프로젝트 보고서

Simple Pow

네트워크프로그래밍(가)반

1 조

김건학(20192804), 안아영(20201737), 이경호(20192854)

**목차**

**1. 서론**

1.1 프로젝트 개요

1.2 목적

**2. 프로토콜 flow 및 패킷 구조**

2.1 멀티프로세스의 역할

2.2 Main server와 Working server 간의 프로토콜 flow

2.2.1 정상 상황에서의 flow

2.2.2 예외 처리 상황에서의 flow

2.3 Main server와 Working server 간의 패킷 payload

2.4 네트워크 브릿지 방식을 사용한 가상머신 네트워크 소통

**3. 주요 API**

3.1 API 기능

3.2 OpenSSL 변수 및 리턴 값

**4. 난이도별 실행시간 차이**

4.1 난이도 7

4.2 난이도 8

**5. Pow nonce**

5.1 단독 실행

5.2 분산 실행

**6. 결론**

**1. 서론**

**1.1프로젝트 개요**

- Main server와 Working server 간의 통신을 위한 프로토콜 플로우와 패킷 구조를 설계 하고, 멀티프로세스 또는 쓰레드를 사용하여 역할을 분배하는 시스템을 구현한다.

- 프로토콜 플로우 및 패킷 구조 섹션에서는 Main server와 Working server 간의 통신 절 차를 설명하고, 정상 상황 및 예외 처리 상황에서의 플로우, 프로토콜 통신을 위한 패킷 의 구조와 페이로드를 정의한다.

- 각 API가 제공하는 주요 기능을 설명하고, 어떤 동작을 수행하는지를 명시한다. API에서 사용되는 입력 변수와 그 역할, API가 반환하는 결과 값과 해당 값의 의미를 설명한다.

- 프로젝트는 단독 실행과 분산 실행의 비교를 통해 성능과 실행 시간의 차이를 분석하고 난이도 7 및 난이도 8에서의 단독 실행과 분산 실행 간의 실행 시간 차이도 측정하여 비교 분석한다. 이를 통해 분산 실행이 프로젝트 성능에 미치는 영향을 평가한다.

**1.2 프로젝트 목적**

- 효율적인 서버 간 통신 : Main server와 Working server 사이 통신을 최적화하여 데이터 교환과 작업처리의 효율성을 높인다.

- 역할 분배 : 작업을 멀티프로세스로 분배하여 병렬처리를 이루어 시스템 성능을 향상시킨다.

- 프로토콜 플로우, 패킷 구조 설계 : 서버간 통신 절차, 패킷 구조를 정의하여 효율적인 데이터 전송과 안정성을 보장한다.

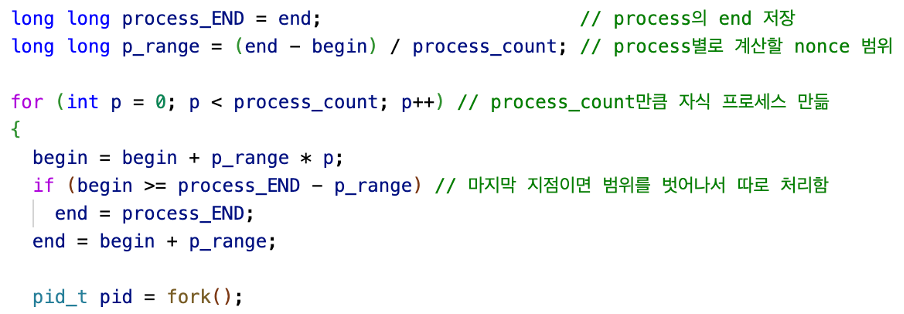
효율적인 서버 간 통신과 역할 분배를 위한 프로토콜 플로우와 패킷 구조 설계, 멀티 프로세스, 멀티 스레드를 활용한 시스템 구현을 통해 전반적인 성능을 개선한다.

**2. 프로토콜 flow 및 패킷 구조**

* 1. **멀티프로세스의 역할**

프로세스 간의 업무 분담을 통해 계산 작업을 병렬화하여 동시에 처리한다.

* 메인 프로세스(부모 프로세스)
* mainserver이 전송한 도전 과제(challenge)와 복잡도(complexity) 등의 정보를 읽는다.
* 프로세스 수(Process\_count)를 읽는다.
* 자식 프로세스를 생성하고, 각 자식 프로세스에 할당된 nonce 범위를 계산하여 전달한다.
* 정답을 받은 Mainserver가 전송한 signal을 받으면 모든 process들을 종료한다.



프로세스 수 (process\_count)에 따라 여러 개의 자식 프로세스를 생성하고, 각 프로세스는 할당된 nonce 범위를 통해 계산 작업을 분담한다.

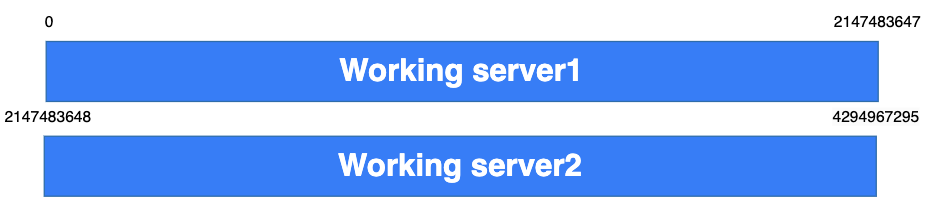
* 자식 프로세스
* mainserver 소켓에 연결된 상태로 독립적으로 실행됩니다.
* 각 프로세스는 자신이 속한 프로세스의 nonce 범위 내에서 작업을 수행한다.
* 주어진 범위에서 nonce 값을 계산하고, 정답을 찾을 때까지 반복한다.
* 정답을 찾았을 경우, 해당 정보를 mainserver로 전송하고 종료한다.

**e.g.** Working server의 개수가 2개이고 각 working server의 자식 Process개수가 5개일때의 업무분담

1. Nonce은 0 ~ 2^32-1(4294967295)값을 가질 수 있다.



2. Nonce범위를 Working server의 개수인 2로 나눠 working server nonce range를 구해 working server끼리 업무를 분담한다. (4294967295 / 2 == 2147483647)



3. Working server nonce range를 자식 process개수인 5로 나눠 process끼리 업무를 분담한다. (2147483647 / 5 == 429496729)

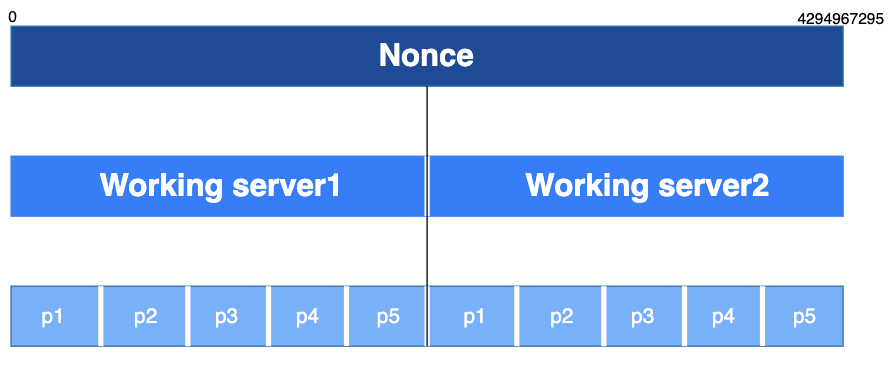
* Working server1의 process





* Working server2의 process



전체 nonce 범위는 아래 그림과 같다. 

**2.2** **Main server와 Working server 간의 프로토콜 flow**

프로토콜 flow

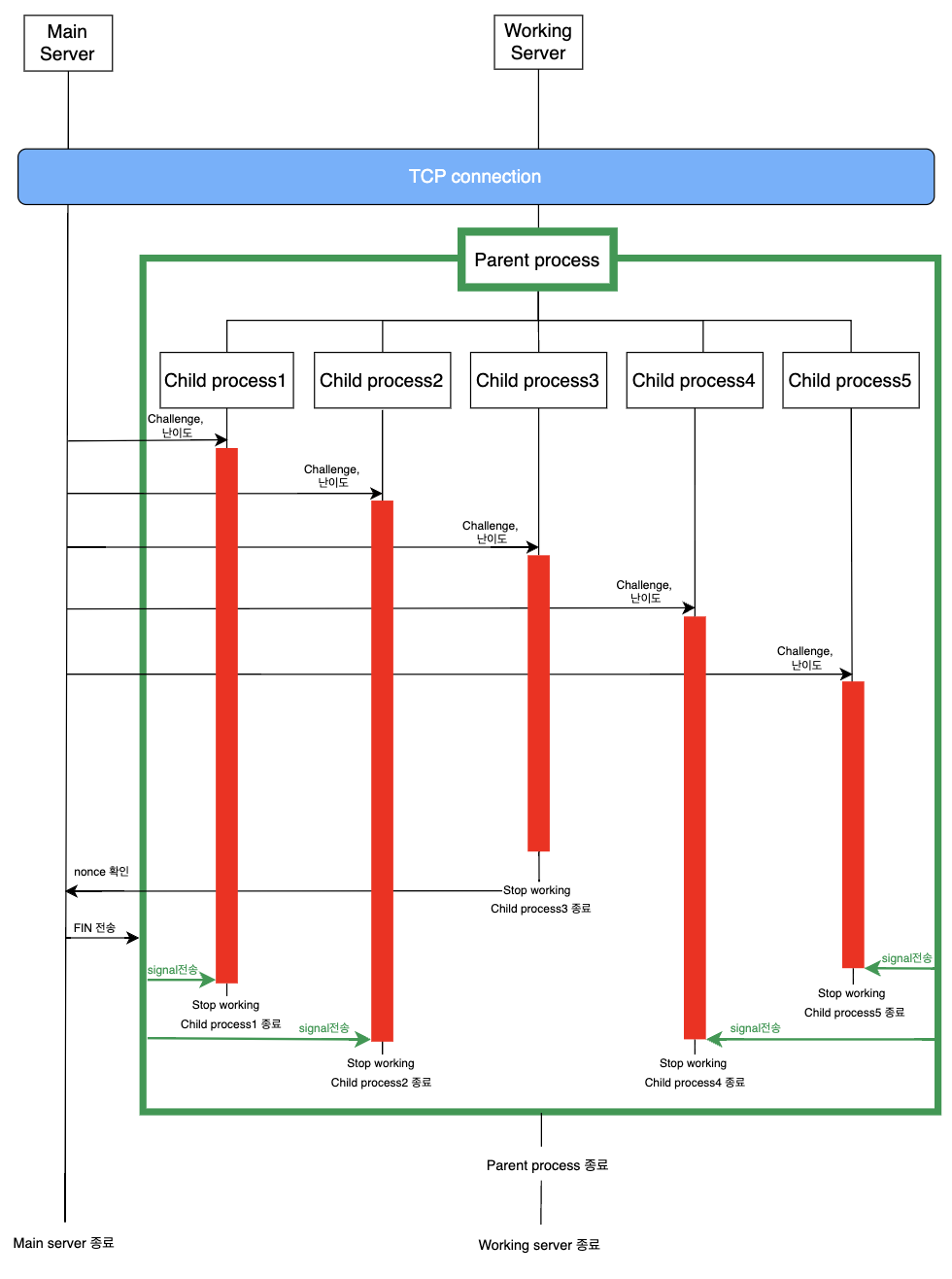
* Main server는 listening중인 Working server들에게 작업을 요청하기 위해 연결(connection)을 수립한다.
* Main server는 작업에 대한 정보(challenge, complexity, begin, end)를 Working server로 전송한다.
* Working server는 작업을 분할하여 여러 프로세스를 생성한다.
* 각 프로세스는 주어진 범위 내에서 작업을 수행하고, 결과를 찾으면 Main server로 결과를 전송한다.
* Main server는 결과를 받아 확인하고, 연결을 종료한다.
* Fin을 받은 부모 프로세스가 모든 자식프로세스들에게 signal을 줘 종료한다.

예외 처리 상황

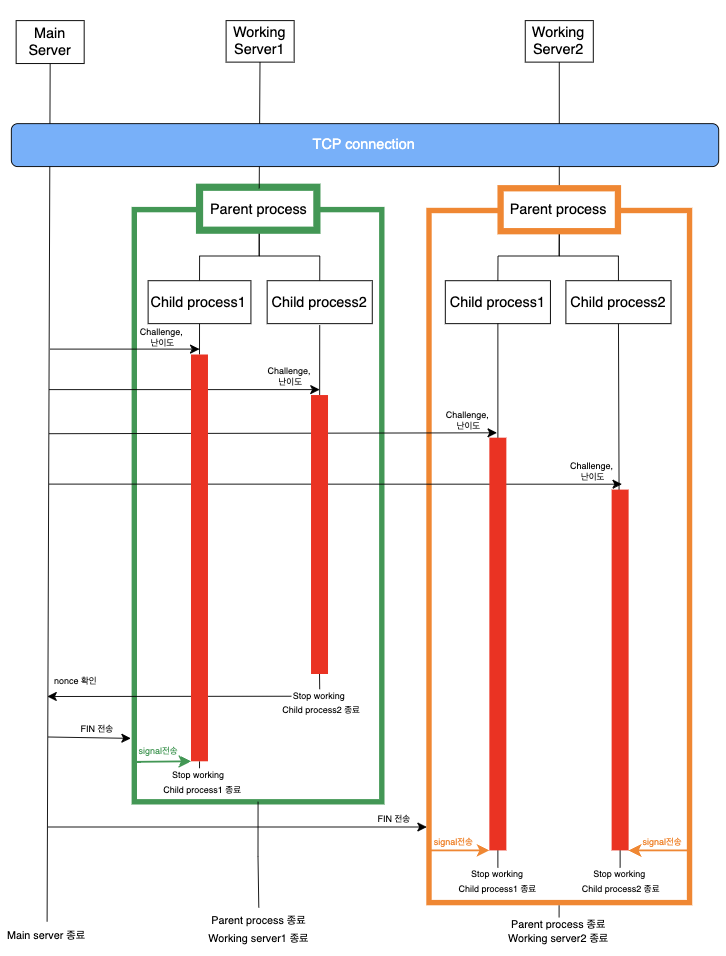
* 입력 인수의 개수가 틀리거나 다른 오류가 발생하면 errProc을 실행해 오류 메시지를 출력하고 중단한다.
  + epoll\_create
  + socket
  + bind
  + listen
  + connect
  + accept
  + epoll\_ctl
  + read
  + fork
  + epoll\_wait

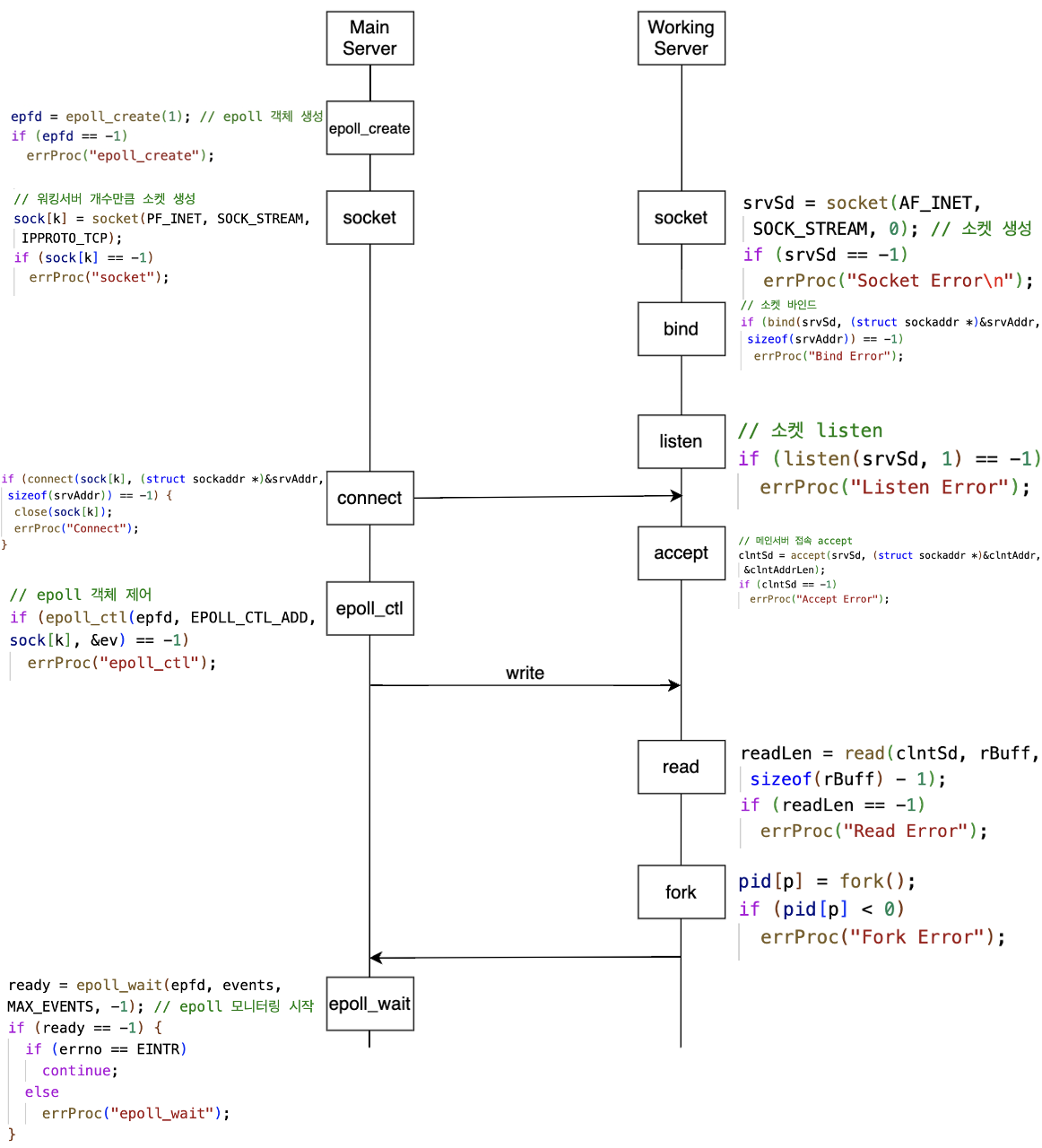
**2.2.1 정상 상황에서의 flow**

- Working server가 1개인 경우



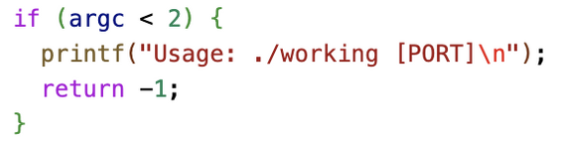
- Working server가 2개인 경우



* + 1. **예외 처리 상황에서의 flow**
* 프로그램 실행도중 errProc가 일어날 수 있는 모든 경우를 나타낸 flow

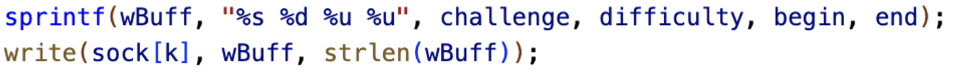
위 그림과 같은 상황에서 errno에 해당하는 오류 메시지를 출력하고 프로그램을 종료한다.

* 프로그램 실행 시 전달된 명령행 인수의 개수를 확인하고, 인수 개수가 2보다 작을경우

"Usage: ./working [PORT]"라는 오류 메시지를 출력하고 프로그램을 종료한다. 

**2.3 Main server와 Working server간의 packet payload 정의**

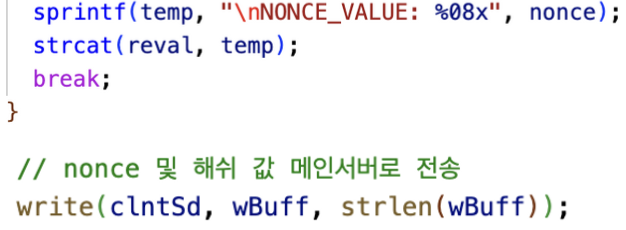
* Main Sever에서 Working server로 전송하는 packet payload
  + challenge : 해시 값을 계산하기 위해 주어지는 학번||학번||학번, 이름||이름||이름 형태의 문자열 (”201928042019285420201737”, “kimgunhakahnahyeongleekyungho”)
  + difficulty : 해시 값의 난이도를 나타내는 정수 (7, 8)
  + begin : 각 Working server에서 계산할 nonce의 시작 값
  + end : 각 Working server에서 계산할 nonce의 끝 값



위 코드로 Main server에서 challenge, difficulty, begin, end를 wBuff에 저장하여 Working server로 전송한다.

* Working server에서 Main server로 전송하는 packet payload
  + hash value : 계산된 해시 값
  + nonce : 해시 값의 난이도를 충족하는 nonce값





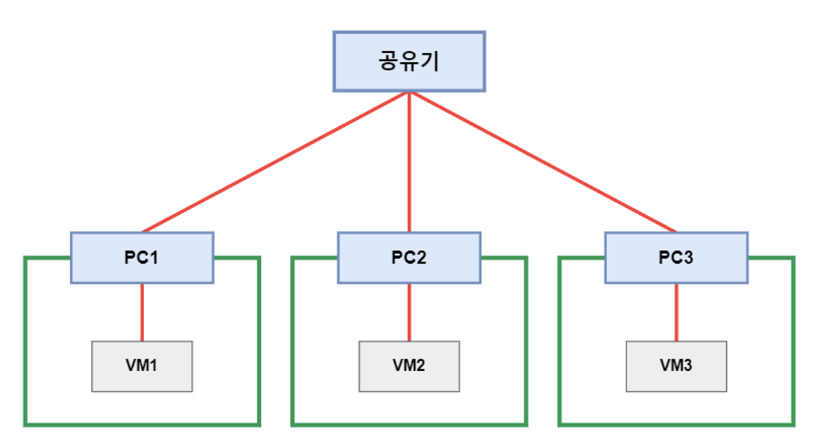
위 코드로 Working server에서 유효한 값을 지닌 nonce값과 hash를 문자열 wBuff(reval)에 붙이고 Main server로 전송한다.

**2.4 네트워크 브릿지 방식을 사용한 가상머신 네트워크 소통**

3대의 PC 각각이 가상머신(VM)을 사용하고 있는 상황에서 각 PC에서의 VM들의 네트워크 소통을 가능하게 하기 위하여 네트워크 브릿지 방법을 사용하였다.

네트워크 브릿지 방식은 가상머신을 포함한 모든 PC를 하나의 망으로 묶어 공유기 망에 있는 모든 PC 및 VM들의 네트워크 소통을 가능하게 하는 방식이다.

네트워크 브릿지 방식으로 가상머신에도 하나의 공유기 아래 있는 PC처럼 똑같이 IP주소를 할당시켜주어 다른 PC들의 VM과의 네트워크 소통을 가능하게 하였다



**3. 주요 API**

**3.1 API 기능**

본 프로젝트에서는 OpenSSL, epoll을 API로 사용한다.

* OpenSSL : C언어를 기반으로 암호화 및 보안 프로토콜 구현에 사용되는 오픈 소스 라이브러리로, 다양한 암호화 작업 및 인증서 관리, 안전한 통신, 디지털 서명 등과 같은 기능을 제공한다.

OpenSSL은 암호화 및 복호화를 위한 함수를 제공하며, 데이터의 기밀성을 보호할 수 있다. 대표적으로 데이터통신을 위한 TLS, SSL 프로토콜을 구현하는 기능을 제공한다. 이를 통해 데이터의 기밀성, 무결성, 인증 등을 제공하고 안전한 네트워크 통신을 돕는다.

SSL은 암호화 통신을 위해 대칭키, 비대칭키 방식을 혼용하여 사용하며 대칭 암호화는 동일한 키를 사용하여 데이터를 암호화하고 복호화하는 반면, 비대칭 암호화는 공개키와 개인 키를 사용하여 데이터를 암호화하고 복호화한다는 차이가 존재한다.

SHA-256과 같은 해시 함수를 제공하여 이를 통해 데이터를 검증하거나 중요 정보를 안전하게 저장할 수 있도록 돕는다. 인증서의 생성, 검증, 서명 등과 같은 인증서 관리 작업을 지원하여 데이터의 인증과 신뢰성을 보장하며 이와 같이 OpenSSL은 암호화, 복호화, 디지털 서명, 해시함수, 인증서 관리 등 다양한 보안 관련 작업에 활용된다.

* Epoll : 리눅스 운영 체제에서 사용되는 이벤트 알림 인터페이스이다. 이는 다수의 파일 디스크립터(소켓, 파일 등)를 효율적으로 관리하고 비동기식 I/O 작업을 처리하는 데 사용된다. epoll은 이벤트 기반 입출력(I/O) 모델을 구현하는 메커니즘 중 하나로, 다른 모델인 select와 poll과 비교하여 더 효율적이다. select와 poll은 사용자가 관심 있는 파일 디스크립터를 직접 지정해야 하지만, epoll은 사용자가 관심 있는 이벤트에 대해 등록만 해도 된다.

**3.2 OpenSSL 변수 및 리턴 값**

SHA-256 OpenSSL의 해시 함수 중 하나로, 주어진 데이터에 대한 256비트(32바이트) 길이의 해시 값을 생성한다. 이 프로젝트에서는 SHA-256 해시 함수를 사용하여 입력 데이터의 해시 값을 계산하고, 난이도에 맞는 조건을 확인하는데 사용되었다.

* SHA256\_CTX OpenSSL 라이브러리에서 SHA-256 해시를 계산하기 위한 컨텍스트 구조체로, SHA-256 알고리즘의 내부 상태 및 중간 결과가 저장된다.
* SHA256\_Init SHA-256 해시를 계산하기 위한 컨텍스트를 초기화하여 해시 알고리즘을 수행하기 위한 내부 상태 및 중간 결과를 설정한다.
* SHA256\_Update 입력 데이터를 컨텍스트에 추가하여 해시 값을 계산한다.
* SHA256\_Final 컨텍스트에 추가된 모든 데이터를 기반으로 최종 해시 값을 계산하고 결과를 출력 버퍼에 저장한다.

epoll의 epoll\_create, epoll\_ctl, epoll\_wait를 사용하였다.

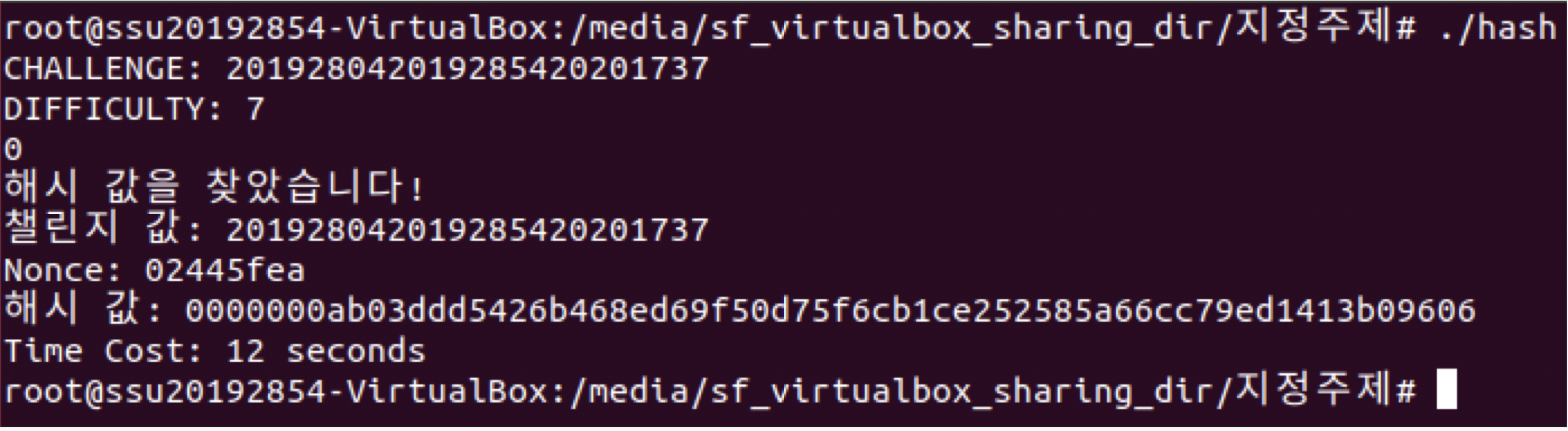
* epoll\_create : epoll 인스턴스를 생성. 이 함수는 epoll 파일 디스크립터를 반환하며, 이를 통해 epoll 인스턴스에 대한 조작이 가능해진다.
* epoll\_ctl : epoll 인스턴스에 파일 디스크립터를 추가하거나 제거한다. 이 함수를 사용하여 관심 있는 이벤트를 등록하거나 제거할 수 있다.
* epoll\_wait : 등록된 파일 디스크립터에 대한 이벤트가 발생할 때까지 대기하며, 이벤트가 발생하면 해당 이벤트를 반환. 이 함수를 반복 호출하여 이벤트를 처리할 수 있다.

epoll은 다수의 파일 디스크립터를 효율적으로 처리할 수 있으므로, 대규모 네트워크 서버나 다중 클라이언트를 처리하는 서버에서 많이 사용된다. 본 프로젝트에서는 MainSever가 여러 workingserver에 연결되었을 때, 전달되어 오는 해쉬값과 nonce를 입력받기위해 사용하였다.

**4. 난이도별 실행시간 차이**

* 1. **난이도 7**
* Challenge : 학번||학번||학번 (201928042019285420201737)

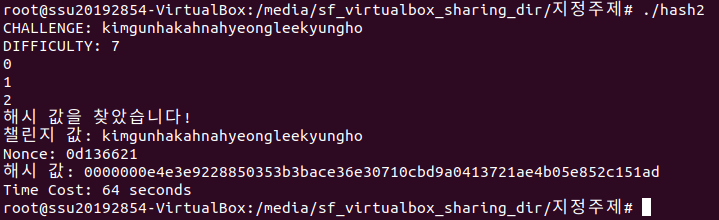
1. 단독실행 🡪 12 seconds

****

1. 분산 실행🡪 11 seconds

* Challenge : 이름||이름||이름 (kimgunhakahnahyeongleekyungho)

1. 단독 실행 🡪 64 seconds



1. 분산 실행 🡪 0 seconds



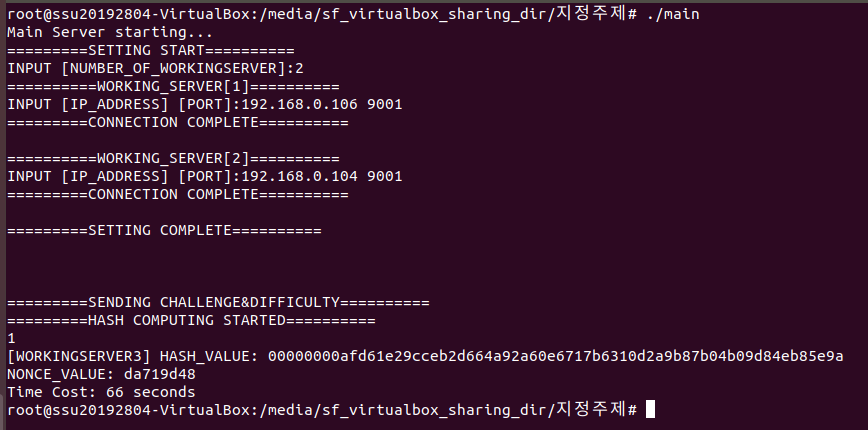
**4.2 난이도 8**

* Challenge : 학번||학번||학번 (201928042019285420201737)

1. 단독실행 🡪 1028 seconds

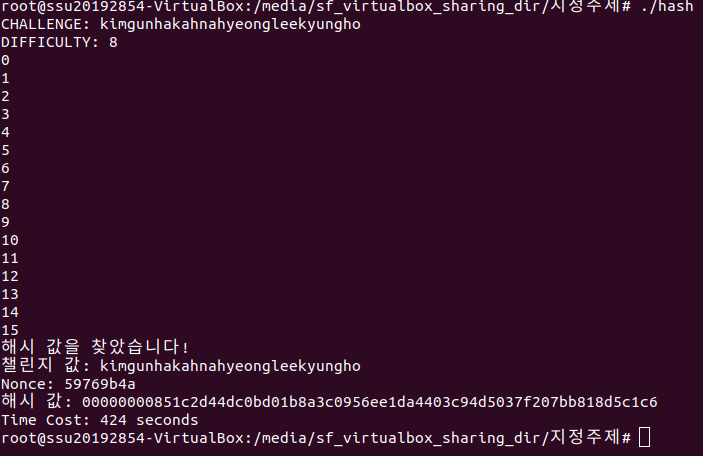


1. 분산 실행 🡪 66 seconds

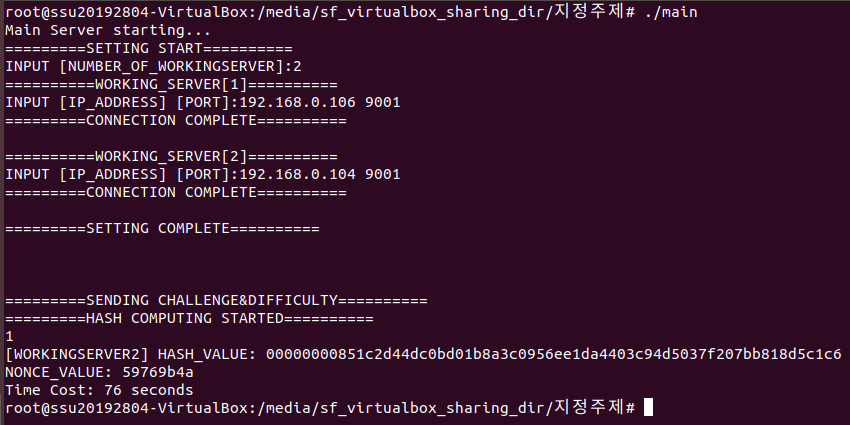


* Challenge : 이름||이름||이름 (kimgunhakahnahyeongleekyungho)

1. 단독 실행 🡪 424 seconds



1. 분산 실행 🡪 76 seconds



**5. POW nonce**

**5.1 단독 실행**

|  |  |  |  |
| --- | --- | --- | --- |
| **nonce** | | **난이도** | |
| **7** | **8** |
| **challenge** | **학번** | 02445fea | da719d48 |
| **이름** | 0d13662 | 59769b4a |

**5.2 분산 실행**

|  |  |  |  |
| --- | --- | --- | --- |
| **nonce** | | **난이도** | |
| **7** | **8** |
| **challenge** | **학번** | 02445fea | da719d48 |
| **이름** | b3525155 | 59769b4a |

**6. 결론**

* 회고

1. 본 프로젝트를 진행하면서 멀티스레드와 멀티프로세스를 병행하려고 처음에 고안했다 하지만 생각보다 멀티프로세스와 멀티스레드가 동시에 실행될 때의 오버헤드가 너무 커서 가상머신에서 I/O가 불가능할 정도로 렉이 걸렸다. 그래서 멀티프로세스만 이용하기로하였다. 멀티프로세스와 멀티스레드를 둘 다 구현해보고 멀티프로세스가 코드가 깔끔하게 나올 것 같았기 때문이다. 두 가지다 구현해 보면서 좀 더 공부를 해볼 수 있었다.

2. hash함수를 구현할 때, 난이도를 검사하는 부분에서 실제로는 4비트가 0이면 난이도가 1이지만, 1byte가 0일 때라고 오해를 하여 실제로 난이도 14인 경우를 약 9시간 동안 구동하였다. 하지만 값이 나오지 않았고, 그 이후 이 사실을 알게되어 수정하였다. 실제 블록체인 코인 채굴을 맛본 느낌이라 좋은 경험이라고 생각했다.