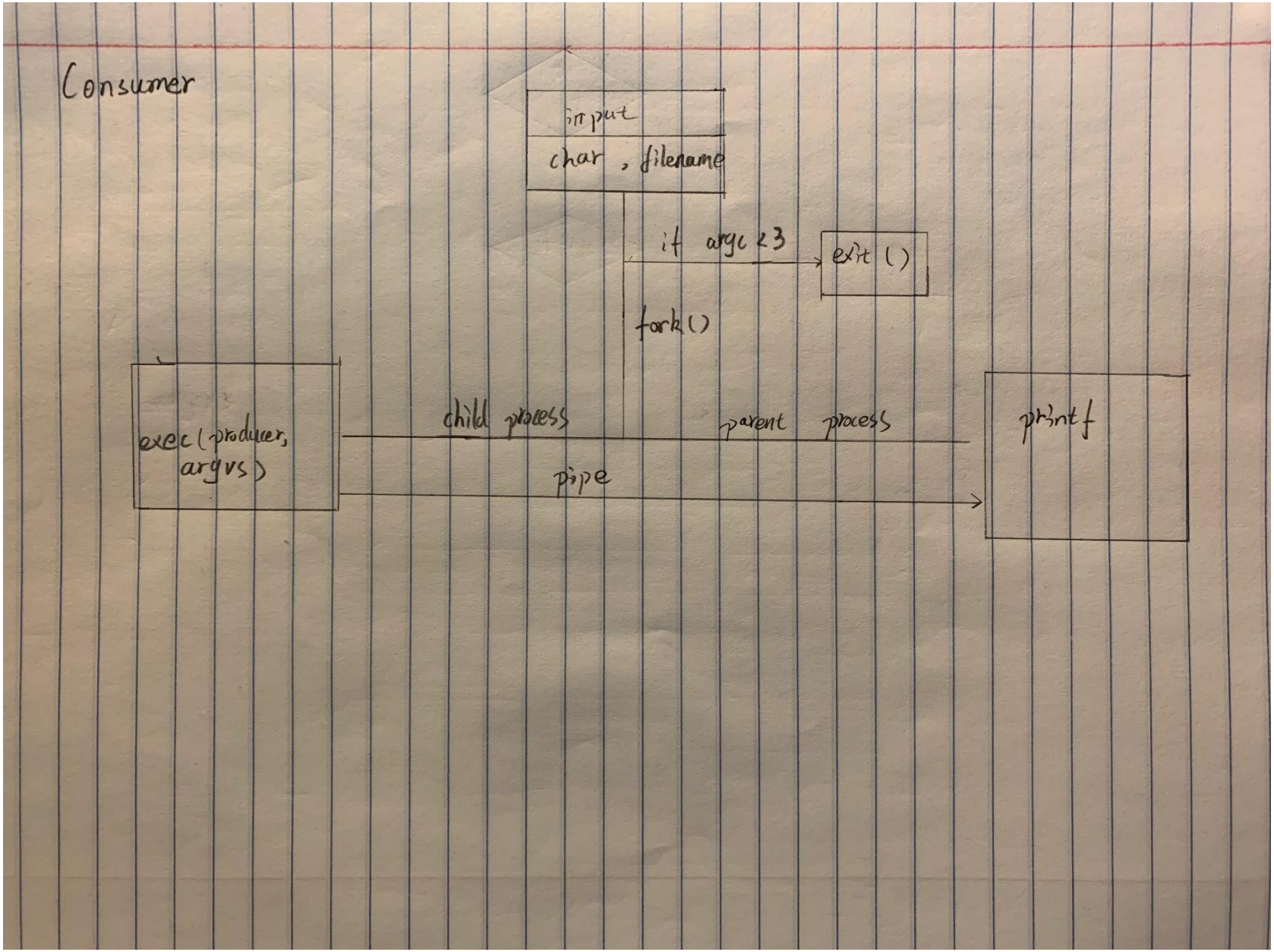
Table of Contents

System Documentation 3
Flow Diagram 3
List of Routines 5
Implementation Details 6
Test Documentation 7
How Program was tested? 7
Outputs 7
User Documentation 7
Source Location 7
How to run Program? 7

System Documentation

Flow Diagram




Consumer
input
char , filename
if argc < 3
exit ()
fork()
exec (producer, argvs)
child process
parent process
printf
pipe

List of Routines

Consumer (cons.c)

Name	Description
`main`	Entry point function which sends user's input file and character to be replaced as arguments to the producer after forking and later displays the output file.

Producer (proc.c)

Name	Description
`reader`	Reads each line from the input file and pushes to the first queue.
`character`	Reads content from the first queue and replaces 'space' character with the character given and pushes the result to the second queue.
`toUpper`	Read content from the second queue, changes all characters to uppercase and pushes the result to the third queue.
`writer`	Read content from the third queue, write the content to the output file.
`main`	Entry point function with an option to demonstrate multi-threading.

Implementation Details

Consumer:

Read user's input and fork process. The child consumer process executes producer and passes arguments to it. The producers prints the output filename after processing input and the consumer parent process gets the result of producer via pipe and displays the file.

Producer:

Reader: read each line from the file and enqueue the content to the first queue if the queue not full.

Character: read content from the fist queue if first queue is not empty, and dequeue the content in the first queue. Replace space with specific character. And then enqueue it into second queue if second queue is not full.

toUpper: read content from second queue if the second queue is not empty, and dequeue the content in the second queue. Uppercase all characters in content, and then enqueue the content into third queue if third queue is not full.

Writer: read content from third queue is the third queue is not empty, and dequeue the content in the third queue. And then write the content to output file.

Data Structure Used

Queue data structure used to buffer i/o between threads

The project was completed in parts

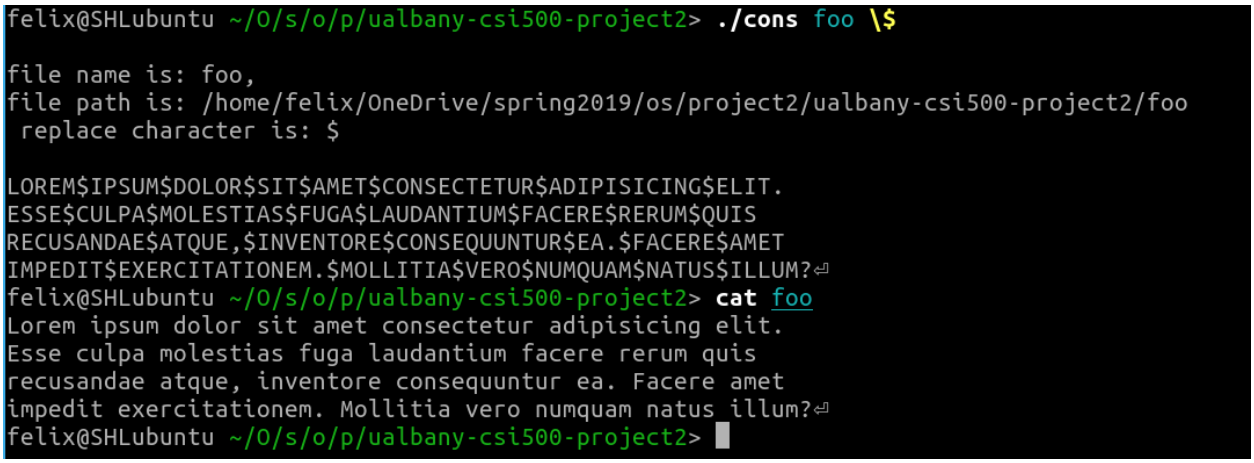
- 4/14/2019 Jinyu created the code for queue and the main parts for consumer and producer
- 4/20/2019 Jinyu and Santhosh worked together to get the first working version which executed sequentially
- 4/22/2019 Santhosh added semaphores to producer
- 4/25/2019 Jinyu added documentation
- 5/4/2019 Santhosh debugged code
- 5/8/2019 Jinyu and Santhosh added comments before final submission

Test Documentation

How Program was tested?

The program was tested on different dummy inputs of varying sizes (4, 1000, 100k and 1 million lines) to check for correctness of output and speed. A demo option was added to producer and results of differing buffer sizes can be seen in the file 'buffersizeeffect.txt'.

Output



User Documentation

Source Location

Submitted along with this documentation. It is present in the folder "ualbany-csi500-project2"

How to run Program?

Simple Shell doesn't use any arguments. To execute, go to "ualbany-csi500-project2" and type

```
./cons <input-filename> <character-to-be-replaced>
```

to run the application.

The producer has additional in-built arguments to demonstrate multi-threading

```
./proc <input-filename> <character-to-be-replaced> --demo --buffersize
<desired-buffer-size>
```