# [506489] System Programming (F'18)

## Term Project (Week 3, Final) Report

## 팀 정보

| No. | 이름 | 학번 |
|---|---|---|
| 1 | 임병준 | 20165157 |
| 2 | 장세영 | 20165160 |
| 3 | 이은지 | 20165152 |
| 4 | | |
| 팀 이름: 5 조 | | |

## <목차>

팀 정보 ..... 1
결과물 제출 ..... 2
보고서 작성 가이드 ..... 2
Part I: 최종 보고서 ..... 3
Part II: 데모/시연 시나리오 및 결과물 ..... 6
Part III: 프로젝트 진행중 발생한 문제점 및 해결방안 ..... 9
Part IV: 개선방향 ..... 11

* 보고서 작성이 완료되면 <목차>를 업데이트 해 주세요. (방법: "목차" 클릭>"목차 업데이트">"목차 전체 업데이트")

## 결과물 제출

- SmartCampus 에 업로드 해야 하는 제출물 목록은 다음과 같습니다.
  - Report(프로젝트 보고서), PDF 형식
  - 프로젝트 발표자료 (ppt, pptx, pdf 등)
  - GitHub 프로젝트를 다운로드 한 zip 파일
- 제출 기한을 넘기면 자동으로 0 점 처리됩니다.
- 팀 과제이며, 팀별로 한 명만 제출하면 됩니다.

* 보고서 및 발표자료도 GitHub 에서 관리해야 합니다!!!

## 보고서 작성 가이드

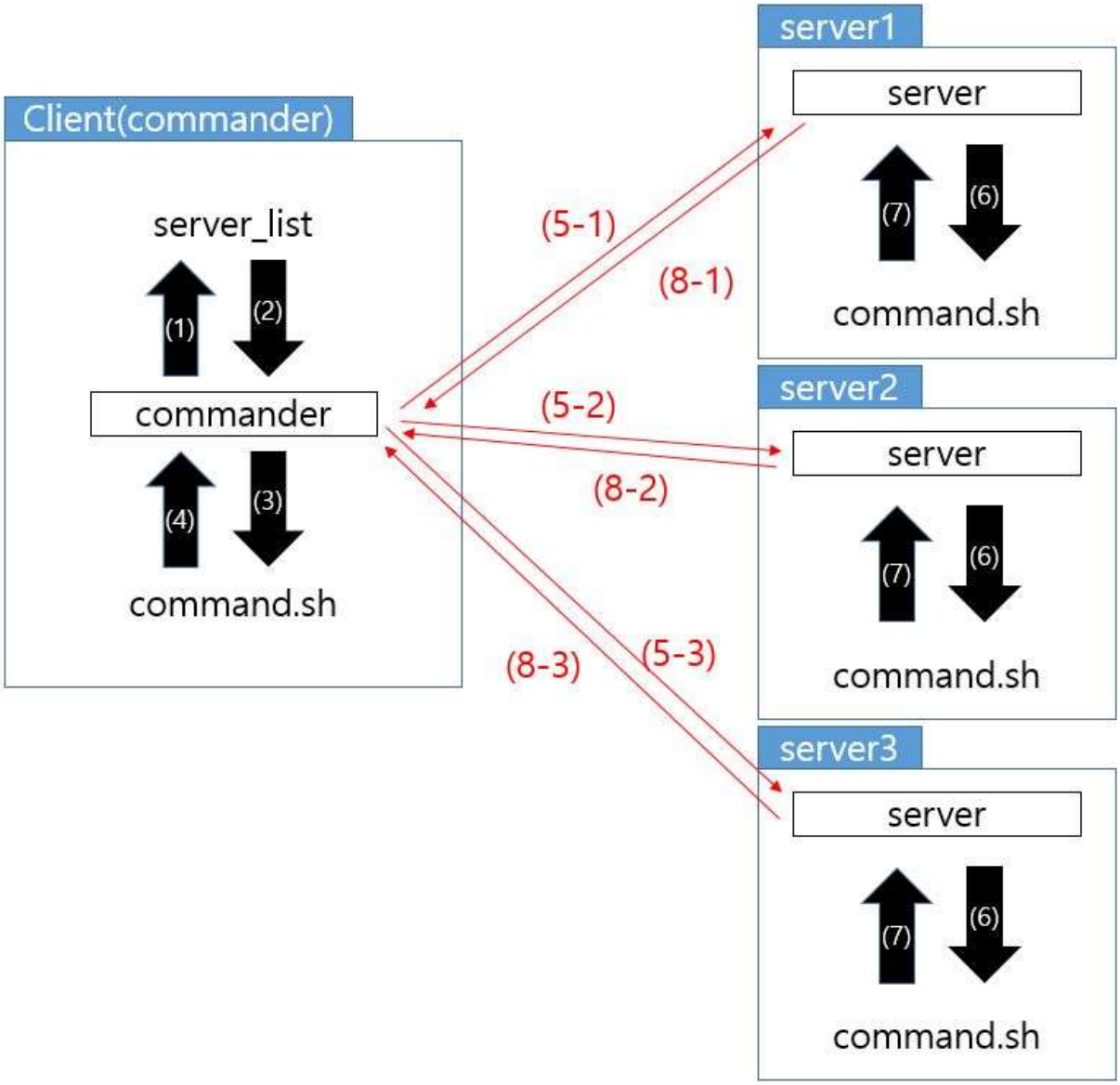
- Week 3 시작: 2018. 12. 12. (Wed) 1:00pm
- Week 3 마감: 2018. 12. 17. (Mon) 12:59pm
- 매 주차별로 반드시 1 회 이상 GitHub 에 commit 한 내역이 있어야 합니다. (팀별로 1 회 이상, 개인별 1 회 아님). GitHub 에 commit 한 내역이 없을 경우, 큰 감점을 받게 됩니다.

본 보고서는 총 3 개의 파트로 구성되어 있습니다.

- Part I: 최종 보고서
  - 1 주차 보고서 내용 전체를 포함하여 작성하고, 그 동안 추가/수정된 내용이 있다면 그에 맞게 보고서를 수정해야 합니다.
- Part II: 데모/시연 결과물
  - 개발한 프로그램이 정상적으로 구동된다는 것을 보여주세요.
  - 데모/시연을 위해 사용한 시나리오를 자세하게 설명하고, 해당 시나리오에 대한 단계별 데모 결과물(예: 스크린샷)을 첨부하거나 또는 데모 영상을 촬영해서 첨부파일로 제출해도 됩니다.
- Part III: 프로젝트 진행중 발생한 문제점 및 해결방안
  - 2 주차 보고서 내용중 "프로젝트 진행 중 발생하는 문제점 및 해결방안"에 대한 내용을 포함하는 내용으로 작성하세요.
  - 그 외에 추가로 발생한 문제점이 있었다면 해당 문제점 및 해결방안을 기술하세요.
- Part IV: 개선방향
  - 개발 결과물을 앞으로 어떻게 개선할 수 있을지에 대한 의견을 서술하세요.

# Part I: 최종 보고서

- github 주소 : https://github.com/BJ-Lim/SystemProgramming
- 제목 : 여러 개의 다른 서버 일괄 제어 프로그램
- 선정 동기 : 일반적으로 리눅스 서버는 SSH 를 통해 한 대의 서버에 접속이 가능합니다. 동시에 여러 대의 서로 다른 서버에 SSH 를 사용하여 개별적으로 접속은 가능하지만 일괄 제어는 어렵습니다. 여러 대의 서버를 관리하는 툴은 존재하지만, 이 주제를 통해서 한 학기 동안 배운 시스템프로그래밍의 내용을 정리하고자 합니다.
- 내용 : 한 대의 서버가 여러 대의 서로 다른 서버에 일괄적으로 명령을 내리고 결과를 돌려받는 프로그램 작성
- 목표 : 한 대의 서버가 여러 대의 서로 다른 서버에 일괄적으로 명령을 내리고 결과를 돌려받는 프로그램 작성을 통해 시스템프로그래밍의 관점에서 '효율성' 생각해보기
- 개발/구현 내용
  - 클라이언트
    - 여러 대의 서로 다른 서버에 명령을 내릴 수 있는 소켓 프로그램 작성
    - 여러 대의 서버에 대한 정보(IP/Port) 파일을 입력으로 받아 처리
  - 서버
    - 클라이언트에게 명령어를 받는 소켓 프로그램 작성
    - 클라이언트에게 전송 받은 쉘 스크립트를 실행하고 결과를 전송
- 구성도(다음 페이지)




(1,2) 연결해야 할 서버 리스트를 파일에서 입력

(3,4) 공유메모리를 생성하고 수행할 명령어를 메모리에 로드

(5) commander 는 fork 후 서로 다른 프로세스로 소켓 생성 및 연결 / 공유메모리의 수행 명령어 전송

(6, 7) 수신 받은 명령어를 파일로 저장 / 파일의 권한 변경 / 출력 결과를 파일로 redirect

(8) 서버는 실행 결과 파일의 내용을 클라이언트로 각각 전송 / commander 는 수신 받은 내용을 출력

- 기대 효과 : 여러 대의 서버가 동시에 같은 동작을 수행해야 할 때 쉽게 활용 가능하며 이를 응용하면 여러 대의 서버 자원 관리를 용이하게 할 수 있다. 또한 tutorial 을 제공하기 때문에 쉽게 이 프로젝트의 과정을 따라하며 이해해 볼 수 있다.

# Part II: 데모/시연 시나리오 및 결과물

- 깃허브 : https://github.com/BJ-Lim/SystemProgramming/blob/master/tutorial/03_demo.md 에서 더 자세한 실행을 보실 수 있습니다.
- 시나리오
  1) 클라이언트에서 다수의 서버에 명령을 전송합니다. 이때 명령은 서버별로 다른 출력결과를 가질만한 것으로 선정합니다.
  2) 각각의 서버는 명령을 전송받아 자신의 서버에서 실행합니다.
  3) 각각의 서버는 명령의 수행이 완료되면 클라이언트에게 결과를 전송합니다.
  4) 클라이언트는 결과를 수신받고 출력합니다.
- 준비
  1) 각 서버에 전송할 명령어를 파일로 정리합니다. 예시는 다음과 같습니다.

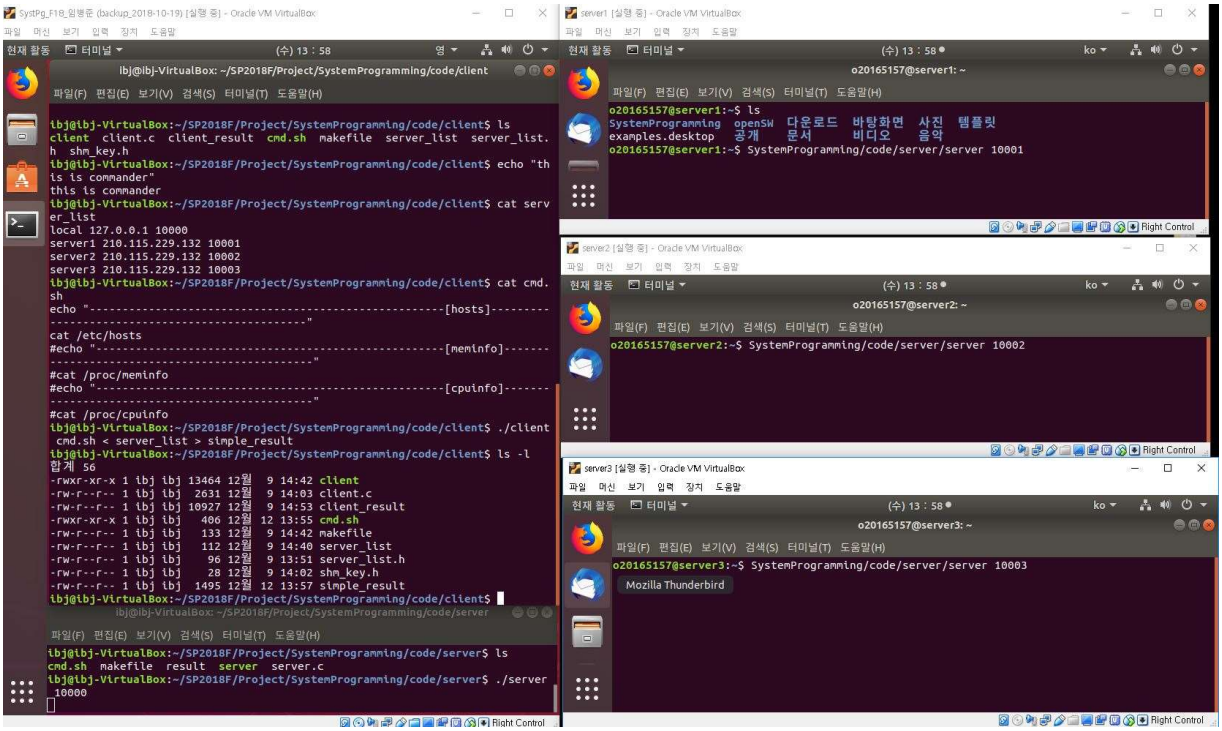
```
echo "------------------------------------------------------[hosts]-------------------------------------------------"
cat /etc/hosts
echo "------------------------------------------------------[meminfo]-----------------------------------------------"
cat /proc/meminfo
echo "------------------------------------------------------[cpuinfo]-----------------------------------------------"
cat /proc/cpuinfo
```

  2) 서버 리스트를 작성합니다. 예제는 다음과 같습니다.

```
local 127.0.0.1 10000
server1 210.115.229.132 10001
server2 210.115.229.132 10002
server3 210.115.229.132 10003
```

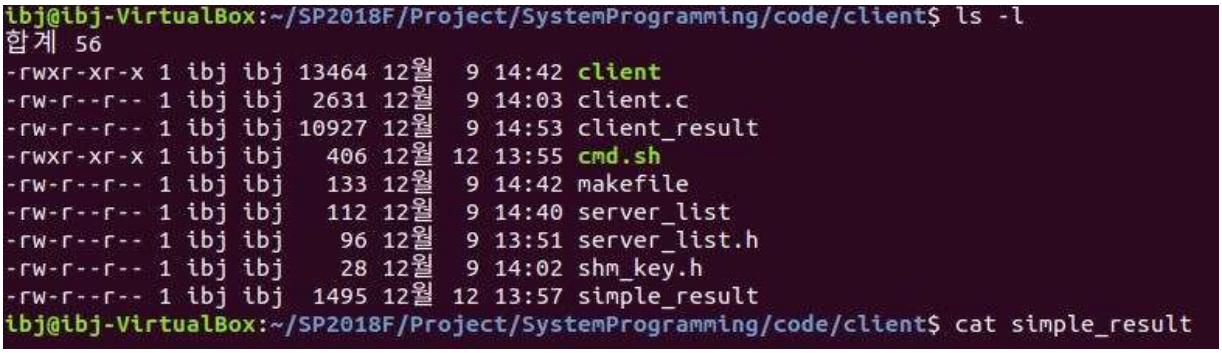
  local 은 localhost 이고, server1~3 은 공인 IP 로 포트포워딩된 서로 다른 가상머신입니다 이때 server1~3 은 호스트명이 각각 server1~3 으로 셋팅되어 있습니다..

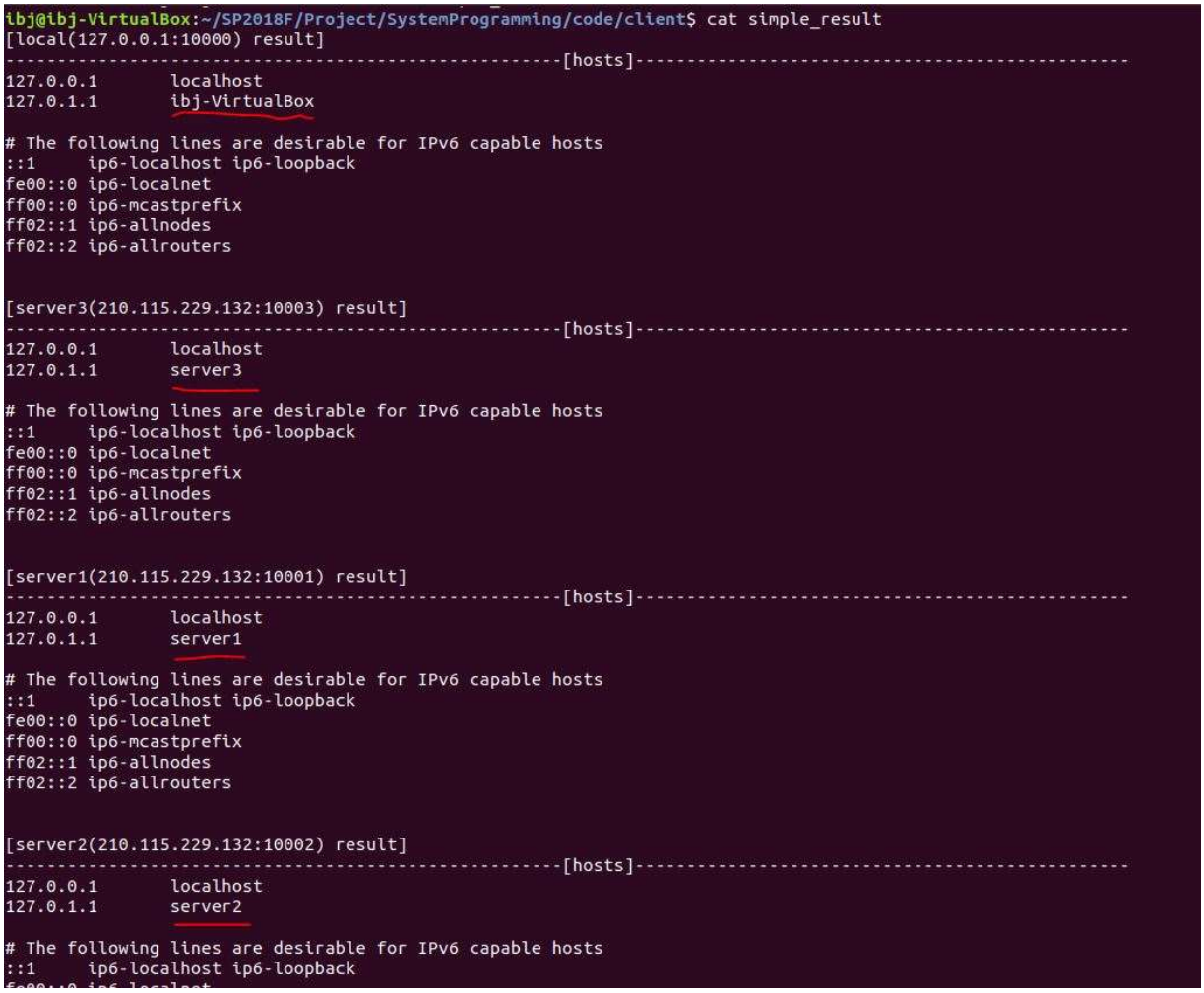
- 실행
  1) 서버에서 서버 프로그램을 실행합니다.(server1, server2, server3, localhost)
  2) 클라이언트에서 명령을 실행합니다.




<1, 2 번의 결과>

3) 결과 파일을 확인합니다.






<3 의 결과>

각 서버의 /etc/hosts 파일이 다른것을 확인할 수 있습니다.

- 주의 사항
  - 출력 결과는 순서가 다를 수 있습니다. 내부적으로 fork()를 사용했기 때문입니다.

# Part III: 프로젝트 진행중 발생한 문제점 및 해결방안

1) 명령어(쉘 파일)을 전송할 때 버퍼 크기를 고정으로 잡아야 하는가?
   - 버퍼의 크기를 고정으로 잡으면, 파일의 크기가 커질 시 파일을 잘라서 읽어야 하는 문제가 있습니다.
   - 버퍼의 크기를 동적으로 잡으면 파일의 크기를 미리 알아야 합니다.(선택)
2) 명령어(쉘 파일)의 크기를 어떻게 알 수 있는가?
   - foepn 및 fseek 를 사용할 수 있습니다. 하지만 이 방법은 linux 스타일이 아닙니다.
   - stat family 를 이용하여 간단하게 구할 수 있습니다.(선택)
3) 수신측(서버측)에서는 명령어 파일 버퍼의 크기를 어떻게 정해야 하는가?
   - 서버측은 고정 크기의 버퍼를 사용합니다. 하지만 이는 비효율적입니다.
   - 전송을 2 번에 나눠서 합니다. 첫 번째는 파일의 크기, 두 번째는 파일의 내용입니다.(선택)
4) 연결을 할 때 단일 프로세스를 사용할 것인가, 다중 프로세스를 사용할 것인가?
   - 연결시 싱글 프로세스를 사용하면, [클라이언트와 서버 1 연결 -> 명령어 전송 -> 서버 1 에서 명령어 수행 후 클라이언트로 전송 -> 클라이언트 수신 -> 연결 종료]의 시간동안 해당 프로세스는 블락되며, 다른 연결이 불가능합니다. 이때 서버들이 처리하는 시간동안도 기다려야 하기 때문에 몹시 비효율적입니다.
   - 연결시 멀티 프로세스를 사용하면 훨씬 효율적으로 처리할 수 있습니다. (선택)
5) 멀티 프로세스를 사용하여 연결할 때 명령어(쉘 파일)의 내용을 어떻게 로드할 것인가?
   - 첫 번째 방법으로는 각각의 child 프로세스가 쉘 파일을 각각 열어 내용을 읽는 것입니다. 하지만 이 방법은 쓸데없이 중복 작업을 하게 됩니다.
   - 두 번째 방법은 부모의 프로세스에서 미리 쉘 파일을 읽어 버퍼에 저장해 놓는 방법입니다. 하지만 이 방법은 child 가 fork 될 때 해당 버퍼가 모두 복사되어야 하므로 이 또한 낭비입니다.
   - 세 번째 방법은 부모의 프로세스에서 공유메모리를 생성하고 쉘 파일을 로드한 뒤 공유메모리에 대한 포인터만 child 가 복사하는 방법입니다. 메모리적인 측면에서 매우 효율적인 방법입니다.(선택)
6) 부모 프로세스는 자식 프로세스를 wait 해야 하는가?

- 이 프로그램은 공유메모리를 사용하는데 부모 프로세스가 자식 프로세스를 기다리지 않으면 공유메모리의 해제 시점을 확정할 수 없습니다. 따라서 모든 자식 프로세스가 종료된 뒤 공유메모리를 해제해야 하므로 부모는 자식 프로세스가 모두 종료될 때까지 기다려야 합니다.

## Part IV: 개선방향

1) 각각의 서버 프로그램은 모두 계속 켜져있어야 합니다. 따라서 데몬으로 작동하는 것이 더 효율적입니다.
2) 쉘 코드를 전송하므로 보안에 취약할 수 있습니다. 따라서 접근 제어를 할 필요가 있습니다. 서버에서 명령어를 처리하는 계정을 따로 분리하고, 웹 서버와 마찬가지로 로그인 쉘을 얻지 못하도록 조치하는 것이 더 안전할 수 있습니다.
3) 현재는 각각의 서버에서 소켓프로그램을 실행하기 이전에 약간의 작업이 필요했습니다. 따라서 포트 포워딩 등의 설정과정을 자동화하는 스크립트를 작성하여, 소켓 프로그램이 실행되기 이전에 이 스크립트를 먼저 실행할 수 있도록 하면, 많은 서버를 관리하는데 더 편할 수 있습니다.
4) 클라이언트가 각 서버에 명령을 전송할 때, 다양한 옵션을 추가하면 더 좋은 프로그램이 될 수 있습니다.

[끝]