

# Responsi Praktikum Sistem Komputer dan Jaringan

Kosmas Rio Legowo

23/512012/PA/21863

Departemen Ilmu Komputer dan Elektronika

Universitas Gadjah Mada

kosmasriolegowo@mail.ugm.ac.id

## Soal-Soal

### Soal 1: Memory Mapping (mmap)

Buatlah program C yang menggunakan mmap untuk memodifikasi isi file bernama hello.txt. Awalnya, file tersebut berisi string "Hello, world!". Program Anda harus mengubah karakter pertama file tersebut sehingga string menjadi "Jello, world!". Pastikan program Anda menangani pembukaan file, memory mapping, dan pemberian dengan benar.

Berikut hasil melengkapi skeleton code yang sudah diberikan:

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>

int main() {
    const char *filename = "hello.txt";
    int fd;
    struct stat file_stat;
    char *mapped_file;
    size_t file_size;

    // Langkah 1: Buka file untuk membaca dan menulis
    fd = open(filename, O_RDWR);
    if (fd == -1) {
        perror("Error membuka file");
        return 1;
    }

    // Langkah 2: Dapatkan ukuran file
    if (fstat(fd, &file_stat) == -1) {
```

```

        perror("Error mendapatkan ukuran file");
        close(fd);
        return 1;
    }
    file_size = file_stat.st_size;

    // Langkah 3: Memory-map file
    mapped_file = mmap(NULL, file_size, PROT_READ |
        PROT_WRITE, MAP_SHARED, fd, 0);
    if (mapped_file == MAP_FAILED) {
        perror("Error mapping file");
        close(fd);
        return 1;
    }

    // Langkah 4: Modifikasi isi file
    mapped_file[0] = 'J'; // Ubah karakter pertama dari
        'H' menjadi 'J'

    // Langkah 5: Sinkronisasi perubahan dan pembersihan
    if (msync(mapped_file, file_size, MS_SYNC) == -1) {
        perror("Error mensinkronisasi perubahan");
    }
    if (munmap(mapped_file, file_size) == -1) {
        perror("Error unmapping file");
    }
    close(fd);

    return 0;
}

```

Penjelasan tentang code di atas:

1. Pada langkah 1 saya menambahkan kode berikut

```

    fd = open(filename, O_RDWR);
    if (fd == -1) {
        perror("Error membuka file");
        return 1;
    }

```

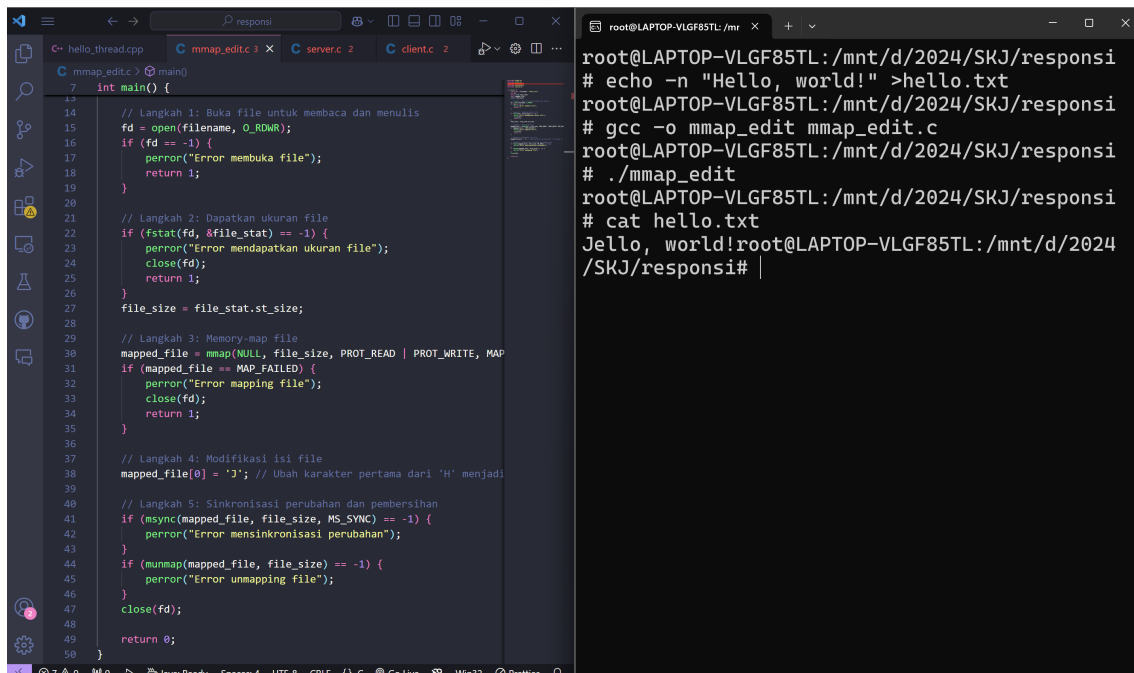
Fungsi open untuk membuka file menggunakan open dengan flag O\_RDWR untuk mode baca dan tulis. Ditambahkan juga pengecekan error jika file gagal dibuka.

2. Pada langkah 2 ditambahkan `file_size = file_stat.st_size;`. Informasi tentang file (termasuk ukuran) diperoleh menggunakan `fstat`, yang

menyimpan data ke dalam struktur `file_stat`. Ukuran file ditentukan oleh anggota `st_size` dari struktur `file_stat`. Nilai ini disimpan dalam variabel `file_size` untuk digunakan dalam operasi berikutnya.

3. Pada langkah 3 file dipetakan ke memori menggunakan fungsi `mmap`. Pemetaan dilakukan dengan mode yang memungkinkan file dibaca dan dimodifikasi langsung di memori (`PROT_READ | PROT_WRITE`) serta sinkron dengan file aslinya (`MAP_SHARED`). Jika pemetaan gagal, program menutup file dan keluar setelah menampilkan pesan kesalahan.
4. Pada langkah 4 ditambahkan `mapped_file[0] = 'J';`  
Modifikasi dilakukan langsung pada area memori yang dipetakan (`mapped_file`). Karakter pertama diubah dari 'H' menjadi 'J' dengan mengakses indeks 0 dari array `mapped_file`.
5. Pada langkah 5 program menyinkronkan perubahan ke file fisik di disk menggunakan fungsi `msync`. Hal ini memastikan bahwa perubahan yang dibuat di memori diterapkan pada file yang sebenarnya. Jika sinkronisasi gagal, program menampilkan pesan kesalahan, tetapi tetap melanjutkan untuk membersihkan memori. Langkah terakhir adalah membatalkan pemetaan dengan `munmap` untuk membebaskan memori yang telah dialokasikan, lalu menutup file descriptor menggunakan `close`.

Berikut hasil pengujian program tersebut untuk file `hello.txt` yang berisi "Hello, world!". Dapat dilihat pada hasil tangkapan layar di bawah ini bahwa karakter pertama dalam file `hello.txt` sudah berubah sehingga menjadi "Jello, world!"



```
C:\hello_thread.cpp C:\mmap_edit.c 3 C:\server.c 2 C:\client.c 2
7 int main() {
14 // Langkah 1: Buka file untuk membaca dan menulis
15 fd = open(filename, O_RDWR);
16 if (fd == -1) {
17     perror("Error membuka file");
18     return 1;
19 }
21 // Langkah 2: Dapatkan ukuran file
22 if (fstat(fd, &file_stat) == -1) {
23     perror("Error mendapatkan ukuran file");
24     close(fd);
25     return 1;
26 }
27 file_size = file_stat.st_size;
28
29 // Langkah 3: Memory-map file
30 mapped_file = mmap(NULL, file_size, PROT_READ | PROT_WRITE, MAP
31 if (mapped_file == MAP_FAILED) {
32     perror("Error mapping file");
33     close(fd);
34     return 1;
35 }
36
37 // Langkah 4: Modifikasi isi file
38 mapped_file[0] = 'J'; // Ubah karakter pertama dari 'H' menjadi
39
40 // Langkah 5: Sinkronisasi perubahan dan pembersihan
41 if (msync(mapped_file, file_size, MS_SYNC) == -1) {
42     perror("Error mensinkronisasi perubahan");
43 }
44 if (munmap(mapped_file, file_size) == -1) {
45     perror("Error unmapping file");
46 }
47 close(fd);
48
49 return 0;
50 }
```

```
root@LAPTOP-VLGF85TL:/mnt # echo -n "Hello, world!" >hello.txt
root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi # gcc -o mmap_edit mmap_edit.c
root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi # ./mmap_edit
root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi # cat hello.txt
Jello, world!root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi#
```

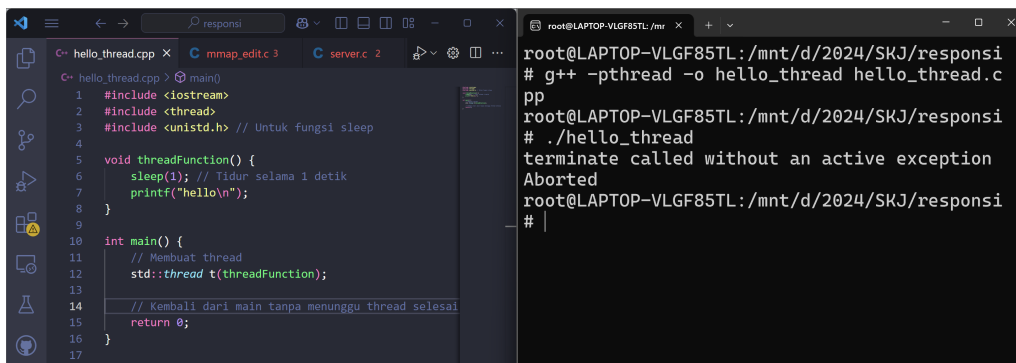
## Soal 2: Debugging Threads

### Tugas:

1. Jelaskan mengapa program ini tidak mencetak "hello" ketika dijalankan.

#### Jawab:

Program ini menggunakan thread untuk menjalankan fungsi threadFunction. Namun, di baris 15 (`return 0;`), fungsi main selesai tanpa menunggu thread selesai. Ketika fungsi main selesai, proses utama (yang mencakup semua thread) akan dihentikan oleh sistem operasi, sehingga thread tidak sempat menyelesaikan pekerjaannya, termasuk mencetak "hello". Pada tangkapan layar di bawah ini terlihat hasil ketika program tersebut dijalankan, program tersebut tidak mencetak "hello".



```
C++ hello_thread.cpp X C mmapi_edit.c 3 C server.c 2
1 #include <iostream>
2 #include <thread>
3 #include <unistd.h> // Untuk fungsi sleep
4
5 void threadFunction() {
6     sleep(1); // Tidur selama 1 detik
7     printf("hello\n");
8 }
9
10 int main() {
11     // Membuat thread
12     std::thread t(threadFunction);
13
14     // Kembali dari main tanpa menunggu thread selesai
15     return 0;
16 }
17

root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi
# g++ -pthread -o hello_thread hello_thread.c
pp
root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi
# ./hello_thread
terminate called without an active exception
Aborted
root@LAPTOP-VLGF85TL:/mnt/d/2024/SKJ/responsi
#
```

2. Modifikasi program agar mencetak "hello" sebelum program selesai. Tulis kode yang sudah diperbaiki.

#### Jawab:

Kita perlu memastikan bahwa thread utama menunggu thread t selesai sebelum keluar dari fungsi main. Hal ini dapat dilakukan dengan memanggil metode `join` pada objek thread `t`. Metode ini memblokir eksekusi thread utama sampai thread `t` selesai menjalankan fungsinya. Dengan demikian, program akan menunggu thread selesai mencetak "hello" sebelum keluar dari fungsi main.

Berikut adalah hasil kode yang sudah diperbaiki dan hasil ketika program tersebut dijalankan:

```
#include <iostream>
#include <thread>
#include <unistd.h> // Untuk fungsi sleep

void threadFunction() {
    sleep(1); // Tidur selama 1 detik
    printf("hello\n");
}

int main() {
    // Membuat thread
```

```

        std::thread t(threadFunction);

        t.join();
        return 0;
    }

```

The screenshot shows a code editor on the left with the following C++ code in `hello_thread.cpp`:

```

1 #include <iostream>
2 #include <thread>
3 #include <unistd.h> // Untuk fungsi sleep
4
5 void threadFunction() {
6     sleep(1); // Tidur selama 1 detik
7     printf("hello\n");
8 }
9
10 int main() {
11     // Membuat thread
12     std::thread t(threadFunction);
13
14     t.join();
15     return 0;
16 }
17

```

On the right, a terminal window shows the execution steps:

```

root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# g++ -pthread -o hello_thread hello_thread.c
pp
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# ./hello_thread
hello
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
#

```

### Soal 3: Pemrograman Jaringan (Socket Programming)

Tulis program C untuk membuat aplikasi client-server sederhana.

Server harus:

1. Bind ke port 8080.
2. Menunggu koneksi dari client.
3. Ketika client terhubung, server harus mengirim pesan "Hello, Client!" ke client dan menutup koneksi.

Client harus:

1. Terhubung ke server yang berjalan di localhost pada port 8080.
2. Menerima pesan dari server.
3. Mencetak pesan yang diterima ke konsol.

Berikut hasil melengkapi skeleton code yang diberikan untuk `server.c`:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {

```

```

int server_fd, client_fd;
struct sockaddr_in server_addr, client_addr;
socklen_t client_len = sizeof(client_addr);
char *message = "Hello, Client!";

// Langkah 1: Membuat socket
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd == -1) {
    perror("Error membuat socket");
    return 1;
}

// Langkah 2: Bind socket ke port 8080
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(8080);

if (bind(server_fd, (struct sockaddr*)&server_addr,
        sizeof(server_addr)) == -1) {
    perror("Error bind socket");
    close(server_fd);
    return 1;
}

// Langkah 3: Listen untuk koneksi masuk
if (listen(server_fd, 5) == -1) {
    perror("Error listen koneksi");
    close(server_fd);
    return 1;
}

printf("Server mendengarkan di port 8080...\n");

// Langkah 4: Terima koneksi
client_fd = accept(server_fd, (struct sockaddr*)&
        client_addr, &client_len);
if (client_fd == -1) {
    perror("Error menerima koneksi");
    close(server_fd);
    return 1;
}

// Langkah 5: Kirim pesan ke client
write(client_fd, message, strlen(message));

// Langkah 6: Tutup koneksi
close(client_fd);

```

```
    close(server_fd);  
  
    return 0;  
}
```

Penjelasan tentang code di atas:

1. Pada langkah 1 saya menambahkan kode berikut:

```
server_fd = socket(AF_INET, SOCK_STREAM, 0);  
if (server_fd == -1) {  
    perror("Error membuat socket");  
    return 1;  
}
```

Bagian ini menggunakan fungsi `socket()` untuk membuat endpoint komunikasi. Fungsi ini menerima tiga parameter: domain (`AF_INET` untuk IPv4), tipe socket (`SOCK_STREAM` untuk TCP), dan protokol (diatur ke 0 untuk protokol default). Jika `socket` berhasil dibuat, ia mengembalikan file descriptor (integer) yang dapat digunakan untuk operasi jaringan selanjutnya. Jika gagal, `socket()` mengembalikan nilai negatif, yang ditangani dengan mencetak pesan kesalahan menggunakan `perror()` dan keluar dari program.

2. Pada langkah 2 ditambahkan

```
if (connect(client_fd, (struct sockaddr*)&  
server_addr, sizeof(server_addr)) == -1) {  
    perror("Error koneksi ke server");  
    close(client_fd);  
    return 1;  
}
```

Fungsi `bind()` menghubungkan socket yang dibuat ke alamat IP dan port tertentu. Fungsi ini menerima tiga parameter, `server_fd` yaitu file descriptor dari socket yang dibuat, kemudian ada `server_addr`, struktur yang berisi alamat IP dan port (sudah diisi sebelumnya), dan yang terakhir `sizeof(server_addr)` dari ukuran struktur alamat. Setelah itu dilakukan pengecekan kesalahan, jika `bind` gagal (misalnya, port sudah digunakan), program akan mencetak pesan kesalahan dan menutup socket.

3. Pada langkah 3 ditambahkan

```
if (listen(server_fd, 5) == -1) {  
    perror("Error listen koneksi");  
    close(server_fd);  
    return 1;  
}
```

Fungsi `listen()` mengubah socket server sebagai "passive socket" yang siap menerima koneksi masuk. Fungsi ini menerima dua parameter, `server_fd` yaitu file descriptor dari socket, dan 5 sebagai jumlah koneksi maksimum yang bisa masuk ke antrian sebelum ditolak. Setelah itu dilakukan pengecekan kesalahan, jika `listen` gagal, program akan mencetak pesan kesalahan, menutup socket, dan keluar.

4. Pada langkah 4 ditambahkan

```
client_fd = accept(server_fd, (struct sockaddr*)&
    client_addr, &client_len);
if (client_fd == -1) {
    perror("Error menerima koneksi");
    close(server_fd);
    return 1;
}
```

Fungsi `accept()` digunakan untuk menerima koneksi dari client yang mencoba terhubung ke server. Fungsi ini menerima tiga parameter, `server_fd` yaitu file descriptor dari socket, kemudian `client_addr` struktur untuk menyimpan alamat client, dan `client_len` yang adalah pointer ke ukuran struktur alamat client. Fungsi ini akan mengembalikan file descriptor baru untuk koneksi client yang berhasil. Setelah itu dilakukan pengecekan kesalahan, jika `accept` gagal, program akan mencetak pesan kesalahan dan mengakhiri program.

5. Pada langkah 5 dipakai fungsi `write` mengirimkan data ke client melalui deskriptor file `client_fd`. Pesan yang dikirim adalah "Hello, Client!".

6. Pada langkah 6 program menutup koneksi client dan server.

Berikut hasil melengkapi skeleton code yang diberikan untuk `client.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int client_fd;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};

    // Langkah 1: Membuat socket
    client_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (client_fd == -1) {
        perror("Error membuat socket");
    }
}
```



```

        return 1;
    }

    // Langkah 2: Tentukan alamat server
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1")
        ;

    // Langkah 3: Terhubung ke server
    if (connect(client_fd, (struct sockaddr*)&
        server_addr, sizeof(server_addr)) == -1) {
        perror("Error koneksi ke server");
        close(client_fd);
        return 1;
    }

    // Langkah 4: Terima pesan dari server
    read(client_fd, buffer, sizeof(buffer));
    printf("Diterima dari server: %s\n", buffer);

    // Langkah 5: Tutup koneksi
    close(client_fd);

    return 0;
}

```

Penjelasan tentang code di atas:

1. Pada langkah 1 saya menambahkan kode berikut:

```

client_fd = socket(AF_INET, SOCK_STREAM, 0);
if (client_fd == -1) {
    perror("Error membuat socket");
    return 1;
}

```

Untuk client, fungsi `socket()` digunakan dengan parameter serupa seperti pada server. Socket ini bertindak sebagai endpoint komunikasi client, yang akan dihubungkan ke server. Jika pembuatan socket gagal, program mencetak pesan kesalahan dengan `perror()` dan keluar.

2. Pada langkah 2 menentukan alamat server, di mana `server_addr.sin_family = AF_INET`; menentukan bahwa socket menggunakan protokol IPv4. Kemudian `server_addr.sin_port = htons(8080)`; menentukan port, di mana 8080 adalah port server dan `htons` mengkonversi port ke format byte network byte order (big-endian), sesuai standar jaringan.

3. Pada langkah 3 ditambahkan

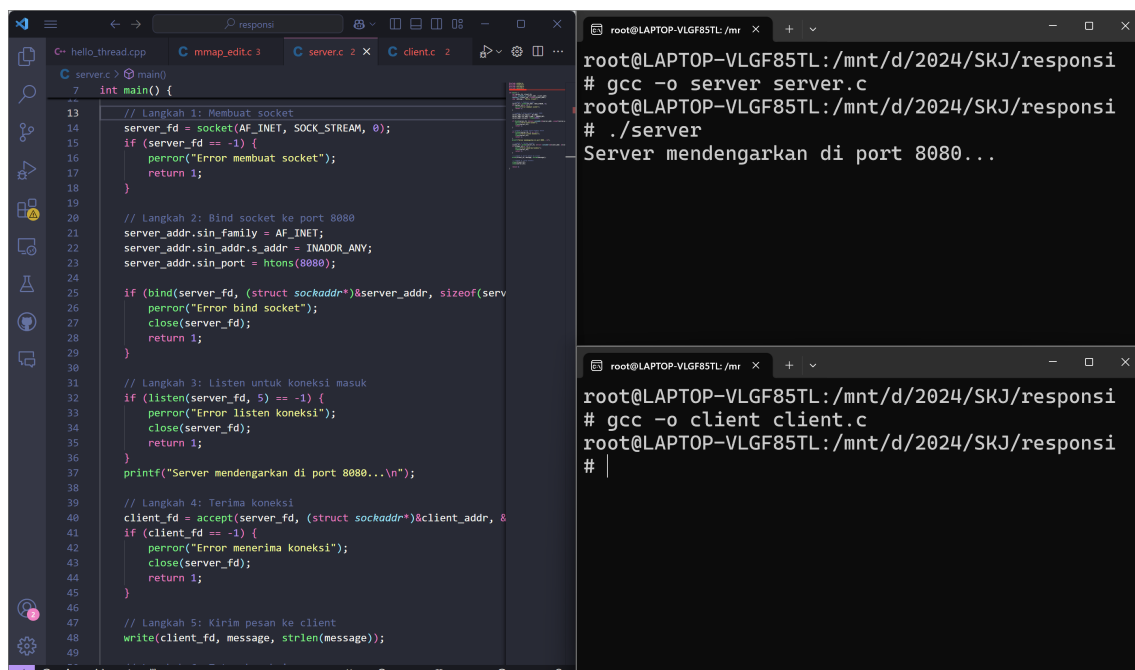
```
if (connect(client_fd, (struct sockaddr*)&
server_addr, sizeof(server_addr)) == -1) {
    perror("Error koneksi ke server");
    close(client_fd);
    return 1;
}
```

Fungsi `connect()` digunakan oleh client untuk menghubungkan socket yang dibuat ke server. Parameter pertama adalah file descriptor socket client, sedangkan parameter kedua adalah struktur `server_addr` yang berisi alamat IP dan port server. Port ditentukan dengan `htons()`, dan alamat IP diatur ke 127.0.0.1 untuk koneksi lokal. Jika koneksi gagal (misalnya, server tidak berjalan atau alamat salah), program mencetak pesan kesalahan dan menutup socket.

4. Pada langkah 4 setelah koneksi berhasil, fungsi `read()` digunakan untuk membaca data yang dikirim server. Data yang diterima disimpan dalam buffer, kemudian dicetak di konsol. Fungsi ini memblokir eksekusi sampai data diterima. Dengan membaca pesan ini, client dapat mengetahui bahwa koneksi berhasil dan menerima informasi dari server.

5. Pada langkah 5 program menutup koneksi client.

Berikut hasil menjalankan program `server.c` dan `client.c`:



The screenshot shows a code editor on the left and a terminal on the right. The code editor displays the source code for `server.c` and `client.c`. The terminal shows the execution of these programs on a system with IP 127.0.0.1.

```
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# gcc -o server server.c
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# ./server
Server mendengarkan di port 8080...
```

```
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# gcc -o client client.c
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
#
```

```

C- hello_thread.cpp  C mmap_edit.c  C server.c  C client.c  x
C client.c > main()
1 #include <stdio.h>
2
3
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 int main() {
8     int client_fd;
9     struct sockaddr_in server_addr;
10    char buffer[1024] = {0};
11
12    // Langkah 1: Membuat socket
13    client_fd = socket(AF_INET, SOCK_STREAM, 0);
14    if (client_fd == -1) {
15        perror("Error membuat socket");
16        return 1;
17    }
18
19    // Langkah 2: Tentukan alamat server
20    server_addr.sin_family = AF_INET;
21    server_addr.sin_port = htons(8080);
22    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
23
24    // Langkah 3: Terhubung ke server
25    if (connect(client_fd, (struct sockaddr*)&server_addr, sizeof(s
26        perror("Error koneksi ke server");
27        close(client_fd);
28        return 1;
29    }
30
31    // Langkah 4: Terima pesan dari server
32    read(client_fd, buffer, sizeof(buffer));
33    printf("Diterima dari server: %s\n", buffer);
34
35    // Langkah 5: Tutup koneksi
36    close(client_fd);
37
38    return 0;
39 }
40
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# gcc -o server server.c
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# ./server
Server mendengarkan di port 8080...
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# |

root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# gcc -o client client.c
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# ./client
Diterima dari server: Hello, Client!
root@LAPTOP-VLGF85TL: /mnt/d/2024/SKJ/responsi
# |

```