並行Serverプログラム作成

C言語プログラム

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <signal.h>
#include <unistd.h>
#include <pthread.h>


void sig_child(int signo){
    int pid, status;
    pid = wait(&status);
    printf("PID: %d terminated\n", pid);
    printf("Status: %d\n", status);
}


static void *do_thread(void *connfd){
    if (pthread_detach(pthread_self()) != 0){
        perror("pthread_detach");
        exit(1);
    }
    char inbuf[1024], obuf[1024];
    memset(&inbuf, 0, sizeof(inbuf));
    if (recv((int)connfd, inbuf, sizeof(inbuf), 0) == -1){
        perror("recv");
        exit(1);
    }
    printf("%s", inbuf);

    memset(&obuf, 0, sizeof(obuf));
    snprintf(obuf, sizeof(obuf),
             "HTTP/1.0 200 OK\r\n"
             "Content-Type: text/html\r\n"
             "\r\n"
             "<font color=red><h1>HELLO</h1></font>\r\n"
    );
    if (send((int)connfd, obuf, (int) strlen(obuf), 0) == -1){
        perror("send");
        exit(1);
    }
    close((int)connfd);
```

```
    pthread_exit((void *)0);
}


int main(int argc, char *argv[]) {
    char *mode = argv[1];
    if (!mode){
        fprintf(stderr, "Usage: ./server [MODE]\n");
        exit(1);
    }

    int sock0, sock;
    struct sockaddr_in addr;
    struct sockaddr_in client;
    socklen_t len;

    if ((sock0 = socket(AF_INET, SOCK_STREAM, 0)) == -1){
        perror("socket");
        exit(1);
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(31600);
    addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sock0, (struct sockaddr *)&addr, sizeof(addr)) == -1){
        perror("bind");
        exit(1);
    }

    if (listen(sock0, 5) == -1){
        perror("listen");
        exit(1);
    }

    len = sizeof(client);
    if (strcmp(mode, "f") == 0){
        pid_t pid;
        char inbuf[1024], obuf[1024];
        while (1){
            if ((sock = accept(sock0, (struct sockaddr *)&client, &len)) == -1){
                perror("accept");
                exit(1);
            }

            if ((pid = fork()) == -1){
                perror("fork");
                exit(1);
            }

            if (pid == 0){
                close(sock0);
                memset(&inbuf, 0, sizeof(inbuf));
                if (recv(sock, &inbuf, sizeof(inbuf), 0) == -1){
                    perror("recv");
                    exit(1);
```

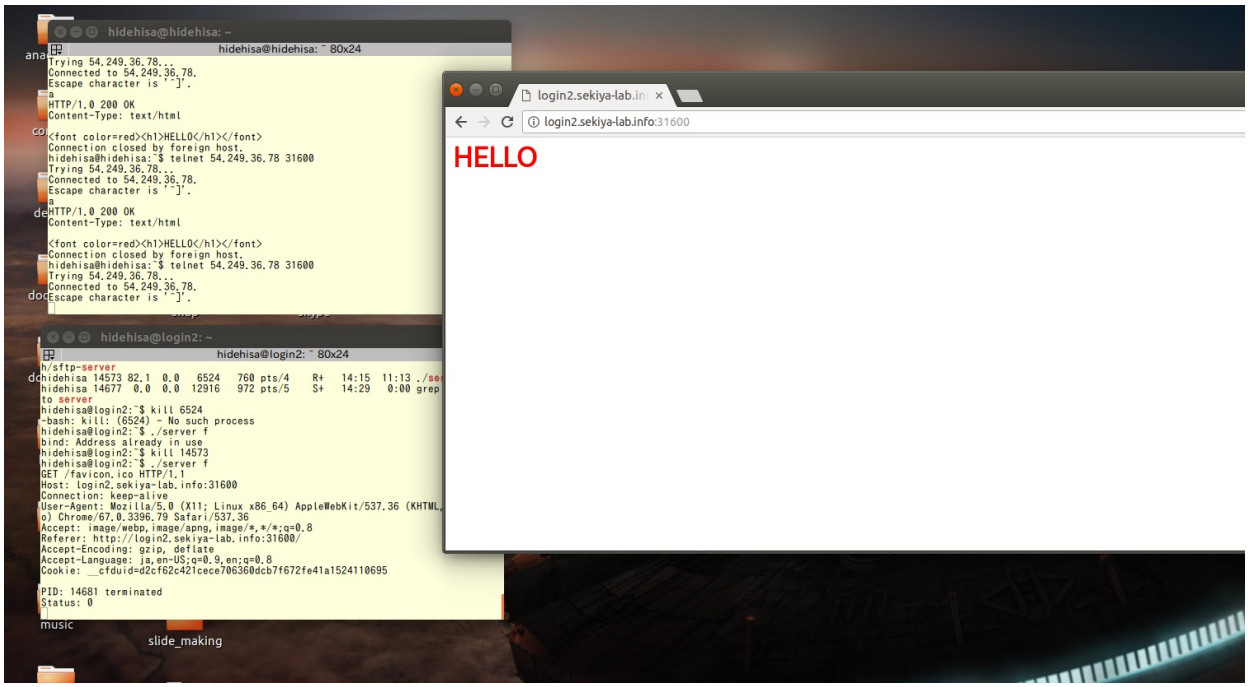
```
                }
                printf("%s", inbuf);

                memset(&obuf, 0, sizeof(obuf));
                snprintf(obuf, sizeof(obuf),
                        "HTTP/1.0 200 OK\r\n"
                        "Content-Type: text/html\r\n"
                        "\r\n"
                        "<font color=red><h1>HELLO</h1></font>\r\n");
                if (send(sock, obuf, (int)strlen(obuf), 0) == -1){
                    perror("send");
                    exit(1);
                }
                close(sock);
                exit(0);
            } else {
                signal(SIGCHLD, sig_child);
                close(sock);
            }
        }
    } else if (strcmp(mode, "t") == 0){
        while (1){
            pthread_t tid;

            if ((sock = accept(sock0, (struct sockaddr *)&client, &len)) == -1){
                perror("accept");
                exit(1);
            }

            if (pthread_create(&tid, NULL, &do_thread, (void *)sock) != 0){
                perror("pthread_create");
                exit(1);
            }
        }
    } else {
        int i, idx;
        int client_arr[FD_SETSIZE];
        int nready;
        int maxfd = FD_SETSIZE;

        ssize_t n;

        char inbuf[1024], obuf[1024];

        fd_set allset, readset;

        struct timeval waitval;
        waitval.tv_sec = 2;
        waitval.tv_usec = 500;

        for (i = 0; i < maxfd; i++) {
            client_arr[i] = -1;
        }

        FD_ZERO(&allset);
        FD_SET(sock0, &allset);
```

```
        while (1){
            memcpy(&readset, &allset, FD_SETSIZE);
            if ((nready = select(maxfd+1, &readset, NULL, NULL, &waitval)) == -1){
                perror("select");
                exit(1);
            }

            if (nready > 0){
                if (FD_ISSET(sock0, &readset)){
                    if ((sock = accept(sock0, (struct sockaddr *)&client, &len)) ==
                        perror("accept");
                        exit(1);
                    }
                    FD_CLR(sock0, &readset);
                    idx = 0;
                    while (idx < FD_SETSIZE && client_arr[idx] != -1) idx++;

                    if (idx != FD_SETSIZE){
                        client_arr[idx] = sock;
                        FD_SET(sock, &readset);
                    } else {
                        fprintf(stderr, "Queue is full\n");
                    }
                } else {
                    fprintf(stderr, "Accept error\n");
                }
            }
            for (i = 0; i < maxfd; i++) {
                if (client_arr[i] != -1 && FD_ISSET(client_arr[i], &readset)){
                    memset(&inbuf, 0, sizeof(inbuf));
                    if ((n = read(client_arr[i], inbuf, sizeof(inbuf))) == -1){
                        perror("read");
                        exit(1);
                    }
                    printf("%s", inbuf);

                    memset(&obuf, 0, sizeof(obuf));
                    snprintf(obuf, sizeof(obuf),
                             "HTTP/1.0 200 OK\r\n"
                             "Content-Type: text/html\r\n"
                             "\r\n"
                             "<font color=red><h1>HELLO</h1></font>\r\n"
                    );
                    if (send(client_arr[i], obuf, (int) strlen(obuf), 0) == -1){
                        perror("send");
                        exit(1);
                    }
                    close(client_arr[i]);
                    FD_CLR(client_arr[i], &readset);
                    client_arr[i] = -1;
                }
            }
        }

    }
```

```
}
```

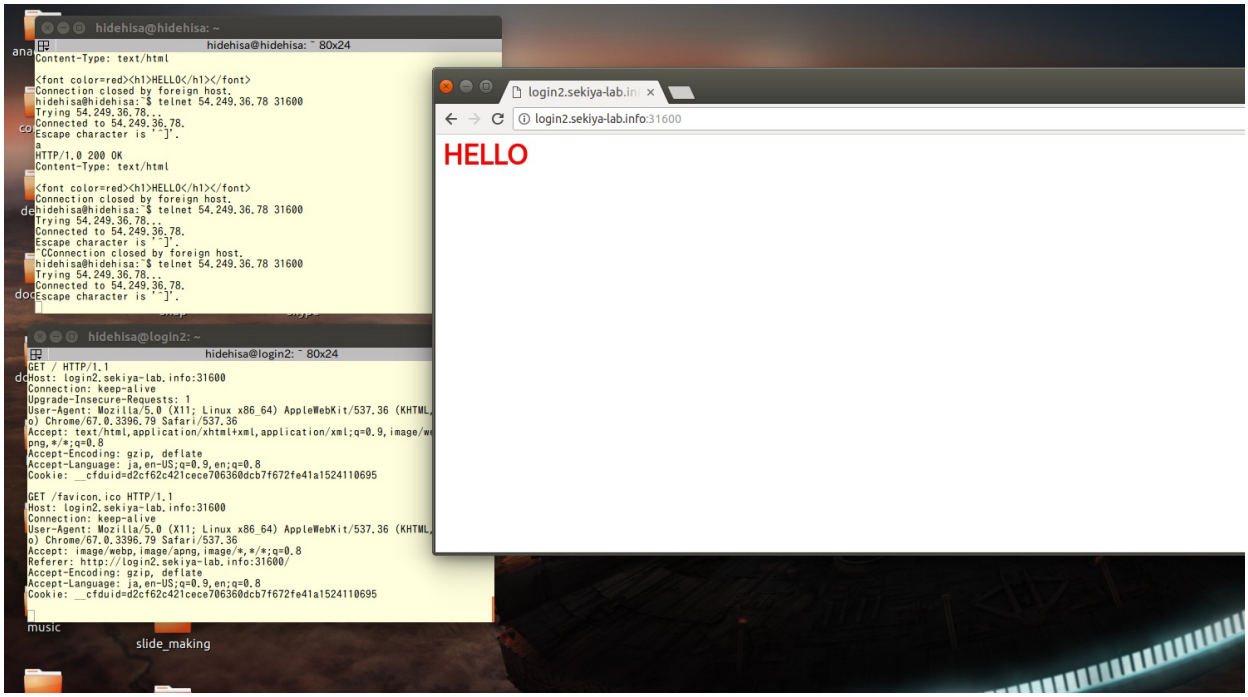
結果

- fork()を用いた場合



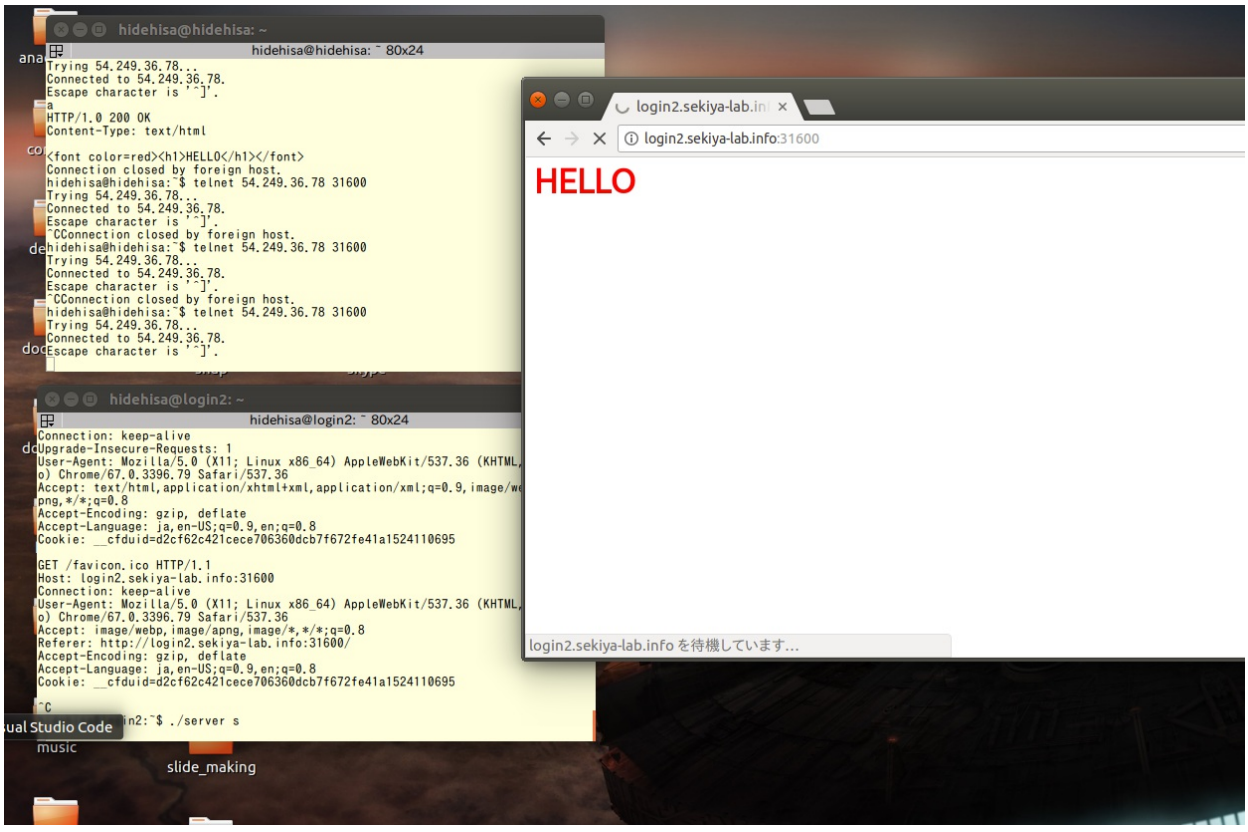
telnetを用いてデータの読み込み待ちにした状態でも、ブラウザからのリクエストに応えることができたため、並行サーバとして構成できていることがわかる。

- pthread_create()を用いた場合



fork()と同様にしてtelnetで読み込み街にした状態でも、ブラウザからのリクエストには応えることができた。

- select()を用いた場合



この場合は、telnetを用いて読み込み待ちの状態にした際にブラウザからのリクエストは読み込み待ちになってしまった。この原因として考えられるのは、FD_ISSET()で読み込み準備が完了したsocketがあ

る場合にはread()するようにしているが、read()先でtelnetから何も送信されてないため、待ち続けた状態になってしまい、また他のソケットの読み込み処理が前のソケットの読み込み処理が終わらない限り始まらないような書き方になってしまっているからであると考えられる。