**实验15**

1. **服务器端配置与运行**
2. 编译服务器代码

文本

AI 生成的内容可能不正确。

1. 运行服务器

文本

AI 生成的内容可能不正确。

1. **单进程客户端实现**
2. 创建并编写单进程客户端文件single\_thread\_time.c

文本

AI 生成的内容可能不正确。

1. 编译单进程客户端

文本

AI 生成的内容可能不正确。

1. **多进程客户端实现**
2. 创建并编写多进程客户端文件multi\_thread\_time.c

文本

AI 生成的内容可能不正确。

1. 编译多进程客户端

文本

AI 生成的内容可能不正确。

1. **运行测试与结果分析**
2. 保持服务器运行，运行单线程程客户端程序

文本

AI 生成的内容可能不正确。

1. 保持服务器运行，运行多线程客户端

图形用户界面, 文本

AI 生成的内容可能不正确。

**[附录]**

源代码

1. 单进程客户端（single\_client.c）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <time.h>

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 8888

#define FILE\_SIZE 19

void print\_time(const char\* message) {

    struct timespec ts;

    clock\_gettime(CLOCK\_MONOTONIC, &ts);

    printf("[%ld.%09ld] %s\n", ts.tv\_sec, ts.tv\_nsec, message);

}

void download\_file(int sock, int offset, int length, char\* buffer) {

    char request[2] = {offset, length};

    send(sock, request, sizeof(request), 0);

    for (int i = 0; i < length; i++) {

        recv(sock, &buffer[offset + i], 1, 0);

    }

}

int main() {

    struct timespec start, end;

    clock\_gettime(CLOCK\_MONOTONIC, &start);

    print\_time("Program started");

    int sock;

    struct sockaddr\_in serv\_addr;

    char buffer[FILE\_SIZE + 1] = {0};

    print\_time("Creating socket...");

    if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

        perror("Socket creation error");

        return -1;

    }

    serv\_addr.sin\_family = AF\_INET;

    serv\_addr.sin\_port = htons(SERVER\_PORT);

    if (inet\_pton(AF\_INET, SERVER\_IP, &serv\_addr.sin\_addr) <= 0) {

        perror("Invalid address/ Address not supported");

        return -1;

    }

    print\_time("Connecting to server...");

    if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

        perror("Connection Failed");

        return -1;

    }

    print\_time("Starting single-thread download...");

    download\_file(sock, 0, FILE\_SIZE, buffer);

    clock\_gettime(CLOCK\_MONOTONIC, &end);

    double elapsed = (end.tv\_sec - start.tv\_sec) +

                    (end.tv\_nsec - start.tv\_nsec) / 1000000000.0;

    printf("Downloaded content: %s\n", buffer);

    printf("Total download time: %.9f seconds\n", elapsed);

    close(sock);

    print\_time("Connection closed");

    return 0;

}

1. 多进程客户端（multi\_process\_client.c）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <pthread.h>

#include <time.h>

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 8888

#define FILE\_SIZE 19

#define THREAD\_COUNT 4

typedef struct {

    int sock;

    int offset;

    int length;

    char\* buffer;

} ThreadArgs;

void print\_time(const char\* message) {

    struct timespec ts;

    clock\_gettime(CLOCK\_MONOTONIC, &ts);

    printf("[%ld.%09ld] %s\n", ts.tv\_sec, ts.tv\_nsec, message);

}

void\* download\_segment(void\* args) {

    ThreadArgs\* targs = (ThreadArgs\*)args;

    char request[2] = {targs->offset, targs->length};

    send(targs->sock, request, sizeof(request), 0);

    for (int i = 0; i < targs->length; i++) {

        recv(targs->sock, &targs->buffer[targs->offset + i], 1, 0);

    }

    return NULL;

}

int main() {

    struct timespec start, end;

    clock\_gettime(CLOCK\_MONOTONIC, &start);

    print\_time("Program started");

    int sock[THREAD\_COUNT];

    struct sockaddr\_in serv\_addr;

    char buffer[FILE\_SIZE + 1] = {0};

    pthread\_t threads[THREAD\_COUNT];

    ThreadArgs args[THREAD\_COUNT];

    int segment\_size = FILE\_SIZE / THREAD\_COUNT;

    int remainder = FILE\_SIZE % THREAD\_COUNT;

    print\_time("Starting multi-thread (4 threads) download...");

    for (int i = 0; i < THREAD\_COUNT; i++) {

        if ((sock[i] = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

            perror("Socket creation error");

            return -1;

        }

        serv\_addr.sin\_family = AF\_INET;

        serv\_addr.sin\_port = htons(SERVER\_PORT);

        if (inet\_pton(AF\_INET, SERVER\_IP, &serv\_addr.sin\_addr) <= 0) {

            perror("Invalid address/ Address not supported");

            return -1;

        }

        print\_time("Connecting to server...");

        if (connect(sock[i], (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

            perror("Connection Failed");

            return -1;

        }

        args[i].sock = sock[i];

        args[i].offset = i \* segment\_size;

        args[i].length = segment\_size + (i == THREAD\_COUNT - 1 ? remainder : 0);

        args[i].buffer = buffer;

        print\_time("Creating download thread...");

        pthread\_create(&threads[i], NULL, download\_segment, &args[i]);

    }

    for (int i = 0; i < THREAD\_COUNT; i++) {

        pthread\_join(threads[i], NULL);

        close(sock[i]);

    }

    clock\_gettime(CLOCK\_MONOTONIC, &end);

    double elapsed = (end.tv\_sec - start.tv\_sec) +

                    (end.tv\_nsec - start.tv\_nsec) / 1000000000.0;

    printf("Downloaded content: %s\n", buffer);

    printf("Total download time: %.9f seconds\n", elapsed);

    print\_time("All connections closed");

    return 0;

}