7조

Unix Project

19011449 윤태강
19011547 김상원
19011582 강주현

Contents

1

Create Data

create.c

분리된 입력 데이터 생성하기

void debug_file(void)

void child_proc(int id)

int create_source_data()

int create_source_data()

```c
int create_source_data() {

        /* create per-process, distrinuted input data
        create one time, if possible. After creating,

        printf("**Distribute input data across process

        int     i;
        int     pid;

        i = 0;
        for (i = 1; i <= 4; i++)
        {
                pid = fork();
                if (pid == -1)
                {
                        perror("fork fail");
                        exit(1);
                }
                else if (pid == 0)
                        child_proc(i);
        }
        i = 0;
        int     status;
        while (i < 4)
        {
                pid = waitpid(-1, &status, 0);
                if (pid == -1)
                {
                        perror("pid error");
                        exit(1);
                }
                i++;
        }
        debug_file();
        return (0);
}
```

fork() 후 자식은 child_proc() 실행

하위 프로세스의 종료를 기다림

void child_proc(int id)

```c
void    child_proc(int id)
{
        char    file_name[1024];
        int             fd;
        int             data[MB];
        int             ret;

        sprintf(file_name, "data/p%d.dat", id);
        fd = open(file_name, O_CREAT | O_WRONLY | O_TRUNC, 0644);
        if (fd == -1)
        {
                perror("fd fail");
                exit(1);
        }
        for (int i = 0; i < MB; i++)
                data[i] = 4 * i + id;
        if ((ret = write(fd, data, MB * sizeof(int))) < 0)
        {
                perror("write fail");
                exit(1);
        }
        if (ret != MB * 4)
        {
                printf("[Error] Error");
                exit(1);
        }
        exit(0);
}
```

데이터 생성

2

1 CreateData

2 Client_Oriented

using **Pipe**

3 Server_Oriented

4 Conclusion

Unix Programming

Client_Oriented

int client_oriented_io()

int client_oriented_io()

클라이언트 중심의 I/O 작업을 수행하고 각 작업 단계에 소요된 시간을 측정하기

```
        unlink("clientOrientedTime"); // clientOriented
        fd = open("clientOrientedTime", O_CREAT | O_WRO
        ZERO = 0;
        write(fd, &ZERO, sizeof(int));
        write(fd, &ZERO, sizeof(int));
        write(fd, &ZERO, sizeof(int));
        close(fd);
        parallel_operation();
```

파일을 만들고 ZERO로 초기화,
parallel_operation() 실행

parallel_operation() 에서 병렬 통신 정렬 등이 이루어짐, 시간 측정

```
#ifdef TIMES
        fd = open("clientOrientedTime", O_RDONLY);
        int     clientTime[4];
        int     i;

        i = 0;
        while ((read(fd, &clientTime[i], sizeof(int)) > 0))
        {
                if (i == COMM)
                        printf("communicate TIMES: %ld\n", (long)clientTime[i]);
                else if (i == COMP)
                        printf("compute TIMES: %ld\n", (long)clientTime[i]);
                else if (i == IO)
                        printf("IO TIMES: %ld\n", (long)clientTime[i]);
                i++;
        }
        close(fd);
#endif
```

이후 clientOrientedTime에 적힌 시간을 읽어서 출력

Unix Programming

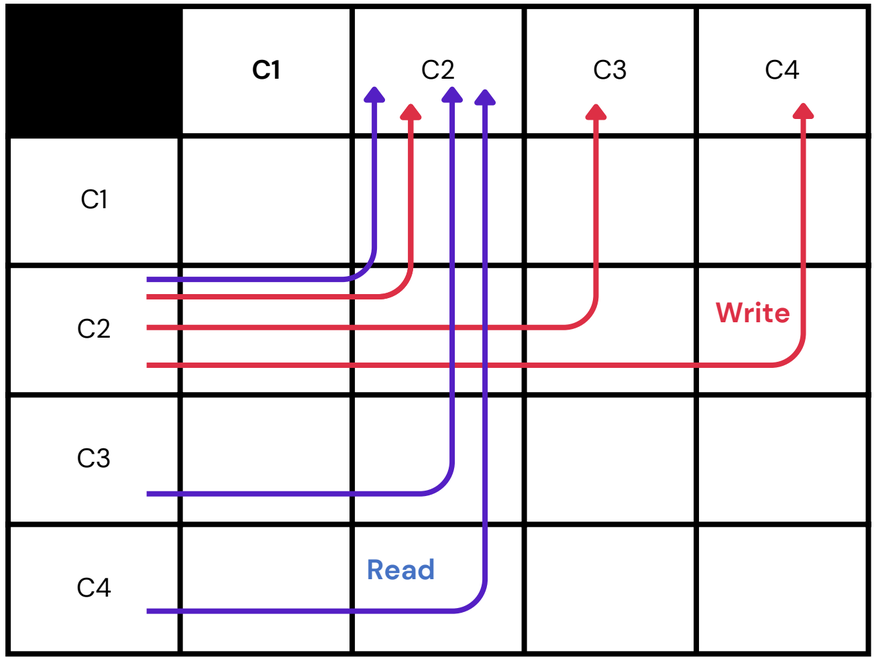
Client_Oriented

Client.c

client 끼리 통신할 파이프를 열어줌
fork() 후, 자식 프로세스는
Client2Server() 실행

void parallel_operation(void)

```
void    parallel_operation(void)
{
        int     client2client[4][4][2];
        int     i;
        int     j;

        for (i = 0; i < 4; i++)
        {
                for (j = 0; j < 4; j++)
                        pipe(client2client[i][j]);
        }
        for (i = 0; i < 4; i++)
        {
                int     pid;

                pid = fork();
                if (pid == 0)
                {
                        Client2Server(i, client2client);
                        exit(0);
                }
        }
        parent("parallel_operation");
        debug_result();
}
```



void Client2Server(int, int)

```
void    Client2Server(int i, int client2client[4][4][2])
{
        int     pid;
        int     pip[2];
        int     status;

        pipe(pip);
        pid = fork();
        if (pid == 0) // client
                do_comm_node(i, client2client, pip);
        else // server
        {
                do_io_node(i, pip);
                wait(&status);
                printf("[DEBUG] Client2Server, pid : %d, status: %d done\n", pid, status);
        }
}
```

자식은 client가 되어 comm_node() 실행
부모는 io_node 실행

Unix Programming

Client_Oriented

void do_comm_node(int id, int client2client[4][4][2], int pip[2])

Client.c

void do_comm_node(int id, int client2client[4][4][2], int pip[2])

comm_time 측정

먼저 comm_init을 이용해 초기화

void comm_init(..)

```
void    comm_init(int id, int client2client[4][4][2], int pip[2], int data[MB])
{
        char            file_name[1024];
        int                     fd;
        int                     i;

        sprintf(file_name, "data/p%d.dat", id + 1);
        fd = open(file_name, O_RDONLY);
        if (fd < 0)
                exit(1);
        read(fd, data, MB * sizeof(int));
        close(pip[0]);
        for (i = 0; i < 4; i++)
                close(client2client[id][i][0]); // id -> i :  close(read)
        for (i = 0; i < 4; i++)
                close(client2client[i][id][1]); // i -> id : close(write)
        // client2client O_NONBLOCK
        for (i = 0; i < 4; i++)
        {
                if (i == id)
                        continue;
                fcntl(client2client[i][id][0], F_SETFL, O_NONBLOCK);
        }
}
```

id에 따른 remain 값 이용, 데이터 선별해 client간 통신

```
for (i = 0; i < MB; i++)
{
        int remain = data[i] % 32;

        if ((8 * id + 1 <= remain && remain <= 8 * id + 8) || (id == 3 && remain == 0))
                dump[++dump_idx] = data[i];
        else if (1 <= remain && remain <= 8)
                ret = write(client2client[id][0][1], &data[i], sizeof(int));
        else if (9 <= remain && remain <= 16)
                ret = write(client2client[id][1][1], &data[i], sizeof(int));
        else if (17 <= remain && remain <= 24)
                ret = write(client2client[id][2][1], &data[i], sizeof(int));
        else if ((25 <= remain && remain < 32) || remain == 0)
                ret = write(client2client[id][3][1], &data[i], sizeof(int));
        else
        {
                char    buffer[1024];

                sprintf(buffer, "[Error : %d] %d", id, remain % 512);
                perror(buffer);
                exit(1);
        }
        int     j;

        for (j = 0; j < 4; j++)
        {
                int     buffer;

                if (id == j)
                        continue;
                // read쪽 pipe buffer 계속 비워줘야 한다.
                ret = read(client2client[j][id][0], &buffer, sizeof(int));
                if (ret > 0)
                        dump[++dump_idx] = buffer;
        }
}
```

Write

pipe는 크기가 정해져 있는데, 파이프의 크기보다 많은 것을 적으면 -> block, nonblock이 돼도 데이터 손실

write해주면 반대편에서 읽어줘야 한다.

Read

Client_Oriented

void do_comm_node(int id, int client2client[4][4][2], int pip[2])

void do_comm_node(...)

comm_time 측정

이어서

```
        // NON-BLOCKING -> busy-waiting 발생
        // BLOCKING -> DEADLOCK 발생
        while (1)
        {
                for (i = 0; i < 4; i++)
                {
                        int     buffer;

                        if (id == i)
                                continue;
                        ret = read(client2client[i][id][0], &buffer, sizeof(int));
                        if (ret > 0)
                                dump[++dump_idx] = buffer;
                }
                // dump_idx == MB - 1의 의미는 dump[MB - 1]에 write를 완료
                if (dump_idx == MB - 1)
                        break ;
        }

#ifdef TIMES
        gettimeofday(&etime, NULL);
        time_result = (etime.tv_usec - stime.tv_usec);
        writeTimeAdvLock(COMM, time_result);
#endif

        do_compute_node(dump); // 정렬하기
        send_server(pip, dump); // server에게 전달
        for (i = 0; i < 4; i++)
                close(client2client[id][i][1]); // id -> i :  close(write)
        for (i = 0; i < 4; i++)
                close(client2client[i][id][0]); // i -> id : close(read)
        close(pip[1]);
}
```

dump_idx == MB-1 일때 까지 데이터 수신

WriteLock을 걸어 동시에 적지 못하게 함

정렬은 C 내장함수 qsort() 이용

server에게 보냄

void send_server()

```
void    send_server(int pip[2], int dump[MB])
{
        int     i;

        i = 0;
        while (i < MB)
        {
                write(pip[1], &dump[i], sizeof(int) * 8);
                i += 8;
        }
}
```

클라이언트와 서버 사이의 파이프 이용

Unix Programming

Client_Oriented

void do_io_node(…)

io_time 측정

void do_io_node(…)

```c
void    do_io_node(int id, int pip[2])
{
        int             ret;
        int             chunk[8];
        char    file_name[25];
        int             fd;

        close(pip[1]);
        sprintf(file_name, "IOnode/IOnode_#%d", id + 1);
        printf("file_name : [%s]\n", file_name);
        fd = open(file_name, O_CREAT | O_WRONLY | O_TRUNC, 0644);
        if (fd < 0)
        {
                perror("I/O node CREATE FAIL");
                exit(1);
        }
#ifdef TIMES
        int time_result;
        struct timeval stime, etime;

        gettimeofday(&stime, NULL);
#endif
        while ((ret = read(pip[0], chunk, sizeof(int) * 8)) > 0)
                write(fd, chunk, sizeof(int) * 8);
#ifdef TIMES
        gettimeofday(&etime, NULL);
        time_result = (etime.tv_usec - stime.tv_usec);
        writeTimeAdvLock(IO, time_result);
#endif
        close(pip[0]);
        close(fd);
}
```

chunk가 준비되면 바로 io_node 파일에 작성

클라이언트와 서버 사이 파이프에서 chunk 단위로 읽기

void writeTimeAdvLock(..)

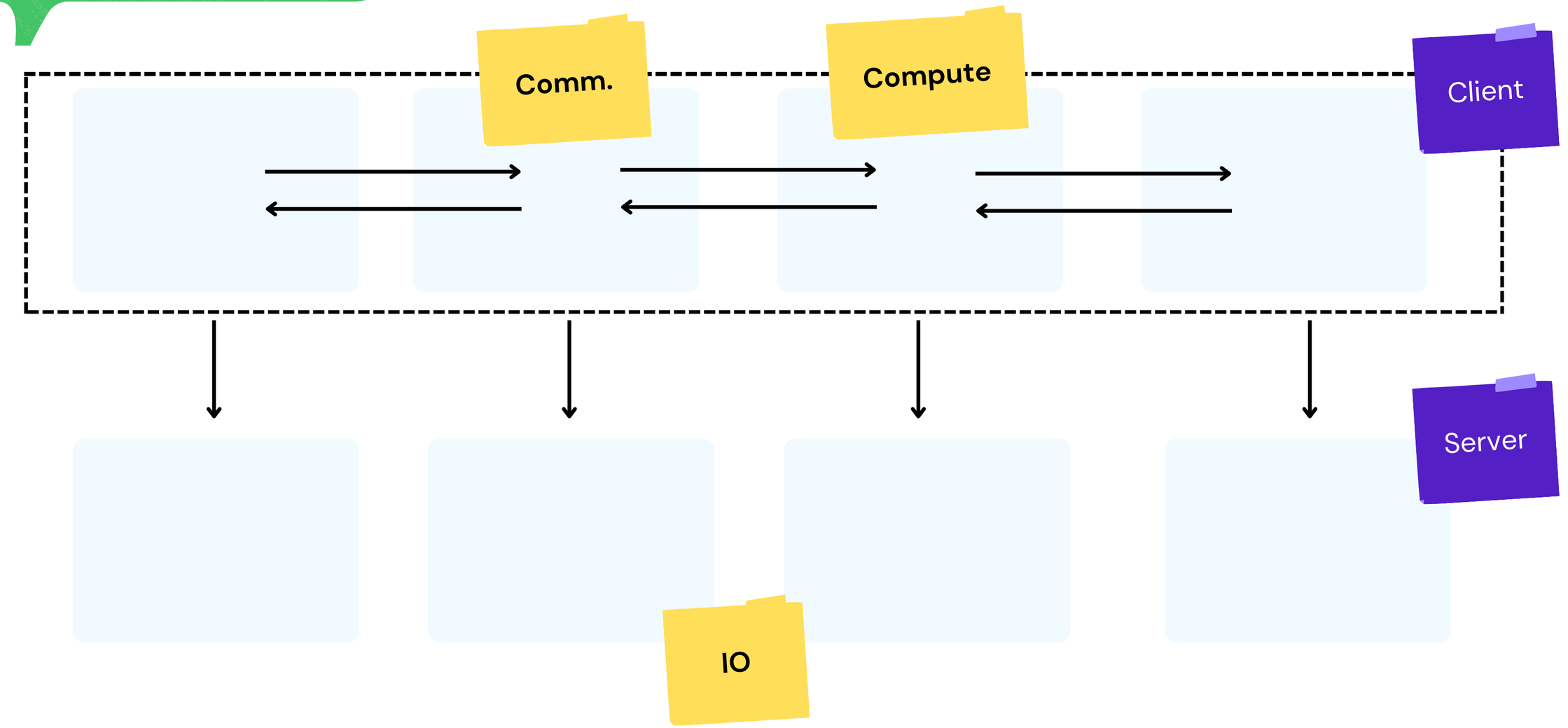
```c
void    writeTimeAdvLock(int index, int time_result)
{
        struct flock    myLock;
        int                             fd;
        int                             buffer;

        if (time_result < 0)
                time_result += SEC;
        fd = open("clientOrientedTime", O_RDWR, 0644);
        myLock.l_type = F_WRLCK;
        myLock.l_whence = SEEK_SET;
        myLock.l_start = index * sizeof(int);
        myLock.l_len = sizeof(int);
        fcntl(fd, F_SETLKW, &myLock); // F_SETLKW로 쓰기 lock
        lseek(fd, index * sizeof(int), SEEK_SET); // index로 위치 이동
        read(fd, &buffer, sizeof(int)); // 읽기
        time_result += buffer;
        // printf("[DEBUG] (index, %d) = %d\n", index, time_result);
        lseek(fd, index * sizeof(int), SEEK_SET);
        write(fd, &time_result, sizeof(int)); // 쓰기
        myLock.l_type = F_UNLCK; // F_SETLKW 해제
        fcntl(fd, F_SETLKW, &myLock);
        close(fd);
}
```

critical section 에 동시 접근 방지

다른 프로세스가 적고 있을 때는 다른 프로세스가 적지 못하게 한다.

Unix Programming

Client_Oriented



3

1 CreateData

2 Client_Oriented

3 Server_Oriented

using **FIFO**

4 Conclusion

Unix Programming

Server_Oriented

server.c

int server_oriented_io()

```
unlink("serverOrientedTime"); // serverOrientedTime은 server_orie
fd = open("serverOrientedTime", O_CREAT | O_WRONLY, 0644);
ZERO = 0;
write(fd, &ZERO, sizeof(int));
write(fd, &ZERO, sizeof(int));
write(fd, &ZERO, sizeof(int));
close(fd);
parallel_operation();
```

Client와 비슷하게 serverOrientedTime 파일 생성 후 초기화

void parallel_operation();

```
static void     parallel_operation(void)
{
        int     i;
        int     j;
        char    client2server[1024];
        char    cmd[1024];

        for (i = 0; i < 4; i++)
        {
                for (j = 0; j < 4; j++)
                {
                        int     ret;

                        sprintf(client2server, "tmp/serverfifo%d2%d", i, j);
                        ret = mkfifo(client2server, 0644);
                }
        }
        for (i = 0; i < 4; i++)
        {
                int     pid;

                pid = fork();
                if (pid == 0)
                {
                        ft_Client2Server(i);
                        exit(0);
                }
        }
        parent("parallel_operation");
        debug_result();
}
```

Unix Programming

Server_Oriented

server.c

void ft_Client2Server(int i)

```c
int     pid;
int     status;

pid = fork();
if (pid == 0) // client
        send_server(i);
else // server
{
        do_comm_node(i);
        pid = wait(&status);
        printf("[DEBUG] Client2Server, pid : %d, status: %d done\n", pid, status);
}
```

FIFO를 각각 데이터 read, write로 open해줌

comm_init()

```c
static void     comm_init(int id, int server2server[4][2])
{
        int                     i;
        char            fifo_name[64];

        for (i = 0; i < 4; i++)
        {
                if (id == i)
                        continue;
                sprintf(fifo_name, "tmp/serverfifo%d2%d", i, id);
                server2server[i][0] = open(fifo_name, O_RDONLY | O_NONBLOCK); // i -> id: read
        }
        for (i = 0; i < 4; i++) {
                if (id == i)
                        continue;
                sprintf(fifo_name, "tmp/serverfifo%d2%d", id, i);
                server2server[i][1] = open(fifo_name, O_WRONLY); // id -> i: write
        }
}
```

FIFO는 항상 Read와 Write가 동시에 Open되어야함.
아니면 Block이 되는데, O_NonBlock을 이용한 다음
바로 뒤에서 Write해주기

Server_Oriented

server.c

void send_server()

client의 데이터를 적절한 server에게 fifo를 통해 전달한다.

```
}
i = 0;
while (i < MB)
{
        int     remain;

        remain = data[i] % 32;
        if (1 <= remain && remain <= 8)
                ret = write(client2server[0], &data[i], sizeof(int)); // id -> 0; write
        else if (9 <= remain && remain <= 16)
                ret = write(client2server[1], &data[i], sizeof(int)); // id -> 1; write
        else if (17 <= remain && remain <= 24)
                ret = write(client2server[2], &data[i], sizeof(int)); // id -> 2; write
        else if ((25 <= remain && remain < 32) || remain == 0)
                ret = write(client2server[3], &data[i], sizeof(int)); // id -> 3; write
        else
        {
                perror("wrong chunk number");
                exit(1);
        }
        i++;
}
for (i = 0; i < 4; i++)
        close(client2server[i]);
close(fd);
}
```

Server_Oriented

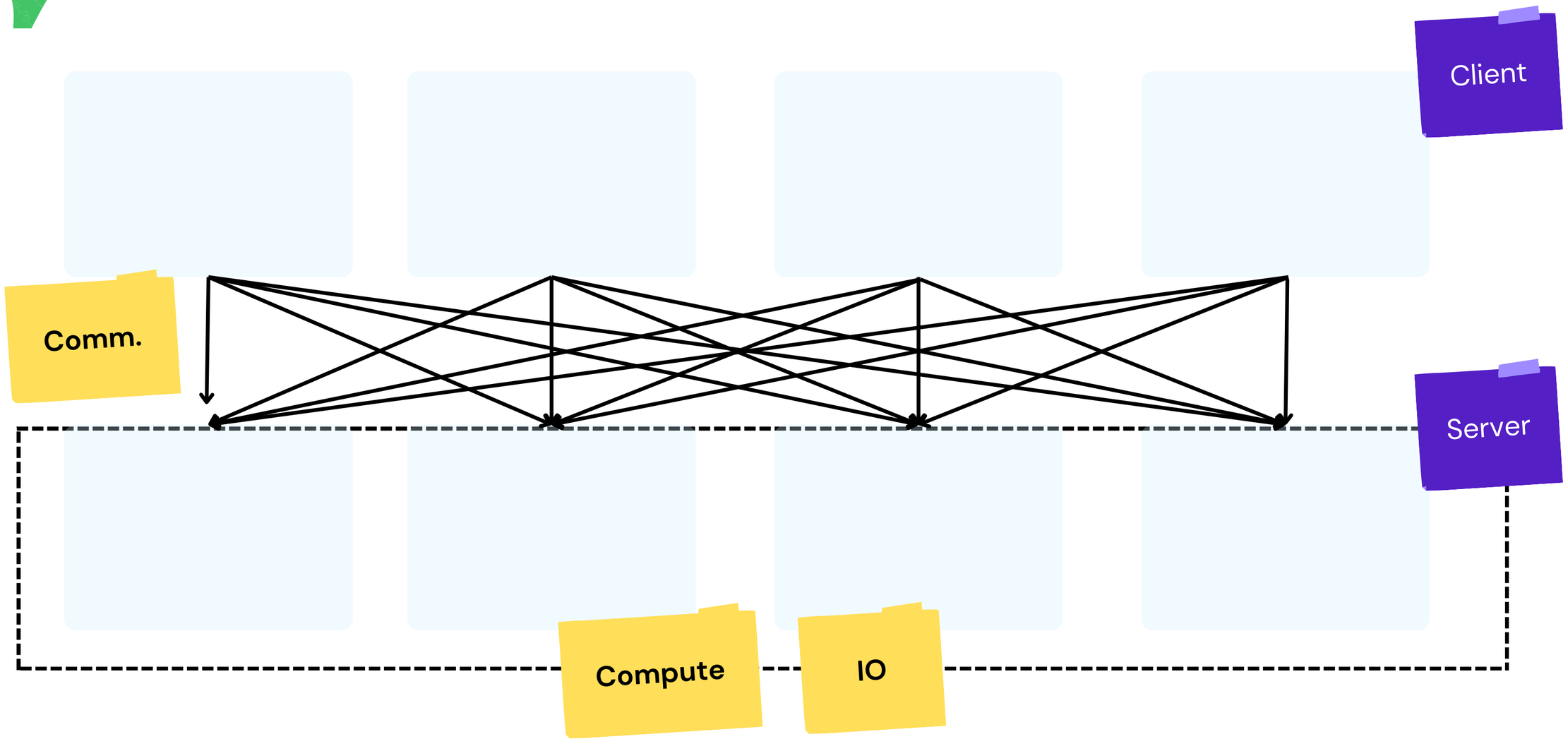
server.c

do_comm_node()

```
comm_init(id, client2server);

dump_idx = 0;
while (dump_idx < MB)
{
        for (i = 0; i < 4; i++)
        {
                ret = read(client2server[i], chunk, sizeof(int) * 8); // NON_BLOCK, chunk 단위로 읽으려고 시도
                // ret은 바이트 단위이고, dump_idx는 sizeof(int) 단위이다.
                if (ret > 0)
                {
                        memcpy(&dump[dump_idx], chunk, ret); // ret만큼만 적는다.
                        dump_idx += (ret / sizeof(int)); // write에서 sizeof(int) 단위로 찍기 때문에 sizeof(int)의 배수
                }
        }
}
```

4개의 client 프로세스에게 전달받은 데이터를 모두 읽는다.

Server_Oriented



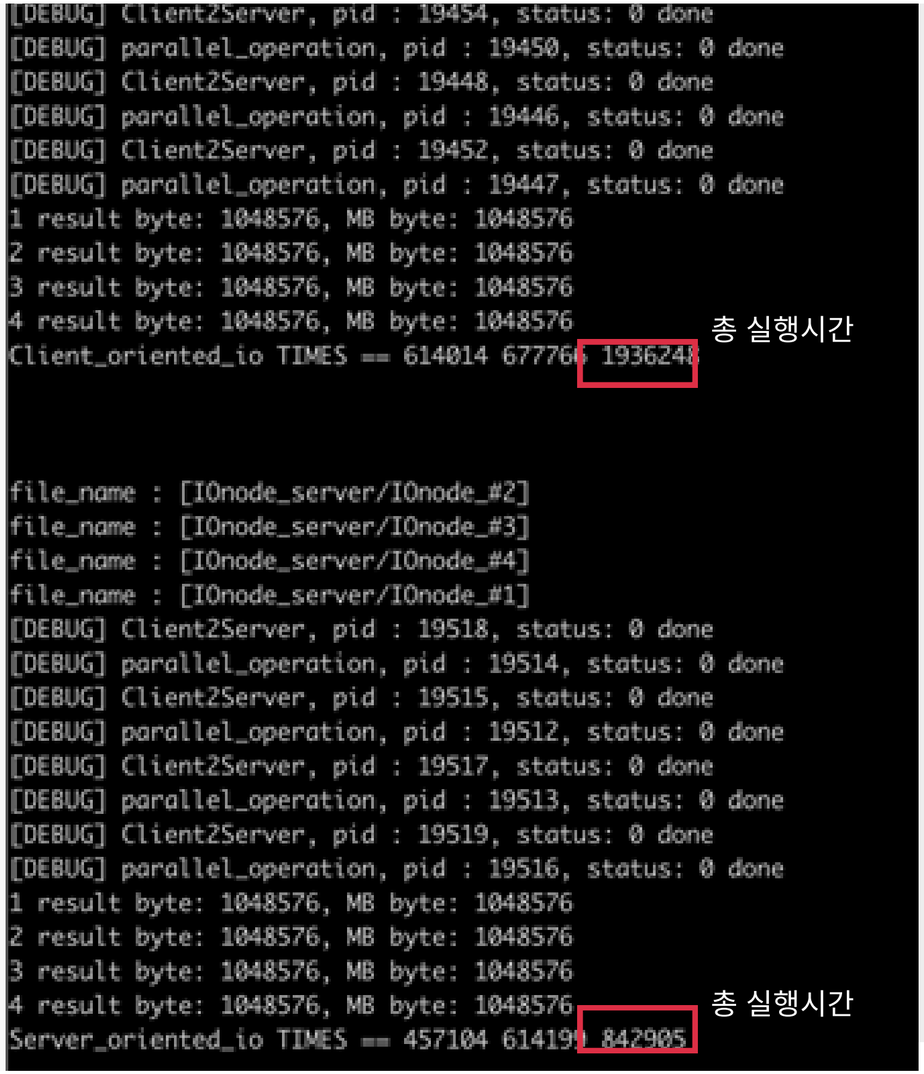
4

Conclusion

	Client_Oriented	Server_Oriented
Comm.	0.953	2.42
Comp.	0.112	0.119
IO	2.96	0.07

(sec)

Conclusion



Client oriented는 교환을 하고 1:1로 server로 보내는데, server_oriented는 server로 곧바로 보내기 때문에(예를 들어 client1이 자신의 데이터를 server1, 2, 3, 4로 곧바로 보냄)

즉, 교환과 전달이 동시에 한다. 따라서 server가 더 빠르다

Conclusion

```c
static void	debug_result4client(void)
{
	int		i;
	int		fd;
	int		io_size;
	int		compute_size;
	int		buffer[MB];
	char	cmd[1024];

	for (i = 1; i <= 4; i++)
	{
		sprintf(cmd, "IOnode_client/IOnode_#%d", i);
		fd = open(cmd, O_RDONLY);
		io_size = (int)lseek(fd, 0, SEEK_END);
		close(fd);
		sprintf(cmd, "client_data/p%d.dat", i);
		fd = open(cmd, O_RDONLY);
		compute_size = (int)lseek(fd, 0, SEEK_END);
		close(fd);
		printf("%d = compute byte: %d, io byte: %d, MB byte: %d\n", i, compute_size, io_size, (int)(MB * sizeo
	}
}
```

```c
static void	debug_result(void)
{
	int		i;
	char	cmd[1024];
	int		fd;
	int		buffer[MB];
	int		io_size;
	int		compute_size;

	for (i = 1; i <= 4; i++)
	{
		sprintf(cmd, "IOnode_server/IOnode_#%d", i);
		fd = open(cmd, O_RDONLY);
		io_size = (int)lseek(fd, 0, SEEK_END);
		close(fd);
		sprintf(cmd, "server_data/p%d.dat", i);
		fd = open(cmd, O_RDONLY);
		compute_size = (int)lseek(fd, 0, SEEK_END);
		close(fd);
		printf("%d = compute byte: %d, io byte: %d, MB byte: %d\n", i, compute_size, io_size,  (int)(MB * sizeof(int
	}
}
```

Compute와 IO node 크기를 확인해주는 *debug_ressult()*

각각 **1048576 Byte로 동일(1MB)**

데이터의 총 합은 4MB(1MB Integer)

```
[DEBUG] Client2Server, pid : 31483, status: 0 done
[DEBUG] parallel_operation, pid : 31480, status: 0 done
1 = compute byte: 1048576, io byte: 1048576, MB byte: 1048576
2 = compute byte: 1048576, io byte: 1048576, MB byte: 1048576
3 = compute byte: 1048576, io byte: 1048576, MB byte: 1048576
4 = compute byte: 1048576, io byte: 1048576, MB byte: 1048576
Client_oriented_io TIMES == 199976 367214 1832762



file name : [IOnode_server/IOnode_#3]
```

Conclusion

```
writeTimeAdvLock(IO, time_result);

char    cmd[1024];

sprintf(cmd, "od -i IOnode_server/IOnode_#%d > dump/server_result%d.dump", id + 1, id + 1);
system(cmd);
close(fd);
```

```
        int             ret;
        char    cmd[1024];

        sprintf(cmd, "od -i server_data/p%d.dat > dump/server_data%d.dump", id + 1, id + 1);
        system(cmd);
        sprintf(file_name, "server_data/p%d.dat", id + 1);
        fd = open(file_name, O_RDONLY);
```

Server에서의 IO node dump,
Compute node dump

Client 에서의 IO node dump,
Compute node dump

```
        writeTimeAdvLock4client(IO, time_result);
#endif
        char    cmd[1024];

        sprintf(cmd, "od -i IOnode_client/IOnode_#%d > dump/client_result%d.dump", id + 1, id + 1);
        system(cmd);
        close(pip[0]);
        close(fd);
```

```
        int                     i;

        sprintf(cmd, "od -i client_data/p%d.dat > dump/client_data%d.dump", id + 1, id + 1);
        system(cmd);
        sprintf(file_name, "client_data/p%d.dat", id + 1);
```

이후 diff 명령어의 결과로 출력값이 없다
=> server의 결과값과 client의 결과값이 동일하다.

7조

감사합니다