전남대학교 '운영체제' - 박태준 교수

Assignment #1. Process and thread (그리고 스케쥴링 한스푼)

아래 문제들을 해결 한 후 풀이(해설)과 함께 이클래스를 통해 제출하세요.

- 즉, 소스파일 + 레포트가 제출되시면 됩니다.

- 레포트 양식은 자유. 단, 너무 불필요하게 길지는 않게!

- 파일형식은 반드시 pdf로 ...

- 파일형식은 반드시 pdf로 ...

- 파일형식은 반드시 pdf로 ...

양식 안지켜지면 과제 제출 자체를 무효로 취급함. 🙏🙏🙏🙏🙏

설마 이렇게 까지 썼는데 설마 또 hwp일까... 라고 했었는데 그런데 그것이 실제로 일어났었습니다. 😡😡

- 코딩은 큰 주제와 맥락을 벗어나지 않는 선에서, 자유롭게. 창의롭게!

*주의: 본 과제를 위한 코드들은 **xNIX 계열 (Linux 또는 MacOS)에서** 실행가능한 코드들 입니다. 윈도우 사용자 분들은

1. WSL (Windows subsystem for Linux)
2. VMware 또는 VirtualBox등의 가상화 환경

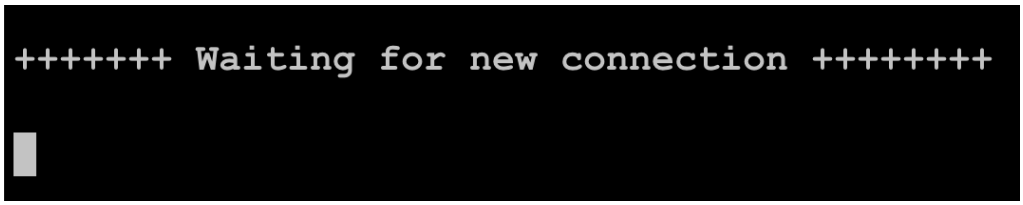
등에서 본 과제를 하시면 되겠습니다. 혹시 과제 실행환경이 제한되시는 분들은 연락주세요 :)

기한: 5월 17일 금요일 까지!

#1. My first web-server

- 웹 서버(Web server)는 웹(web)을 통해 수신되는 요청(Request)들을 처리해주는 애플리케이션 입니다. 백엔드 그 자체라 봐도 좋겠네요 :) 그리고 웹서버를 한번 직접 만들어 보는 것은 웹의 동작 원리와 백엔드에 대해 잘 이해할 수 있는 아주 좋은 경험이 됩니다! 만들면서 OS에서 배운 내용들을 활용 해보는 것은 덤이구요!

본 과제의 "server.c"는 아주 간단한 웹서버에 대한 예제 코드입니다. 컴파일 이후 → 프로그램을 실행시켜보면, 다음과 같은 화면을 보실 수 있습니다 (e.g., gcc server.c; ./a.out)
(만약 "In bind: Address already in use" 라는 에러메시지가 출력되면 포트번호를 8090이 아닌 다른 것으로 변경해보세요.)



그리고 프로그램이 실행중인 상태에서 웹브라우져를 통해 "http://localhost:8090/"로 접속해보시면 아래와 같이 간단한 메시지가 출력이 되는 것을 확인할 수 있습니다.

웹서버가 갖춰야할 중요한 특성 중 하나는 많은 클라이언트들의 요청들을 짧은 시간안에 잘 처리 하는 것 입니다. 하지만, 우리의 웹서버는 아주 단순한 웹서버로서, 클라이언트의 요청을 하나에 하나씩만 처리할 수 있는 구조라서 많은 수의 요청을 처리하기에 부적절합니다.

이를 극대화 하여 보기위해서 **60번째 라인의 sleep의 주석을 해제하여 5초의 처리 지연을 추가** 해봅시다. 그리고 여러 탭을 통해 동시에 여럿 접속을 하게되면, 나중에 접속이 시도된 탭은 앞선 요청들이 다 처리가 될때까지 접속이 지연되거나, 최종적으로 접속에 실패하는 것을 볼 수 있으

실겁니다. 또는, 이를 조금 더 자세히 살펴보기 위해서 accesses.py 파이썬 코드를 실행시켜보실 수도 있습니다. 이 코드는 파이썬 쓰레드를 활용해 여러개의 동시접속을 만들어냅니다.

약 100여개의 접속을 시도하는 와중에... 많은 수의 접속들이 실패를 하는군요! 모든 처리가 끝날 때 까지 시간도 오래걸리구요...!

이제, 이러한 문제를 해결하기 위해, server.c 코드를 fork를 활용하여 동시에 여러 요청을 능숙하게 처리할 수 있게 만들어 봅시다!

- **Step 1)** server.c를 컴파일 하고 실행해 보세요.
- **Step 2)** 웹브라우져를 통해 http://localhost:8090/로 접속을 해보세요. 그리고, server.c의 sleep 주석을 uncomment하시고 여러개의 동시접속을 만들어보세요. accesses.py를 이용하시면 더욱 쉬울겁니다!
- **Step 3)** fork를 server.c에 추가하여 sleep 5가 있더라도 여러 요청을 잘 처리할 수 있게 만들어 봅시다!
- **기대 결과물)** 내용이 반영된 server_01.c, 이에 대한 설명 또는 레포트

#2. My second web server

- 1번의 My first web server는 Multi-process기반으로 다수의 요청을 처리하는 형태입니다. 고작 짧은 메시지 하나 처리하는데, 프로세스라니...너무 무겁습니다. 이런 방식이라면 실제 서비스라면 아마 하나의 서버 장치에서 많아야 수백-수천의 클라이언트만을 처리할 수 있을겁니다. 네

이버나 구글같은 거대 서비스들은 한번에 수십만~수백만의 클라이언트들을 동시에 처리해야하는데... 흠... 한참 모자라네요!

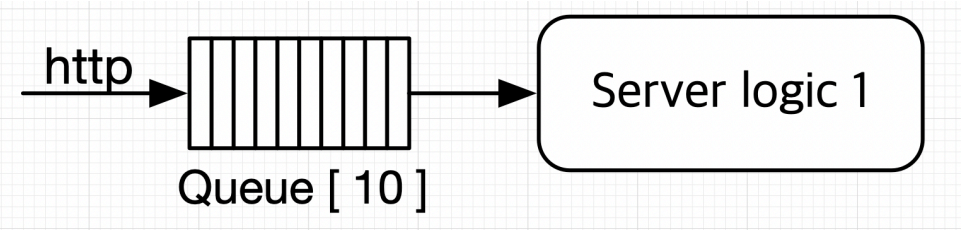
다행히 우리는 좋은 방법을 알고있습니다! 바로 Thread 입니다! 이번에는 server.c 프로그램을 pthread 라이브러리를 활용하여 multi-process기반이 아닌 multi-thread 기반으로 다수의 메시지를 처리할 수 있도록 만들어 봅시다.

- **Step 1)** server.c를 컴파일 하고 실행해 보세요.
- **Step 2)** 웹브라우져를 통해 http://localhost:8090/로 접속을 해보세요. 그리고, server.c의 sleep 주석을 uncomment하시고 여러개의 동시접속을 만들어보세요. accesses.py를 이용하시면 더욱 쉬울겁니다!
- **Step 3)** pthread 라이브러리를 활용하여 server.c를 업그레이드 해보세요! 아마 프로그램 구조가 조금 바뀌게 될겁니다!
- **기대 결과물)** 내용이 반영된 server_02.c, 이에 대한 설명 또는 레포트

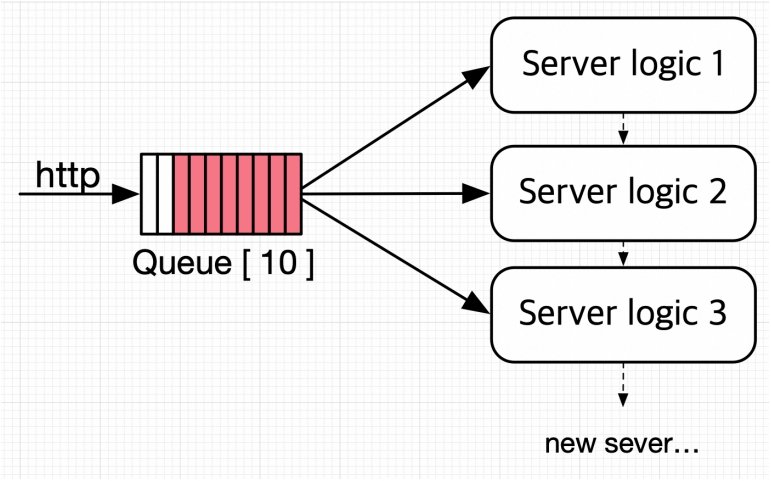
#3. Orchestration and Load-balancing

- 좋습니다! 이제는 어느 정도 더 많은 클라이언트들을 처리할 수 있을 것 같습니다. 그런데 사실 매 요청마다 새로운 process 나 thread를 생성하는 것은 음... 때론 적절하지 않을 수 있습니다. 생성 속도보다 훨씬 빠르게 요청이 온다고 하면 제 시간에 생성에 실패할 수도 있거든요. 그렇다고 미리 process나 thread들을 항상 띄어놓고 기다리는건... 요고도 낭비가 심합니다. 그래서 실제 서비스들은 현재 트래픽의 양에 따라 미리 서버의 용량을 증가시키거나 줄이는 Orchestration 이라는 기능을 갖추고 있으며, Load-balancer를 통해서 부하를 분산시킵니다.

우리도 한번 도입을 해보죠. 여기서는 아주 간단하게 흉내만 내보도록 합시다. 우선 아래와 같이 서버에 큐를 추가하여 어느정도 서버의 처리 지연을 완화해줄 수 있게 해줍시다.

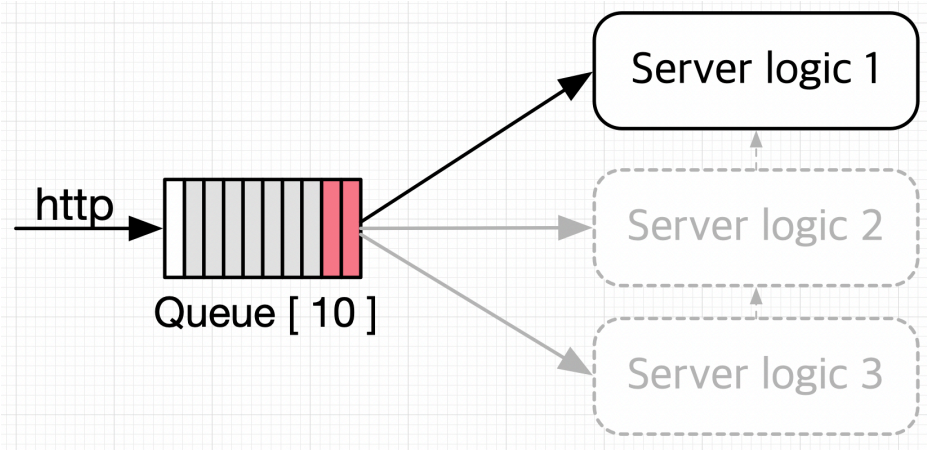


그런데 큐 10개는... 크기가 너무 부족합니다. #1에서 제가 예제로 넣은 사진의 Connection error의 메시지 갯수만 해도 10개를 훨씬 넘으니깐요. 이제 큐에 메시지가 8개 이상이 차 있으면, 자동으로 서버 로직을 증가시키고, 그 로직들 간에 요청을 돌아가며 처리하도록 만들어 줍시다. 예를 들어, 위의 기본 형태에서 메지시 큐에 8개 이상의 메시지가 쌓이면 서버 로직 2가 추가되고,
각 메시지는 1-2번이 번갈아가며 처리하게 됩니다. 또 8개가 쌓이면, 서버 로직 3이 추가되고 로직 1-2-3번들간에 메시지들을 처리합니다. 아래 그림 처럼요! (스케쥴링 챕터에서 곧 배우실텐데, 이런걸 라운드로빈(Round-Robin, RR)이라고 합니다.)



로직이 정상적으로 라운드로빈으로 처리가 되는지, 또는 몇 번 로직이 메시지를 처리하였는지 확인해보기 위해서 Response message에 서버 로직의 ID등을 붙여주면 쉽게 확인이 될겁니다!

마지막으로, 메시지가 오는 속도가 많이 줄어들어 큐에 남아있는 메시지의 수가 2개 이하가 되면, 서버 로직을 갯수를 줄여나가봅시다. 아래 그림처럼요!



물론 서버 로직 1번은 사라지면 안되고 유지가 되어야 합니다. 추가적으로 필요 또는 구현 방식에 따라 각 서버로직에 개별 큐가 필요할 수도 있습니다. 자유롭게 만들어 보세요! :)

- **Step 0)** 아마 2번의 문제에 이어서 시작한다면 조금 수월할 겁니다!
- **Step 1)** server.c에 10칸정도의 크기를 가지는 메세지 큐를 추가하세요.
- **Step 2)** 메시지큐의 용량이 80%가량 차오르면 서버로직을 추가하도록 만들어보세요. 그리고 이후 메시지큐 메시지들은 추가된 로직들끼리 돌아가며 처리(Round-robin)가 되도록 합시다.
- **Step 3)** 반대로 메시지큐의 용량이 20% 이하로 유지되면 서버 로직을 제거하도록 만들어보세요. 당연히 첫번째 로직은 사라지면 안됩니다! 메시지 수신 속도에 따라 서버 로직의 갯수의 변화를 한번 확인해보는 것도 재미있는 결과일겁니다. 뭐 적당히 초당 1개에서 100개 정도로 메세지를 보내보면 될거같네요. 함께 제공해드린 accesses.py에 sleep과 쓰레드 생성 갯수등을 조절하면 하실 수 있으실겁니다!
- **기대 결과물)** 내용이 반영된 server_03.c, 이에 대한 설명 또는 레포트

#4. What is the difference between #1 and #2?

- Python이라면 단 몇줄로 끝날 것을, 1학년때 배우고 켜보지도 않은 C를 다뤄보자니 머리가 지끈지끈 하고 막 교수님이 미워지려고 합니다. 거기다가 윈도우도 아닌 리눅스에서라뇨?! 이야... 남들이 자기처럼 막 vi잘쓰고 그런줄 아는가 봅니다. 거참... 너무하네요!

그래도 한번 해보는게 참 중요한 경험이 될겁니다! 그리고 웹서버의 process 및 thread로의 구현은 구글링을 통해 정말 많은 참고자료를 찾을 수 있으니 잘 해내실 수 있을겁니다! 그리고 뭐 100% bug free 하게 만들 것 까진 없습니다. 동작을 이해하고 왜 이렇게 되는지를 이해하는 것이 더더욱 중요합니다.

이러한 맥락에서, 4번은 코딩 문제가 아닙니다! server.c의 개략적인 동작과, #1과 #2의 접근방법에 대한 차이점과 장단점 등을 설명해주세요. 특히 변수와 메모리, 프로그램의 작동 형태등을 함께 설명해주세요! (아마 위의 #1-3에 대한 설명 및 레포트와 내용이 많이 겹치기도 하겠네요!)

- **Step 0)** OS 프로세스와 쓰레드 강의내용을 참고하세요!
- **Step 1)** server.c의 대략적인 동작, #1의 접근방법과 #2의 접근방법의 장단점, 동작형태등을 적당히 적어봅시다
- **기대 결과물)** #1-#2에 대한 설명
