Lab #5: Network Lab

Prof. Jae W. Lee (jaewlee@snu.ac.kr)
Department of Computer Science and Engineering
Seoul National University

TA (snu-arc-sysprog-ta@googlegroups.com)

Before get started

- **Make sure your submission is correctly formatted**

  -
    ```
    {student_id}.tar
    ├── {student_id}.pdf
    └── src
        └── mcdonalds.c
        └── client.c
    ```

- **Please double check your submissions!**

Contents

Important Dates

- 04 Jun. - Lab Hand-out Session (Today!)
- 13 Jun (Thur) at 11:00. - Live Q&A Session
- 17 Jun. 23:59 - Submission Deadline

- Questions about the lab will be conducted through github issue. Feel free to post questions (except your code!)
- Live Q&A Session is totally optional

Before the Presentation:

- **All content on this slide is sourced from README file.**
- **For detailed information, please refer to README.**
  - https://github.com/SNU-ARC/2024_spring_sysprog_Lab5/blob/main/README.md

Goal of this Lab(1/2)

- **We implement a virtual McDonalds Server & Client**

Goal of this Lab(2/2)

- **You will learn**
  - how to communicate under a TCP/IP network environment
  - how to assure atomicity on critical sections between threads
  - how to limit the number of the clients on the listening socket

Environment setup(1/3)

- You can get the skeleton code from the git repo

```
git clone https://github.com/SNU-ARC/2024_spring_sysprog_Lab5.git
```

Environment setup(2/3)<Optional>

- **If you want to keep your own repository, you should keep the lab's visibility to private. Otherwise, others would see your work.**

- **Changing visibility**
  - After cloning the repository, you should change the push remote URL to your own repository.

1. **Create an empty repository that you're going to manage (again, keep it private)**
2. **Copy the url of that repository**
3. **On your terminal in the cloned directory, type**
   ```
   git remote set-url --push origin <repo_url>
   ```
4. **Check with git remote -v if the push URL has changed to yours while the fetch URL remains the same (this repo)**

Environment setup(3/3)

- **The handout contains the following files and directories.**

dir	file	description
src	burger.c/h	Macro definitions for socket connection and enum types for burgers.
	net.c/h	Network helper functions for the lab.
	parser.c/h	Implementation of command line parser. Do not modify!
	mcdonalds.c	The McDonald's server. A skeleton is provided. Implement your solution by editing this file.
	client.c	Client-side implementation. A skeleton is provided. Implement your solution by editing this file.
reference	mcdonalds	Reference implementation of server
	client	Reference implementation of client
.	Makefile	Makefile for compiling mcdonalds and client
	README.md	

Overview of McDonalds(1/4)

- **Server & Client communication via socket interface**
  - Server : Serves client and generates burger
  - Client : Requests burgers

- **Threaded execution**
  - Server : A serving thread for each client thread + Kitchen threads
  - Client : Multiple threads created

Overview of McDonalds(2/4)

- **Each client thread sends a single request to the server**
  - Request = Sequence of Orders (e.g. “bigmac bigmac chicken”)
    - The number of orders and the types are selected randomly
  - Order = A type of burger (e.g. “bigmac”)
  - Available types = {bigmac, cheese, chicken, bulgogi}

- **A serving thread is spawned in the server for each client thread**
  - After receiving a request from client, split the request into orders
  - Orders are enqueued into the order queue

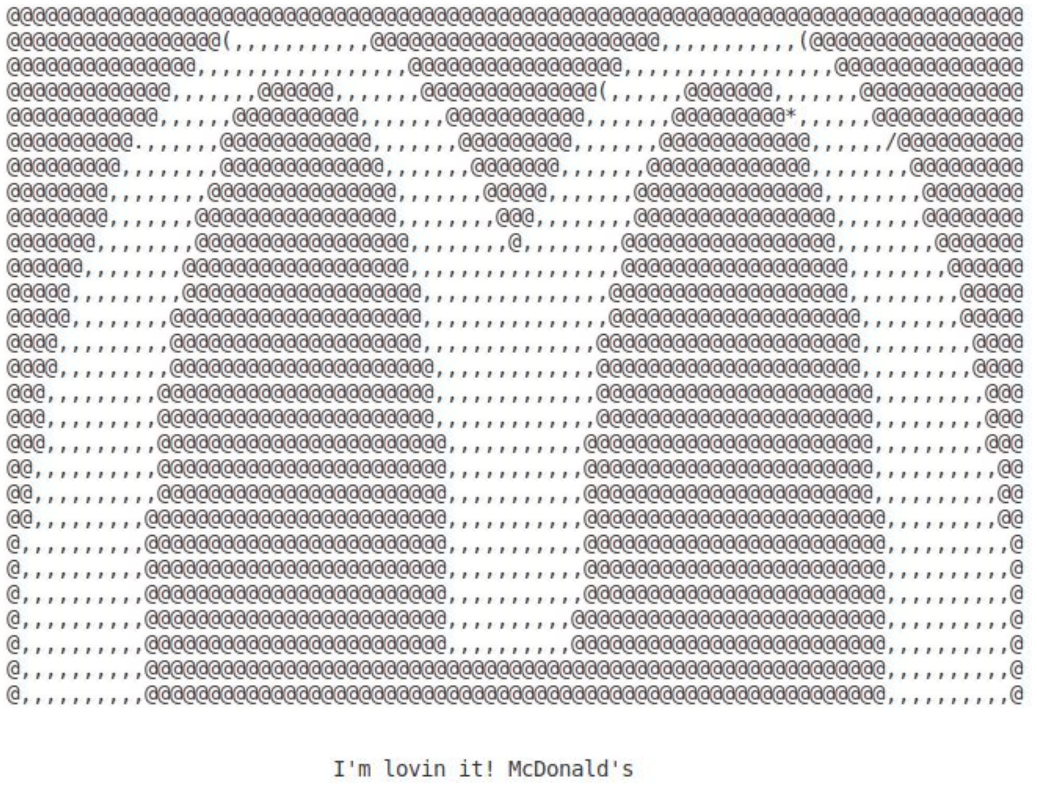
Overview of McDonalds(3/4)

- **Kitchen threads dequeue from the order queue, and “cook”**
  - A single order is dequeued at a time
  - “Cook” means to append the burger name to the order string
    - The order string is shared within a same request
    - The order string is initialized as a empty string
    - The sequence of the burgers in the order string may differ from the original request sequence, but it must contain all of the orders
  - If the kitchen thread “cooked” the last burger, signal the serving thread

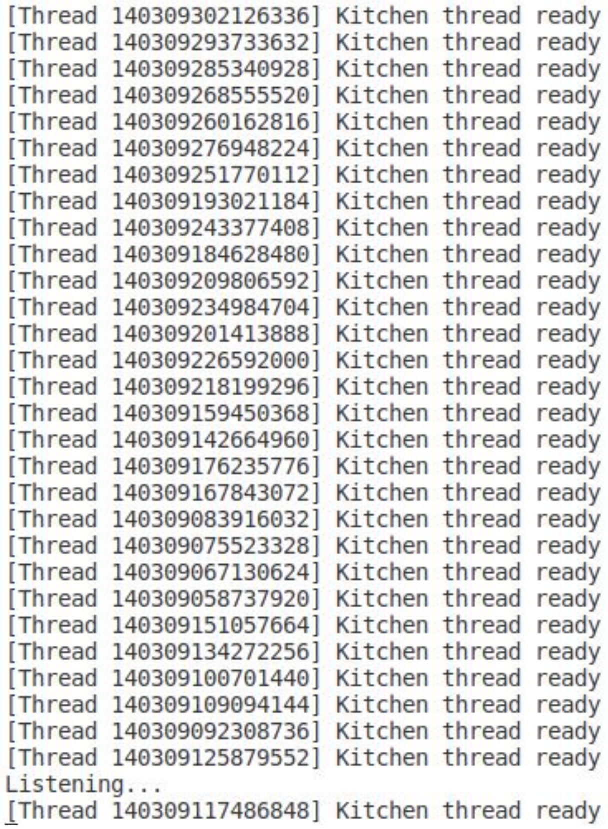
Overview of McDonalds(4/4)

- **Serving thread sends the “cooked” order string back to the client**
  - Bon appetit!

Example Execution - Start Server



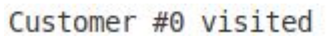
```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@(,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,(@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@,,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@
@@@@@@@@@@@@@,,,,,,,@@@@@@,,,,,,,@@@@@@@@@@@@@@(,,,,,@@@@@@@,,,,,,,@@@@@@@@@@@@@
@@@@@@@@@@@@,,,,,,@@@@@@@@@@,,,,,,,@@@@@@@@@@@,,,,,,@@@@@@@@@*,,,,,,@@@@@@@@@@@@
@@@@@@@@@@.,,,,,,@@@@@@@@@@@@,,,,,,,@@@@@@@@@,,,,,,@@@@@@@@@@@@,,,,,,/@@@@@@@@@@
@@@@@@@@@,,,,,,,,@@@@@@@@@@@@@,,,,,,,@@@@@@@,,,,,,@@@@@@@@@@@@@,,,,,,,,@@@@@@@@@
@@@@@@@@,,,,,,,,@@@@@@@@@@@@@@@,,,,,,,@@@@@,,,,,,@@@@@@@@@@@@@@@,,,,,,,,@@@@@@@@
@@@@@@@@,,,,,,,@@@@@@@@@@@@@@@@,,,,,,,,@@@,,,,,,,@@@@@@@@@@@@@@@@,,,,,,,@@@@@@@@
@@@@@@@,,,,,,,,@@@@@@@@@@@@@@@@@,,,,,,,@,,,,,,,,@@@@@@@@@@@@@@@@@,,,,,,,,@@@@@@@
@@@@@@,,,,,,,,@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@,,,,,,,,@@@@@@
@@@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@@@
@@@@@,,,,,,,,@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@,,,,,,,,@@@@@
@@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@@
@@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@@
@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@
@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@
@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@
@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@
@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@
@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@
@,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,,,,,,,,,,@


                         I'm lovin it! McDonald's
```



```
[Thread 140309302126336] Kitchen thread ready
[Thread 140309293733632] Kitchen thread ready
[Thread 140309285340928] Kitchen thread ready
[Thread 140309268555520] Kitchen thread ready
[Thread 140309260162816] Kitchen thread ready
[Thread 140309276948224] Kitchen thread ready
[Thread 140309251770112] Kitchen thread ready
[Thread 140309193021184] Kitchen thread ready
[Thread 140309243377408] Kitchen thread ready
[Thread 140309184628480] Kitchen thread ready
[Thread 140309209806592] Kitchen thread ready
[Thread 140309234984704] Kitchen thread ready
[Thread 140309201413888] Kitchen thread ready
[Thread 140309226592000] Kitchen thread ready
[Thread 140309218199296] Kitchen thread ready
[Thread 140309159450368] Kitchen thread ready
[Thread 140309142664960] Kitchen thread ready
[Thread 140309176235776] Kitchen thread ready
[Thread 140309167843072] Kitchen thread ready
[Thread 140309083916032] Kitchen thread ready
[Thread 140309075523328] Kitchen thread ready
[Thread 140309067130624] Kitchen thread ready
[Thread 140309058737920] Kitchen thread ready
[Thread 140309151057664] Kitchen thread ready
[Thread 140309134272256] Kitchen thread ready
[Thread 140309100701440] Kitchen thread ready
[Thread 140309109094144] Kitchen thread ready
[Thread 140309092308736] Kitchen thread ready
[Thread 140309125879552] Kitchen thread ready
Listening...
[Thread 140309117486848] Kitchen thread ready
```

Example Execution - Start Client

Server:



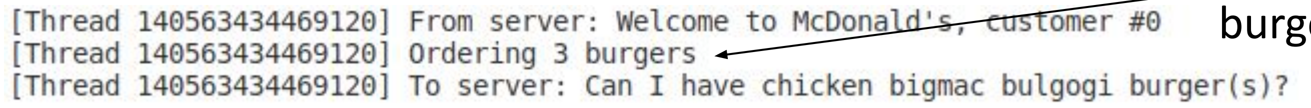
```
Customer #0 visited
```

Assign customer ID when client visits

Message from server

Client:

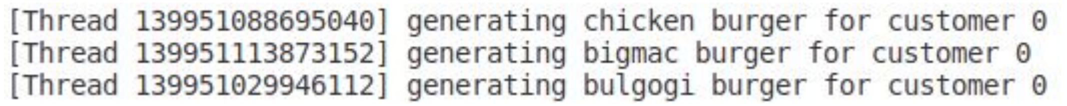
Randomly choose number of burgers



```
[Thread 140563434469120] From server: Welcome to McDonald's, customer #0
[Thread 140563434469120] Ordering 3 burgers
[Thread 140563434469120] To server: Can I have chicken bigmac bulgogi burger(s)?
```

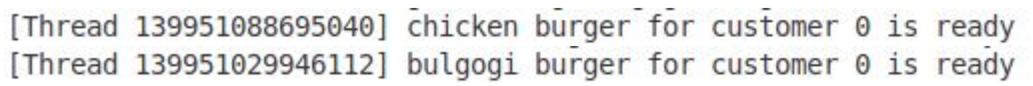
Send request

Example Execution - “Cook” burger

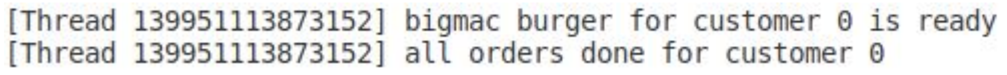


```
[Thread 139951088695040] generating chicken burger for customer 0
[Thread 139951113873152] generating bigmac burger for customer 0
[Thread 139951029946112] generating bulgogi burger for customer 0
```

Each kitchen thread dequeues and “cooks” a burger



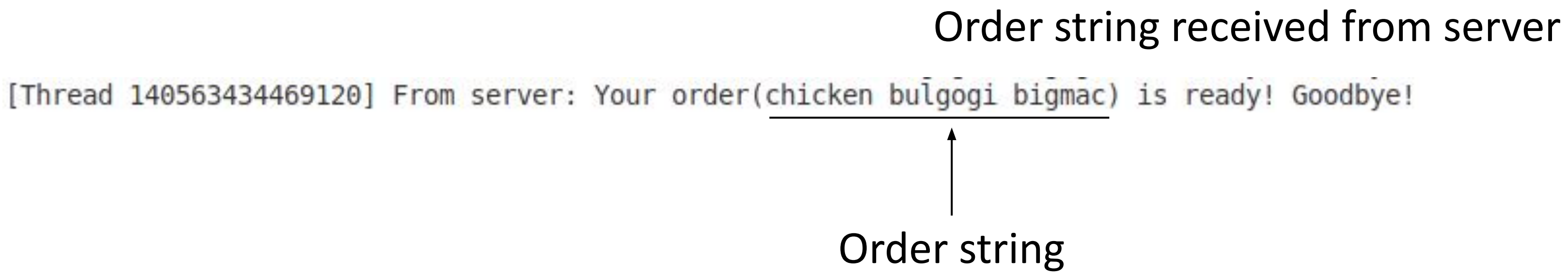
```
[Thread 139951088695040] chicken burger for customer 0 is ready
[Thread 139951029946112] bulgogi burger for customer 0 is ready
```



```
[Thread 139951113873152] bigmac burger for customer 0 is ready
[Thread 139951113873152] all orders done for customer 0
```

The kitchen thread to make the final burger signals the serving thread

Example Execution - Receive request

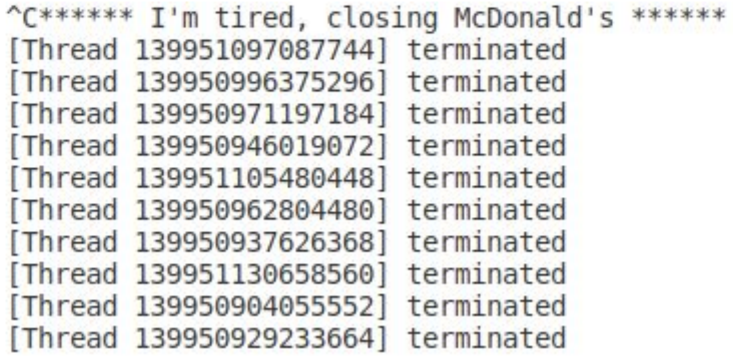


Order string received from server

```
[Thread 140563434469120] From server: Your order(chicken bulgogi bigmac) is ready! Goodbye!
```

Example Execution - Ctrl+C

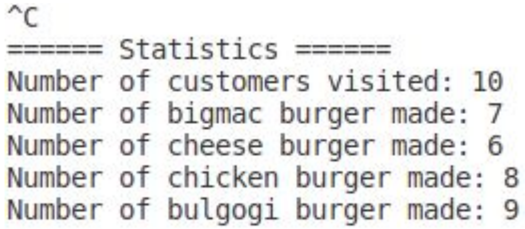
First Ctrl+C



```
^C****** I'm tired, closing McDonald's ******
[Thread 139951097087744] terminated
[Thread 139950996375296] terminated
[Thread 139950971197184] terminated
[Thread 139950946019072] terminated
[Thread 139951105480448] terminated
[Thread 139950962804480] terminated
[Thread 139950937626368] terminated
[Thread 139951130658560] terminated
[Thread 139950904055552] terminated
[Thread 139950929233664] terminated
```

⋮

Second Ctrl+C



```
^C
====== Statistics ======
Number of customers visited: 10
Number of bigmac burger made: 7
Number of cheese burger made: 6
Number of chicken burger made: 8
Number of bulgogi burger made: 9
```

Parallelism Optimization (Step-2)

- **Reference(and also your code after following step-1) uses a global mutex, which every kitchen thread shares**
- **Devise a strategy to improve parallelism**
- **Compare the performance difference**
- **Use `time ./client <n>` to check the performance**

Code & Test(1/2)

- **Reference Solution is provided**
  - `reference/client`
  - `reference/mcdonalds`

- **Reference mcdonalds is compiled with:**
  - `CUSTOMER_MAX = 10`
  - `NUM_KITCHEN = 30`

- **Reference client is compiled with:**
  - `MAX_BURGERS = 3`
  - `BURGER_NUM_RAND = 0(False)`

Code & Test(2/2)

- **Test with various settings!**
  - You may change constant definitions in `src/burger.h`
  - Try various number of threads
  - Try various max burger settings
- **But your code must be able to run with arbitrary constants!**

Grading Policy

- **Test Bench: 75 %**
- **Report** : **25 % (should include the following contents)**
  - Description of your implementation
    - how to communicate under a TCP/IP network environment
    - when and how to assure atomicity between threads
  - Description of your parallelism optimization strategy and the performance analysis of your implementation against the reference implementation.
- **For late submission:**
  - ~~A deduction of 20%p per 24 hours~~
    -> No late penalty until 20 Jun. 23:59 (professor's grace!)
    -> 20%p deduction until 21 Jun. 23:59, no submission allowed afterwards

Submission(via eTL)

- **Write-up**
  - Briefly describe your implementation.
  - Filename: [student_id].pdf (example: 2024-12345.pdf)
  - **Please** submit it in **pdf** format. Other formats are not accepted.
- **Compress your source code and write-up into a single file**
  - Compress **client.c, mcdonalds.c** and your report with following command
  - `$ tar -cvf 2024-12345.tar src/client.c src/mcdonalds.c 2024-12345.pdf`
  - Filename should be [student_id].tar (example: 2024-12345.tar).
  - **Please** submit it in **tar** format. Other formats are not accepted.
  - Refer README.md for submission instructions.
- **Submission deadline: by 23:59 on June 17, 2024**

Questions?