Linux Programming Lab Programs

Sai Hemanth Bheemreddy
CVR College of Engineering

2020-2021

Contents

1 Implement ‘cp’ and ‘mv’ shell commands using file related system calls.

copy.c

```c
// Copies contents of <source file> into <destination file>

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if(argc != 3) {
        printf("Invalid arguments.\nUsage: copy <source file> <destination file>\n");
        return -1;
    }

    ssize_t retIn, retOut;
    char buffer[BUF_SIZE];

    int srcID = open(argv[1], O_RDONLY);
    if(srcID == -1) {
        printf("Unable to open %s\n", argv[1]);
        return -1;
    }

    int destID = open(argv[2], O_WRONLY | O_CREAT, 0644);
    if(destID == -1) {
        printf("Unable to open %s\n", argv[2]);
        return -1;
    }

    while((retIn = read(srcID, &buffer, BUF_SIZE)) > 0) {
        retOut = write(destID, &buffer, retIn);
        if(retOut != retIn) {
            printf("Unable to copy files\n");
            return -1;
        }
    }

    close(srcID);
    close(destID);

    printf("Successfully copied %s to %s\n", argv[1], argv[2]);
    return 0;
}
```

Output

```
$ ./bin/copy source.txt dest.txt
Successfully copied source.txt to dest.txt
```

move.c

```c
// Moves <source file> into <destination file>

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
```

```c
#include <unistd.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if(argc != 3) {
        printf("Invalid arguments.\nUsage: move <source file> <destination file>\n");
        return -1;
    }

    if((link(argv[1], argv[2])) == -1) {
        printf("Unable to move file: %s\n", argv[1]);
        return -1;
    }
    if((unlink(argv[1])) == -1) {
        printf("Unable to move file: %s\n", argv[1]);
        unlink(argv[2]);
        return -1;
    }

    printf("Successfully moved %s to %s\n", argv[1], argv[2]);
    return 0;
}
```

Output

```
$ ./bin/move source.txt src.txt
Successfully moved source.txt to src.txt
```

2 Create a new file with 0666 access permissions and enable the close-on-exec flag.

close_on_exec.c

```c
// Creates a file named "close-on-evec.txt" with permissions 666 and close-on-evec flag set.

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

int main(void) {
    int fd;
    if((fd = open("close-on-exec.txt", O_WRONLY|O_CREAT, 0666)) == -1) {
        printf("Unable to open the file.");
        return -1;
    }

    int old_flags = fcntl(fd, F_GETFD);
    printf("Old flags: %d\n", old_flags);

    fcntl(fd, F_SETFD, FD_CLOEXEC);

    int new_flags = fcntl(fd, F_GETFD);
    printf("New flags: %d\n", new_flags);

    return  0;
}
```

Output

```
$ ls
bin close_on_exec.c

$ ./bin/close_on_exec
Old flags: 0
New flags: 1

$ ls
close-on-exec.txt bin close_on_exec.c
```

3 Change the file control information while setting O_SYNC flag

# Not in Syllabus

4 Write a C Program to implement a UNIX ‘ls -ls file1’ command using File related API & stat structure.

ls_helper.h

```
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>

typedef struct {
    long int block_no;
    char type;
    char *perms;
    long int nlink;
    char *uname;
    char *gname;
    long int size;
    char *time_str;
    char *name;
} file_info;

int getFileInfo(file_info*, char*);
int getFileInfoAt(file_info*, char*, int);

```

```c
int getFileInfoAt(file_info *info, char* pathname, int dirfd) {
    struct stat st;
    struct passwd *pwd;
    struct group *grp;

    char type;
    if(fstatat(dirfd, pathname, &st, 0) < 0) {
        return -1;
    }

    char *time_str = (char*) calloc(20, sizeof(char));
    char *perms = (char*) calloc(10, sizeof(char));

    pwd = getpwuid(st.st_uid);
    grp = getgrgid(st.st_gid);

    long int block_no = st.st_blocks / 2;
    strftime(time_str, 20, "%b %d %H:%M", localtime(&st.st_mtime));

    if(S_ISREG(st.st_mode))
        type = '-';
    else if(S_ISDIR(st.st_mode)) {
        type = 'd';
        block_no = 0;
    } else if(S_ISCHR(st.st_mode))
        type = 'c';
    else if(S_ISBLK(st.st_mode))
        type = 'b';
    else if(S_ISFIFO(st.st_mode))
        type = 'p';
    else if(S_ISLNK(st.st_mode))
        type = 'l';
    else if(S_ISSOCK(st.st_mode))
        type = 's';

    perms[0] = (S_IRUSR & st.st_mode) ? 'r' : '-';
    perms[1] = (S_IWUSR & st.st_mode) ? 'w' : '-';
    perms[2] = (S_IXUSR & st.st_mode) ? 'x' : '-';
    perms[3] = (S_IRGRP & st.st_mode) ? 'r' : '-';
    perms[4] = (S_IWGRP & st.st_mode) ? 'w' : '-';
    perms[5] = (S_IXGRP & st.st_mode) ? 'x' : '-';
    perms[6] = (S_IROTH & st.st_mode) ? 'r' : '-';
    perms[7] = (S_IWOTH & st.st_mode) ? 'w' : '-';
    perms[8] = (S_IXOTH & st.st_mode) ? 'x' : '-';
    perms[9] = '\0';

    info->block_no = block_no;
    info->type = type;
    info->perms = perms;
    info->nlink = st.st_nlink;
    info->uname = pwd->pw_name;
    info->gname = grp->gr_name;
    info->size = st.st_size;
    info->time_str = time_str;
    info->name = pathname;

    return 0;
}

int getFileInfo(file_info *info, char* pathname) {
    return getFileInfoAt(info, pathname, AT_FDCWD);
}
```

ls_file.c

```
// Displays file info like 'ls -ls'

#include <stdio.h>
#include <math.h>
#include "ls_helper.h"

#define MAX(a,b) (((a)>(b))?(a):(b))

int main(int argc, char *argv[]) {
    if(argc <= 1) {
        printf("Invalid arguments.\nUsage: ls_file <file> [<file> <file>]\n");
        return -1;
    }

    // Variables for formatting
    int p_blockno = 0;
    int p_nlink = 0;
    int p_size = 0;
    file_info infos[argc-1];

    for(int i = 1; i < argc; i++) {
        if(getFileInfo(&infos[i-1], argv[i]) < 0) {
            printf("Unable to retrive File Details");
            return -1;
        }
```

```c

        p_blockno = MAX((int)log10(infos[i-1].block_no)+1, p_blockno);
        p_nlink = MAX((int)log10(infos[i-1].nlink)+1, p_nlink);
        p_size = MAX((int)log10(infos[i-1].size)+1, p_size);
    }

    for(int i = 0; i < (argc-1); i++) {
        printf("%*ld %c%s %*ld %s %s %*ld %s %s\n", p_blockno, infos[i].block_no,
            infos[i].type, infos[i].perms, p_nlink, infos[i].nlink, infos[i].uname,
            infos[i].gname, p_size, infos[i].size, infos[i].time_str, infos[i].name);
    }
}
```

Output

```
$ ls -ls copy.c
4 -rw-r--r-- 1 ubuntu root 1032 Jan  3 18:29 copy.c

$ ./bin/ls_file copy.c
4 -rw-r--r-- 1 ubuntu root 1032 Jan 03 18:29 copy.c

$ ls -ls ls_file.c ls_dir.c
4 -rw-r--r-- 1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
4 -rw-r--r-- 1 ubuntu root 1140 Dec 22 14:20 ls_file.c

$ ./bin/ls_file ls_file.c ls_dir.c
4 -rw-r--r--  1 ubuntu root 1140 Dec 22 14:20 ls_file.c
4 -rw-r--r--  1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
```

5 Write a C Program to implement a UNIX ‘ls –ls dir1’ command using directory related system calls

Note: While implementing, you also need to implement “ls_helper.h” from Question. 4.

`ls_dir.c`

```c
// Displays directory info like 'ls -ls'

#include <stdio.h>
#include <math.h>
#include "ls_helper.h"

#define true (1)
#define false (0)
#define MAX_ENTRIES 1024
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

int main(int argc, char *argv[]) {
    if(argc != 2) {
        printf("Invalid arguments.\nUsage: ls_dir <dir> \n");
        exit(1);
    }

    // Variables for formatting
    int p_blockno = 0;
    int p_nlink = 0;
    int p_size = 0;

    // Variables
    int len = 0, total_blocks = 0;
    int dirfd = -1;
    DIR *dir = NULL;
    struct dirent *direntry;
    file_info infos[MAX_ENTRIES];

    if((dir = opendir(argv[1])) == NULL) {
        printf("Unable to open %s.\n", argv[1]);
        exit(1);
    }

    if((dirfd = open(argv[1], O_RDONLY)) < 0) {
        printf("Unable to open %s.\n", argv[1]);
        closedir(dir);
        exit(1);
    }

    while(true) {
        if((direntry = readdir(dir)) == NULL)
            break;

        if(direntry->d_name[0] == '.')
            continue;


        if(getFileInfoAt(&infos[len], direntry->d_name, dirfd) < 0) {
            printf("Unable to retrive File Details");
            return -1;
        }

        total_blocks += infos[len].block_no;
        p_blockno = MAX((int)log10(infos[len].block_no)+1, p_blockno);
        p_nlink = MAX((int)log10(infos[len].nlink)+1, p_nlink);
        p_size = MAX((int)log10(infos[len].size)+1, p_size);
        len++;
    }

    printf("total %d\n", total_blocks);
    for(int i = 0; i < len; i++) {
        printf("%*ld %c%s %*ld %s %s %*ld %s %s\n", p_blockno, infos[i].block_no,
            infos[i].type, infos[i].perms, p_nlink, infos[i].nlink, infos[i].uname,
            infos[i].gname, p_size, infos[i].size, infos[i].time_str, infos[i].name);
    }

    if(dirfd != -1) {
        close(dirfd);
    }
    if(dir != NULL) {
        closedir(dir);
    }

    return 0;
}
```

Output

```
$ ./bin/ls_dir .
total 80
0 drwxr-xr-x 25 ubuntu root  800 Jan 03 18:33 bin
4 -rw-r--r--  1 ubuntu root  574 Dec 22 16:05 close_on_exec.c
4 -rw-r--r--  1 ubuntu root 1032 Jan 03 18:29 copy.c
4 -rw-r--r--  1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
4 -rw-r--r--  1 ubuntu root 1140 Dec 22 14:20 ls_file.c
4 -rw-r--r--  1 ubuntu root 2258 Dec 22 14:37 ls_helper.h
4 -rw-r--r--  1 ubuntu root  666 Dec 22 16:05 move.c
4 -rw-r--r--  1 ubuntu root   53 Dec 22 06:20 source.txt

$ls -ls .
total 80
0 drwxr-xr-x 25 ubuntu root  800 Jan  3 18:33 bin
4 -rw-r--r--  1 ubuntu root  574 Dec 22 16:05 close_on_exec.c
4 -rw-r--r--  1 ubuntu root 1032 Jan  3 18:29 copy.c
4 -rw-r--r--  1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
4 -rw-r--r--  1 ubuntu root 1140 Dec 22 14:20 ls_file.c
4 -rw-r--r--  1 ubuntu root 2258 Dec 22 14:37 ls_helper.h
4 -rw-r--r--  1 ubuntu root  666 Dec 22 16:05 move.c
4 -rw-r--r--  1 ubuntu root   53 Dec 22 06:20 source.txt
```

6 Write a C program which creates a child process and the parent waits for child’s exit.

parent_child_wait.c

```c
// Creates a child processes and the parent waits until its children exit.

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_CHILDS 5

int main() {
    for(int i = 0; i < MAX_CHILDS; i++)
        if(!fork()) {
            printf("Parent(pid) %d -> Child(pid) %d:\t", getppid(), getpid());
            srand(getpid());

            int n = 100;
            long sum = 0;
            for(int i = 0; i < n; i++) {
                sum += rand();
            }
            printf("Sum of %d random numbers = %ld\n", n, sum);

            exit(0);
        }

    for(int i = 0; i < MAX_CHILDS; i++) {
        int wstatus;

        int cid = wait(&wstatus);

        if(WIFEXITED(wstatus))
            printf("\033[0;32m> child(pid): %d of parent (pid): %d exited normally with status: "
                    "%d\033[0m\n", cid, getpid(), WEXITSTATUS(wstatus));
        else
            printf("\033[0;31m> child(pid): %d of parent (pid): %d exited abnormally\033[0m\n",
                    cid, getpid());
    }

    return 0;
}
```

Output

```
$ ./bin/parent_child_wait
parent(pid) 193 -> Child(pid) 194:        Sum of 100 random numbers = 112349801883
parent(pid) 193 -> Child(pid) 195:        Sum of 100 random numbers = 110287567662
parent(pid) 193 -> Child(pid) 196:        Sum of 100 random numbers = 117306909504
parent(pid) 193 -> Child(pid) 197:        Sum of 100 random numbers = 100113144996
> child(pid): 194 of parent (pid): 193 exited normally with status: 0
> child(pid): 195 of parent (pid): 193 exited normally with status: 0
parent(pid) 193 -> Child(pid) 198:        Sum of 100 random numbers = 99795688877
> child(pid): 196 of parent (pid): 193 exited normally with status: 0
> child(pid): 197 of parent (pid): 193 exited normally with status: 0
> child(pid): 198 of parent (pid): 193 exited normally with status: 0

$ ./bin/parent_child_wait
parent(pid) 199 -> Child(pid) 200:        Sum of 100 random numbers = 97012711868
parent(pid) 199 -> Child(pid) 201:        Sum of 100 random numbers = 113382333867
> child(pid): 200 of parent (pid): 199 exited normally with status: 0
> child(pid): 201 of parent (pid): 199 exited normally with status: 0
parent(pid) 199 -> Child(pid) 202:        Sum of 100 random numbers = 108751118111
parent(pid) 199 -> Child(pid) 203:        Sum of 100 random numbers = 117941568918
parent(pid) 199 -> Child(pid) 204:        Sum of 100 random numbers = 102851697320
> child(pid): 202 of parent (pid): 199 exited normally with status: 0
> child(pid): 203 of parent (pid): 199 exited normally with status: 0
> child(pid): 204 of parent (pid): 199 exited normally with status: 0
```

7 Write a C program to demonstrate the difference between the fork and vfork system calls.

fork_vfork.c

```c
// Demonstrates the difference between the fork and vfork system calls.

#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

void forktest();
void vforktest();

int main() {
    forktest();
    vforktest();

    return 0;
}

void forktest() {
    int a = 3, b = 2;

    if(!fork()) {
        a++;
        b++;

        printf("fork child   -> a: %d; b: %d\n", a, b);
        exit(0);
    }

    int cid = wait(NULL);
    printf("fork parent  -> a: %d; b: %d\n", a, b);
}

void vforktest() {
    int a = 3, b = 2;

    if(!vfork()) {
        a++;
        b++;

        printf("vfork child  -> a: %d; b: %d\n", a, b);
        // Make sure this is _exit() and not exit()
        _exit(0);
    }

    printf("vfork parent -> a: %d; b: %d\n", a, b);
}
```

Output

```
$ ./bin/fork_vfork
fork child   -> a: 4; b: 3
fork parent  -> a: 3; b: 2
vfork child  -> a: 4; b: 3
vfork parent -> a: 4; b: 3
```

8 Write a C program in which main process creates a child process and registers a signal handler to get the exit status of the child asynchronously.

async_child_wait.c

```
// Creates a child process and registers a signal handler to get exit status of child process
// asynchronously

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>

#define true (1)
#define false (0)

void signalHandler(int);

int stop = false;

int main() {
    srand(time(0));

    if(!fork()) {
        sleep(rand() % 15 + 1);
        exit((rand() % 100));
    }

    signal(SIGCHLD, signalHandler);

    int i = 1;
    while (!stop) {
        printf("\rWaiting for Child Exit Signal..... %d", i);
        fflush(stdout);
        sleep(1);
        i++;
    }

    return 0;
}

void signalHandler(int signum) {
    printf("\nChild Exit Signal Received.\n");

    int wstatus;
    int cid = wait(&wstatus);
    if(WIFEXITED(wstatus))
        printf("\033[0;32m> child(pid): %d of parent (pid): %d exited normally with status: "
                "%d\033[0m\n", cid, getpid(), WEXITSTATUS(wstatus));
    else
        printf("\033[0;31m> child(pid): %d of parent (pid): %d exited abnormally\033[0m\n",
                cid, getpid());
    stop = true;
}
```

Output

```
$ ./bin/async_child_wait
Waiting for Child Exit Signal.... 5s
Child Exit Signal Received.
> child(pid): 182 of parent (pid): 181 exited normally with status: 4

$ ./bin/async_child_wait
Waiting for Child Exit Signal.... 2s
Child Exit Signal Received.
> child(pid): 184 of parent (pid): 183 exited normally with status: 52

$ ./bin/async_child_wait
Waiting for Child Exit Signal.... 9s
Child Exit Signal Received.
> child(pid): 186 of parent (pid): 185 exited normally with status: 5
```

9 Implement ‘ls | wc -l -c –w’ command using pipe and exec functions.

pipe_exec.c

```c
// Implements `ls | wc -l -w -c` using pipe() and exec()

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    int pipefd[2];

    if(pipe(pipefd) < 0) {
        printf("Unable to create pipe. Exiting...");
        return -1;
    }

    if(!fork()) {
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[0]);
        close(pipefd[1]);

        execl("/usr/bin/ls", "ls", ".", NULL);
        _exit(1);
    }

    dup2(pipefd[0], STDIN_FILENO);
    close(pipefd[0]);
    close(pipefd[1]);

    execl("/usr/bin/wc", "wc", "-l", "-c", "-w", NULL);
    wait(NULL);

    return 0;
}
```

Output

```
$ ./bin/pipe_exec
    27      27     346

$ ls | wc -l -c -w
    27      27     346
```

10 Establish bidirectional communication between sender program and receiver program using multiple FIFO's.

fifo_server.c

```c
// Bidirectional communication between sender and receiver programs using multiple FIFOs.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>

#define true (1)
#define false (0)
#define BUF_SIZE 1024

void cleanup(int);

int server_fd = -1, client_fd = -1;

int main() {
    signal(SIGINT, cleanup);

    char *serverPath = "/tmp/fifo_ex_server";
    char *clientPath = "/tmp/fifo_ex_client";

    char buffer_in[BUF_SIZE], buffer_out[BUF_SIZE];

    if(mkfifo(serverPath, 0666) < 0) {
        printf("Error creating FIFO file. Exiting...\n");
        return -1;
    }

    printf("Server Program.\n\n");
    while(true) {
        printf("> ");
        scanf("%1000[^\n]%*c", buffer_out);

        if((server_fd = open(serverPath, O_WRONLY)) == -1) {
            printf("Error opening FIFO file. Exiting...\n");
            break;
        }

        write(server_fd, buffer_out, strlen(buffer_out)+1);
        close(server_fd);
        server_fd = -1;

        if((client_fd = open(clientPath, O_RDONLY)) == -1) {
            printf("Error opening FIFO file. Exiting...\n");
            break;
        }

        read(client_fd, &buffer_in, BUF_SIZE);
        printf("Client: %s\n", buffer_in);
        close(client_fd);
        client_fd = -1;
    }
```

```c

    return 0;
}

void cleanup(int signum) {
    printf("\nClosing FIFO Files and performing cleanup...\n");
    if(server_fd != -1) close(server_fd);
    if(client_fd != -1) close(client_fd);
    exit(0);
}
```

fifo_client.c

```c
// Bidirectional communication between sender and receiver programs using multiple FIFOs.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>

#define true (1)
#define false (0)
#define BUF_SIZE 1024

void cleanup(int);

```

```c
int server_fd = -1, client_fd = -1;

int main() {
    signal(SIGINT, cleanup);

    char *serverPath = "/tmp/fifo_ex_server";
    char *clientPath = "/tmp/fifo_ex_client";

    char buffer_in[BUF_SIZE], buffer_out[BUF_SIZE];

    if(mkfifo(clientPath, 0666) < 0) {
        printf("Error creating FIFO file. Exiting...\n");
        return -1;
    }

    printf("Client Program.\n\n");
    while(true) {
        if((server_fd = open(serverPath, O_RDONLY)) == -1) {
            printf("Error opening FIFO file. Exiting...\n");
            break;
        }

        read(server_fd, &buffer_in, BUF_SIZE);
        printf("Server: %s\n", buffer_in);
        close(server_fd);
        server_fd = -1;

        printf("> ");
        scanf("%1000[^\n]%*c", buffer_out);

        if((client_fd = open(clientPath, O_WRONLY)) == -1) {
            printf("Error opening FIFO file. Exiting...\n");
            break;
        }

        write(client_fd, buffer_out, strlen(buffer_out)+1);
        close(client_fd);
        client_fd = -1;
    }

    return 0;
}

void cleanup(int signum) {
    printf("\nClosing FIFO Files and performing cleanup...\n");
    if(server_fd != -1) close(server_fd);
    if(client_fd != -1) close(client_fd);
    exit(0);
}
```

Output - fifo_server.c

```
$ ./bin/fifo_server
Server Program.

> Hello, World!
Client: Hello, Server!
> How are You?
Client: Im Fine
> Bye
Client: Bye
> ^C
Closing FIFO Files and performing cleanup...
```

Output - fifo_client.c

```
$ ./bin/fifo_client
Client Program.

Server: Hello, World!
> Hello, Server!
Server: How are You?
> Im Fine
Server: Bye
> Bye
^C
Closing FIFO Files and performing cleanup...
```

11 Implement SVR based Message Queue IPC mechanism to establish asynchronous communication between two communicating processes.

mq_server_sync.c

```c
// Creates a Message Queue and communicates with mq_client_sync.c

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define true (1)
#define false (0)
#define BUF_SIZE (1024)

typedef struct {
    long msg_type;
    char msg_text[BUF_SIZE];
} message;

int main() {
    key_t key;
    int msg_id;

    printf("Server Program.\nType \".exit\" to exit.\n\n");

    if((key = ftok("mq_server_sync.c", 69)) < 0) {
        printf("Error creating key using ftok(). Exiting...\n");
        return -1;
    }

    if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
        printf("Error getting message queue using msgget(). Exiting...\n");
        return -1;
    }

    message msg;
    while(true) {
        printf("> ");
        scanf("%1000[^\n]%*c", msg.msg_text);
        msg.msg_type = 1;
        msgsnd(msg_id, &msg, BUF_SIZE, 0);
        if(strcmp(msg.msg_text, ".exit") == 0) break;

        msgrcv(msg_id, &msg, BUF_SIZE, 2, 0);
        printf("Client: %s\n", msg.msg_text);
        if(strcmp(msg.msg_text, ".exit") == 0) break;
    }

    msgctl(msg_id, IPC_RMID, NULL);
    return 0;
}
```

mq_client_sync.c

```c
// Creates a Message Queue and communicates with mq_server_sync.c

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define true (1)
#define false (0)
#define BUF_SIZE (1024)

typedef struct {
    long msg_type;
    char msg_text[BUF_SIZE];
} message;

int main() {
    key_t key;
    int msg_id;

    printf("Client Program.\nType \".exit\" to exit.\n\n");

    if((key = ftok("mq_server_sync.c", 69)) < 0) {
        printf("Error creating key using ftok(). Exiting...\n");
        return -1;
    }

    if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
        printf("Error getting message queue using msgget(). Exiting...\n");
        return -1;
    }

    message msg;
    while(true) {
        msgrcv(msg_id, &msg, BUF_SIZE, 1, 0);
        printf("Server: %s\n", msg.msg_text);
        if(strcmp(msg.msg_text, ".exit") == 0) break;

        printf("> ");
        scanf("%1000[^\n]%*c", msg.msg_text);
        msg.msg_type = 2;
        msgsnd(msg_id, &msg, BUF_SIZE, 0);
        if(strcmp(msg.msg_text, ".exit") == 0) break;
    }

    return 0;
}
```

Output - mq_server_sync.c

```
$ ./bin/mq_server_sync
Server Program.
Type ".exit" to exit.

> Hello, World
Client: Hello, Server
> This is a Message
Client: 200 OK
> .exit
```

Output - mq_client_sync.c

```
$ ./bin/mq_client_sync
Client Program.
Type ".exit" to exit.

Server: Hello, World
> Hello, Server
Server: This is a Message
> 200 OK
Server: .exit
```

```
$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x454aa1f3 32768      ubuntu     666        1024         1
```

12 Program to demonstrate IPC by implementing Server-Client Model to perform ‘isEven’ & ‘isPrime’ operations

Implement the following communication model:

- Process 1 enacts the server role
- Process 2 and 3 are clients
- Process 2 seeks ‘isprime’ service from the server by inserting the payload in the message queue
- Process 3 seeks ‘iseven’ service form the server by inserting the payload in the message queue
- Server retrieves the service request from the Message queue and inserts the reply. Intended Client retrieves the response.

prog_12/message.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>

typedef struct {
    long req_type;
    int req_data;
    int req_service;
} request;

typedef struct {
    long res_type;
    int res_data;
} response;

#define REQ_SIZE (2 * sizeof(int))
#define RES_SIZE (sizeof(int))

#define MSG_TYPE_REQ 1
#define MSG_TYPE_RES_EVEN 2
#define MSG_TYPE_RES_PRIME 3

#define REQ_SRV_EVEN 1
#define REQ_SRV_PRIME 2
```

prog_12/server.c

```
// Creates a Message Queue and acts as a Server providing "isPrime" & "isEven" services.

#include <math.h>
#include "message.h"

void sigint(int);
int isEven(int);
int isPrime(int);

key_t key = -1;
int msg_id = -1;

int main() {
    signal(SIGINT, sigint);

    printf("Server Program.\nPress Ctrl+C to exit.\n\n");

    if((key = ftok("message.h", 69)) < 0) {
        printf("Error creating key using ftok(). Exiting...\n");
        return -1;
    }

    if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
        printf("Error getting message queue using msgget(). Exiting...\n");
        return -1;
    }

    request req;
    response res;

    while(true) {
        msgrcv(msg_id, &req, REQ_SIZE, MSG_TYPE_REQ, 0);

        if(req.req_service == REQ_SRV_EVEN) {
            printf("Received Request: isEven(%d)... ", req.req_data);

            res.res_type = MSG_TYPE_RES_EVEN;
            res.res_data = isEven(req.req_data);
            msgsnd(msg_id, &res, RES_SIZE, 0);

            printf("Processed.\n");
        } else if(req.req_service == REQ_SRV_PRIME) {
            printf("Received Request: isPrime(%d)... ", req.req_data);

            res.res_type = MSG_TYPE_RES_PRIME;
            res.res_data = isPrime(req.req_data);
            msgsnd(msg_id, &res, RES_SIZE, 0);

            printf("Processed.\n");
        } else {
            printf("Received Request: Unknown... Discarding.\n");
        }
    }

    return 0;
}

void sigint(int signum) {
    printf("\nDeleting message queue and Stopping Server.....\n");
    if(msg_id != -1) msgctl(msg_id, IPC_RMID, NULL);
    exit(0);
}

int isEven(int n) {
    return n % 2 == 0;
}

int isPrime(int n) {
    if(n < 2) return false;

    int max = (int)sqrt(n);
    for(int i = 2; i <= max; i++) {
        if(n%i == 0) return false;
    }

    return true;
}
```

prog_12/client_is_even.c

```
// Using Message Queue, created by server.c, seeks "isEven" service.

#include "message.h"

int main() {
    key_t key = -1;
    int msg_id = -1;

    printf("Client (isEven) Program.\nPress Ctrl+C to exit.\n\n");

    if((key = ftok("message.h", 69)) < 0) {
        printf("Error creating key using ftok(). Exiting...\n");
        return -1;
    }

    if((msg_id = msgget(key, 0666)) < 0) {
        printf("Error getting message queue using msgget().\nTry running the Server first.\n"
                    "Exiting...\n");
        return -1;
    }

    request req;
    response res;
    req.req_type = MSG_TYPE_REQ;

```

```c
    while(true) {
        printf("Enter a Number: ");
        scanf("%d", &req.req_data);
        req.req_service = REQ_SRV_EVEN;
        if(msgsnd(msg_id, &req, REQ_SIZE, 0) < 0) {
            printf("Unable to send request to Server process.\n\n");
            continue;
        }

        if(msgrcv(msg_id, &res, RES_SIZE, MSG_TYPE_RES_EVEN, 0) < 0) {
            printf("Unable to receive response from Server process.\n\n");
            continue;
        }

        if(res.res_data)
            printf("%d is an even no.\n\n", req.req_data);
        else
            printf("%d is an odd no.\n\n", req.req_data);
    }

    return 0;
}
```

prog_12/client_is_prime.c

```c
// Using Message Queue, created by server.c, seeks "isPrime" service.

#include "message.h"

int main() {
    key_t key = -1;
    int msg_id = -1;

    printf("Client (isPrime) Program.\nPress Ctrl+C to exit.\n\n");

    if((key = ftok("message.h", 69)) < 0) {
        printf("Error creating key using ftok(). Exiting...\n");
        return -1;
    }

    if((msg_id = msgget(key, 0666)) < 0) {
        printf("Error getting message queue using msgget().\nTry running the Server first.\n"
                    "Exiting...\n");
        return -1;
    }

    request req;
    response res;
    req.req_type = MSG_TYPE_REQ;

    while(true) {
        printf("Enter a Number: ");
        scanf("%d", &req.req_data);
        req.req_service = REQ_SRV_PRIME;

        if(msgsnd(msg_id, &req, REQ_SIZE, 0) < 0) {
            printf("Unable to send request to Server process.\n\n");
            continue;
        }

        if(msgrcv(msg_id, &res, RES_SIZE, MSG_TYPE_RES_PRIME, 0) < 0) {
            printf("Unable to receive response from Server process.\n\n");
            continue;
        }
        if(res.res_data)
            printf("%d is a prime no.\n\n", req.req_data);
        else
            printf("%d is not a prime no.\n\n", req.req_data);
    }

    return 0;
}
```

Output - Process 1 (server.c)

```
$ ./bin/server
Server Program.
Press Ctrl+C to exit.

Received Request: isEven(5)... Processed.
Received Request: isEven(8)... Processed.
Received Request: isEven(0)... Processed.
Received Request: isPrime(5)... Processed.
Received Request: isPrime(25)... Processed.
Received Request: isPrime(-10)... Processed.
Received Request: isEven(-3)... Processed.
Received Request: isPrime(0)... Processed.
^C
Deleting message queue and Stopping Server.....
```

Output - Process 2 (client_is_prime.c)

```
$ ./bin/client_is_prime
Client (isPrime) Program.
Press Ctrl+C to exit.

Enter a Number: 5
5 is a prime no.

Enter a Number: 25
25 is not a prime no.

Enter a Number: -10
-10 is not a prime no.

Enter a Number: 0
0 is not a prime no.

Enter a Number: ^C
```

Output - Process 3 (client_is_even.c)

```
$ ./bin/client_is_even
Client (isEven) Program.
Press Ctrl+C to exit.

Enter a Number: 5
5 is an odd no.

Enter a Number: 8
8 is an even no.

Enter a Number: 0
0 is an even no.

Enter a Number: -3
-3 is an odd no.

Enter a Number: ^C
```

13 Implement Shared Memory based communication model

Implement the following features:

- Server and multiple clients communicate with each other through shared memory.
- Synchronization of SHM access is realized through semaphores.

`prog_13/shm_helper.h`

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define MAX_COUNT 25

#define SEM_COUNT 2
#define SEM_READ 0
#define SEM_WRITE 1
#define SEMOP_WAIT -1
#define SEMOP_RELEASE 1

void initialize();

key_t key = -1;
int shm_id = -1, sem_id = -1;

```

```c
void initialize() {
    if((key = ftok("shm_helper.c", 69)) < 0) {
        printf("Error creating key using ftok(): %s\n", strerror(errno));
        exit(-1);
    }

    if((shm_id = shmget(key, sizeof(int), 0600 | IPC_CREAT)) < 0) {
        printf("Error getting shm_id using shmget(): %s\n", strerror(errno));
        exit(-1);
    }

    if((sem_id = semget(key, SEM_COUNT, 0600 | IPC_CREAT)) < 0) {
        printf("Error getting sem_id using semget(): %s\n", strerror(errno));
        exit(-1);
    }
}
```

prog_13/shm_writer.c

```c
// Gets a Shared Memory (creates if doesn't exists) and writes MAX_COUNT (defined in
// "shm_helper.h") random numbers and uses Semaphores for synchronization of shm access

#include "shm_helper.h"

void cleanup();

int *buf = NULL;

int main() {
    struct sembuf sbuf;
    sbuf.sem_flg = SEM_UNDO;

    atexit(cleanup);
    signal(SIGINT, exit);
    initialize();

    if((buf = (int*) shmat(shm_id, NULL, 0)) == NULL) {
        printf("Error attaching shm using shmat(): %s\n", strerror(errno));
        return -1;
    }

    printf("Writing %d Random Numbers.\n", MAX_COUNT);
    for(int i = 0; i < MAX_COUNT; i++) {
        sbuf.sem_num = SEM_READ;
        sbuf.sem_op = SEMOP_WAIT;
        semop(sem_id, &sbuf, 1);

        *buf = rand();
        printf("\033[0;32m Wrote: %3d\033[0m\n", *buf);

        sbuf.sem_num = SEM_WRITE;
        sbuf.sem_op = SEMOP_RELEASE;
        semop(sem_id, &sbuf, 1);
    }
    printf("Done.....\n");

    return 0;
}

void cleanup() {
```

```c
    if(buf != NULL) shmdt(buf);
    if(shm_id != -1) shmctl(shm_id, IPC_RMID, NULL);
    if(sem_id != -1) semctl(sem_id, 0, IPC_RMID);
}
```

prog_13/shm_reader.c

```c
// Gets a Shared Memory (creates if doesn't exists) and reads MAX_COUNT (defined in
// "shm_helper.h") random numbers and uses Semaphores for synchronization of shm access

#include "shm_helper.h"

void cleanup();

int *buf = NULL;

int main() {
    struct sembuf sbuf;
    sbuf.sem_flg = SEM_UNDO;

    atexit(cleanup);
    signal(SIGINT, exit);
    initialize();

    if((buf = (int*) shmat(shm_id, NULL, 0)) == NULL) {
        printf("Error attaching shm using shmat(): %s\n", strerror(errno));
        return -1;
    }

    printf("Reading %d Random Numbers.\n", MAX_COUNT);
    for(int i = 0; i < MAX_COUNT; i++) {
        sbuf.sem_num = SEM_READ;
        sbuf.sem_op = SEMOP_RELEASE;
        semop(sem_id, &sbuf, 1);

        sbuf.sem_num = SEM_WRITE;
        sbuf.sem_op = SEMOP_WAIT;
        semop(sem_id, &sbuf, 1);

        printf("\033[0;31m Read: %3d\033[0m\n", *buf);
    }
    printf("Done.....\n");

    return 0;
}

void cleanup() {
    if(buf != NULL) shmdt(buf);
}
```

Output - Writer Process (shm_writer.c)

```
$ ./bin/shm_writer
Writing 25 Random Numbers.
 Wrote: 1804289383
 Wrote: 846930886
 Wrote: 1681692777
 Wrote: 1714636915
 Wrote: 1957747793
 Wrote: 424238335
 Wrote: 719885386
 Wrote: 1649760492
 Wrote: 596516649
 Wrote: 1189641421
 Wrote: 1025202362
 Wrote: 1350490027
 Wrote: 783368690
 Wrote: 1102520059
 Wrote: 2044897763
 Wrote: 1967513926
 Wrote: 1365180540
 Wrote: 1540383426
 Wrote: 304089172
 Wrote: 1303455736
 Wrote: 35005211
 Wrote: 521595368
 Wrote: 294702567
 Wrote: 1726956429
 Wrote: 336465782
Done.....
```

Output - Reader Process (shm_reader.c)

```
$ ./bin/shm_reader
Reading 25 Random Numbers.
 Read: 1804289383
 Read: 846930886
 Read: 1681692777
 Read: 1714636915
 Read: 1957747793
 Read: 424238335
 Read: 719885386
 Read: 1649760492
 Read: 596516649
 Read: 1189641421
 Read: 1025202362
 Read: 1350490027
 Read: 783368690
 Read: 1102520059
 Read: 2044897763
 Read: 1967513926
 Read: 1365180540
 Read: 1540383426
 Read: 304089172
 Read: 1303455736
 Read: 35005211
 Read: 521595368
 Read: 294702567
 Read: 1726956429
 Read: 336465782
Done.....
```

14 Implement client/server model using socket API.

prog_14/server.c

```c
// TCP Server using Linux sockets API

#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <string.h>

#define true (1)
#define false (0)
#define BACKLOG 16
#define BUF_SIZE 1024

int main() {
    int tcp_socket = -1, conn_fd = -1;
    char msg[BUF_SIZE];

    printf("Server Program. (Simple Message Echo Server)\nPress Ctrl+C to exit.\n\n");

    if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int server_addr_size = sizeof(server_addr);

    if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
        printf("Unable to bind socket to server address: %s\n", strerror(errno));
        return -1;
    }

    if(listen(tcp_socket, BACKLOG) < 0) {
        printf("Unable to listen to socket: %s\n", strerror(errno));
        return -1;
    }

    while(true) {
        if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
                                &server_addr_size)) < 0) {
            printf("Unable to accept new connection: %s\n", strerror(errno));
            break;
        }

        if(recv(conn_fd, msg, BUF_SIZE, 0) < 0) {
            printf("Unable to read message: %s\n", strerror(errno));
            break;
        }

        printf("Received: %s; Echoing back message... ", msg);
        if(send(conn_fd, msg, BUF_SIZE, 0) < 0) {
            printf("Unable to send message: %s\n", strerror(errno));
            break;
```

```c
        }

        printf("Message Sent.\n");
        fflush(stdout);

        if(conn_fd != -1) close(conn_fd);
        conn_fd = -1;
    }

    return 0;
}
```

prog_14/client.c

```c
// TCP Client using Linux sockets API

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <string.h>

#define true (1)
#define false (0)
#define BACKLOG 16
#define BUF_SIZE 1024

```

```c
int main() {
    int tcp_socket = -1;
    char msg[BUF_SIZE];

    printf("Client Program.\nPress Ctrl+C to exit.\n\n");

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int server_addr_size = sizeof(server_addr);

    while(true) {
        printf("> ");
        scanf("%1000[^\n]%*c", msg);

        if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
            printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
            return -1;
        }

        if(connect(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
            printf("Unable to get new connection: %s\n", strerror(errno));
            break;
        }

        if(send(tcp_socket, msg, BUF_SIZE, 0) < 0) {
            printf("Unable to send message.: %s\n", strerror(errno));
            break;
        }

        if(recv(tcp_socket, msg, BUF_SIZE, 0) < 0) {
            printf("Unable to read message.: %s\n", strerror(errno));
            break;
        }

        printf("Server: %s\n", msg);

        if(tcp_socket != -1) close(tcp_socket);
        tcp_socket = -1;
    }

    if(tcp_socket != -1) close(tcp_socket);
    return 0;
}
```

Output - Server (server.c)

```
$ ./bin/server
Server Program. (Simple Message Echo Server)
Press Ctrl+C to exit.

Received: Hello, Server; Echoing back message... Message Sent.
Received: Hello, Server!; Echoing back message... Message Sent.
Received: This is Client 1; Echoing back message... Message Sent.
Received: Good Bye; Echoing back message... Message Sent.
Received: This is Client 2; Echoing back message... Message Sent.
Received: Byeee; Echoing back message... Message Sent.
```

Output - Client 1 (client.c)

```
$ ./bin/client
Client Program.
Press Ctrl+C to exit.

> Hello, Server
Server: Hello, Server
> This is Client 1
Server: This is Client 1
> Good Bye
Server: Good Bye
> ^C
```

Output - Client 2 (client.c)

```
$ ./bin/client
Client Program.
Press Ctrl+C to exit.

> Hello, Server!
Server: Hello, Server!
> This is Client 2
Server: This is Client 2
> Byeee
Server: Byeee
> ^C
```

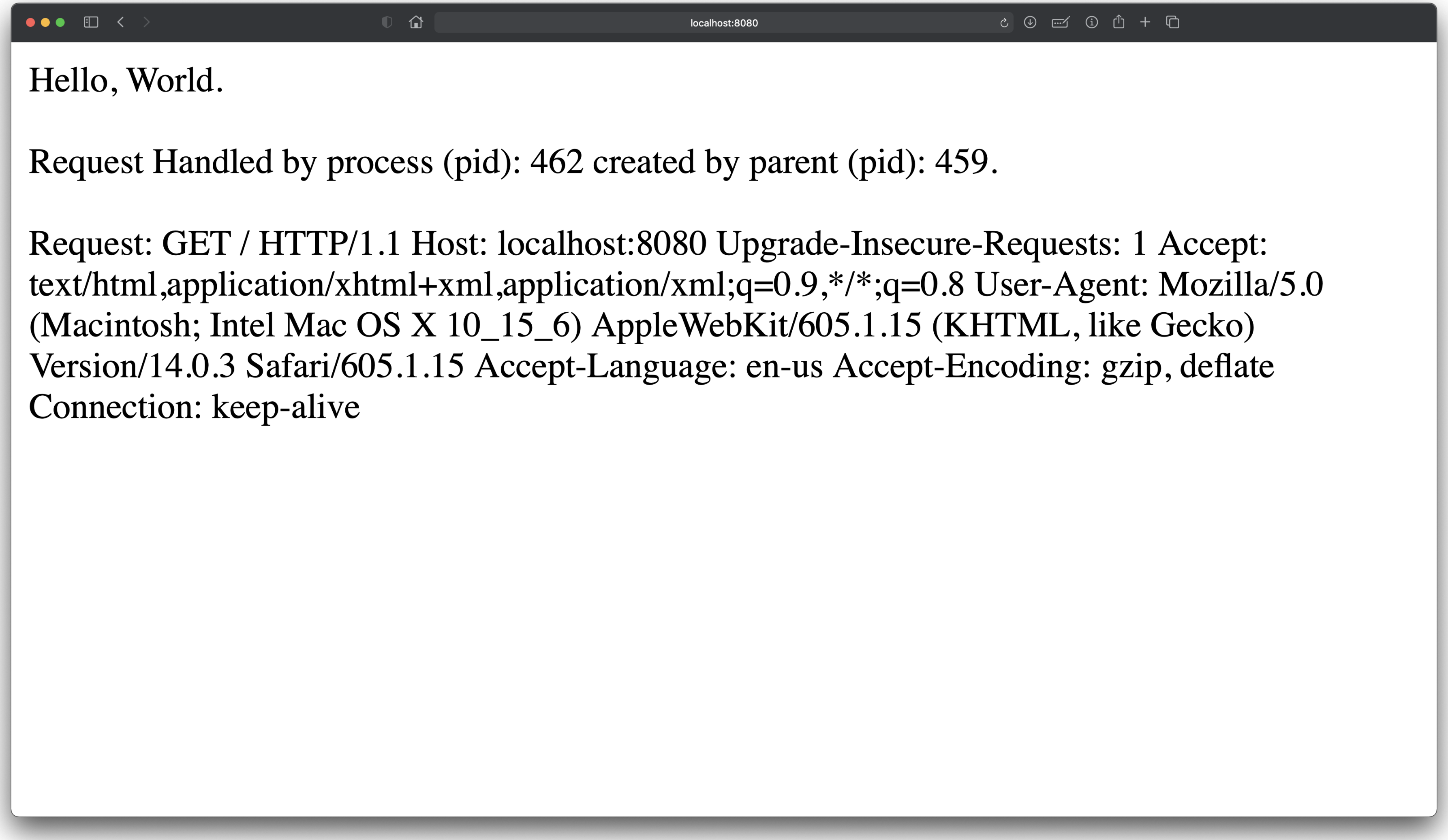
15 Implement concurrent server using fork based model while avoiding the zombie state of the client.

http_server_fork.c

```c
// TCP Server that accepts requests and creates child process to handle requests.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <errno.h>
#include <string.h>

#define true (1)
#define false (0)
#define BACKLOG 16
#define REQ_BUF_SIZE 1024
#define RES_BUF_SIZE 2048

void sigchld(int);

int main() {
    signal(SIGCHLD, sigchld);

    int tcp_socket = -1, conn_fd = -1, on = 1;
    char req[REQ_BUF_SIZE], res[RES_BUF_SIZE];

    printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");

    if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int server_addr_size = sizeof(server_addr);

    if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
        printf("Unable to bind socket to server address: %s\n", strerror(errno));
        return -1;
    }

    if(listen(tcp_socket, BACKLOG) < 0) {
        printf("Unable to listen to socket: %s\n", strerror(errno));
        return -1;
    }

    while(true) {
        if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
                                &server_addr_size)) < 0) {
            printf("Unable to accept new connection: %s\n", strerror(errno));
            continue;
        }

        if(!fork()) {
            close(tcp_socket);

            if(recv(conn_fd, req, REQ_BUF_SIZE, 0) < 0) {
                printf("Unable to read message: %s\n", strerror(errno));
                close(conn_fd);
                _exit(0);
            }

            printf("Received Request... ");

            sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World.<br/><br/>\n"
                    "Request Handled by process (pid): %d created by parent (pid): %d.<br/><br/>\n"
                    "Request: %s", getpid(), getppid(), req);

            if(send(conn_fd, res, strlen(res), 0) < 0) {
                printf("Unable to send message: %s\n", strerror(errno));
                close(conn_fd);
                _exit(0);
            }

            close(conn_fd);
            printf("Request Served. (cid: %d)\n", getpid());
            _exit(0);
        }

        close(conn_fd);
    }

    return 0;
}

void sigchld(int signum) {
    wait(NULL);
}
```

Terminal Output

```
$ ./bin/http_server_fork
Very Simple HTTP Web Server.
Press Ctrl+C to exit.

Received Request... Request Served. (cid: 462)
Received Request... Request Served. (cid: 463)
Received Request... Request Served. (cid: 464)
^C
```



```
Hello, World.

Request Handled by process (pid): 462 created by parent (pid): 459.

Request: GET / HTTP/1.1 Host: localhost:8080 Upgrade-Insecure-Requests: 1 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 User-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko)
Version/14.0.3 Safari/605.1.15 Accept-Language: en-us Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Output in Safari

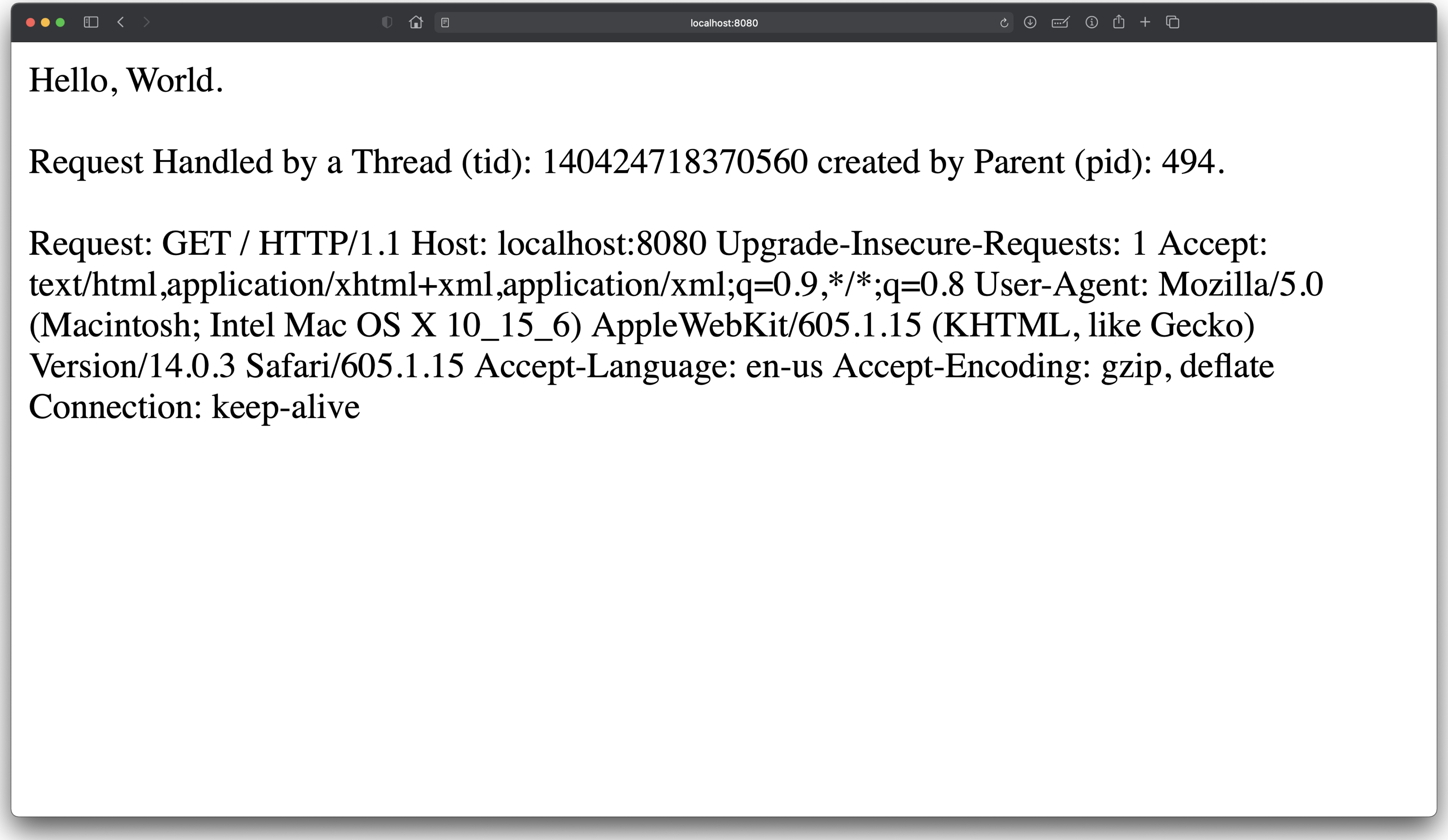
16 Implement a concurrent server model using pthread API.

http_server_pthread.c

```
// TCP Server that accepts requests and creates a thread using pthread to handle requests.

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <errno.h>
#include <string.h>

#define true (1)
#define false (0)
#define BACKLOG 16
#define REQ_BUF_SIZE 1024
#define RES_BUF_SIZE 2048

void* handleRequest(void*);

int main() {
    int tcp_socket = -1, conn_fd = -1, on = 1;

    printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");

    if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int server_addr_size = sizeof(server_addr);

    if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
        printf("Unable to bind socket to server address: %s\n", strerror(errno));
        return -1;
    }

    if(listen(tcp_socket, BACKLOG) < 0) {
        printf("Unable to listen to socket: %s\n", strerror(errno));
        return -1;
    }

    while(true) {
        if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
                                &server_addr_size)) < 0) {
            printf("Unable to accept new connection: %s\n", strerror(errno));
            continue;
        }

        pthread_t tid;
        int *new_conn_fd = (int*)malloc(sizeof(conn_fd));
        *new_conn_fd = conn_fd;
        if(pthread_create(&tid, NULL, handleRequest, (void*) new_conn_fd) != 0) {
            printf("Unable to create new thread: %s\n", strerror(errno));
            close(conn_fd);
            free(new_conn_fd);
            continue;
        }

        if(pthread_detach(tid) != 0) {
            printf("Unable to detach new thread: %s\n", strerror(errno));
            continue;
        }
    }

    return 0;
}

void* handleRequest(void *new_conn_fd) {
    int conn_fd = *((int*)new_conn_fd);
    char req[REQ_BUF_SIZE], res[RES_BUF_SIZE];

    if(recv(conn_fd, req, REQ_BUF_SIZE, 0) < 0) {
        printf("Unable to read message: %s\n", strerror(errno));
        close(conn_fd);
        free(new_conn_fd);
        pthread_exit(-1);
    }

    printf("Received Request... ");

    sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World.<br/><br/>\n"
            "Request Handled by a Thread (tid): %ld created by Parent (pid): %d.<br/><br/>\n"
            "Request: %s", pthread_self(), getpid(), req);

    if(send(conn_fd, res, strlen(res), 0) < 0) {
        printf("Unable to send message: %s\n", strerror(errno));
        close(conn_fd);
        free(new_conn_fd);
        pthread_exit(-1);
    }

    close(conn_fd);
    printf("Request Served. (tid: %ld)\n", pthread_self());
    free(new_conn_fd);
    pthread_exit(0);
}
```

Terminal Output

```
$ ./bin/http_server_pthread
Very Simple HTTP Web Server.
Press Ctrl+C to exit.

Received Request... Request Served. (tid: 140424718370560)
Received Request... Request Served. (tid: 140351225526016)
Received Request... Request Served. (tid: 140424709977856)
^C
```



Hello, World.

Request Handled by a Thread (tid): 140424718370560 created by Parent (pid): 494.

Request: GET / HTTP/1.1 Host: localhost:8080 Upgrade-Insecure-Requests: 1 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Safari/605.1.15 Accept-Language: en-us Accept-Encoding: gzip, deflate Connection: keep-alive

Output in Safari

17 Solve the producer consumer problem using pthread API.

prod_con_pthread.c

```c
// Solves Producer - Consumer problem using pthreads and Mutex

#include <stdio.h>
#include <pthread.h>

#define BUF_SIZE 6
#define MAX_COUNT 42

int buf[BUF_SIZE];
int size = 0, front = 0, rear = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t m_produce = PTHREAD_COND_INITIALIZER;
pthread_cond_t m_consume = PTHREAD_COND_INITIALIZER;

void insert();
int delete();
void* producer(void*);
void* consumer(void*);

int main() {
    pthread_t producer_tid, consumer_pid;

    pthread_create(&producer_tid, NULL, producer, NULL);
    pthread_create(&consumer_pid, NULL, consumer, NULL);

    pthread_join(producer_tid, NULL);
    pthread_join(consumer_pid, NULL);

    return 0;
}

void insert(int data) {
    buf[rear] = data;
    rear = (rear + 1) % BUF_SIZE;
    size++;
}

int delete() {
    int data = buf[front];
    front = (front + 1) % BUF_SIZE;
    size--;
    return data;
}

void* producer(void* arg) {
    for(int i = 0; i < MAX_COUNT; i++) {
        pthread_mutex_lock(&mutex);
        while(size == BUF_SIZE) pthread_cond_wait(&m_produce, &mutex);
        insert(i);
        printf("\033[0;32m Produced: %3d\033[0m\n", i);
        pthread_cond_signal(&m_consume);
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}

```

```
void* consumer(void* arg) {
    for(int i = 0; i < MAX_COUNT; i++) {
        pthread_mutex_lock(&mutex);
        while(size == 0) pthread_cond_wait(&m_consume, &mutex);
        int data = delete();
        printf("\033[0;31m Consumed: %3d\033[0m\n", data);
        pthread_cond_signal(&m_produce);
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}
```

Output

```
$./bin/prod_con_pthread
 Produced:   0
 Produced:   1
 Produced:   2
 Produced:   3
 Produced:   4
 Produced:   5
 Consumed:   0
 Consumed:   1
 Consumed:   2
 Consumed:   3
 Consumed:   4
 Consumed:   5
 Produced:   6
 Produced:   7
 Produced:   8
 Produced:   9
 Produced:  10
 Produced:  11
 Consumed:   6
 Consumed:   7
 Consumed:   8
 Consumed:   9
 Consumed:  10
 Consumed:  11
 Produced:  12
 Produced:  13
 Produced:  14
 Produced:  15
```

```
Produced:  16
Produced:  17
Consumed:  12
Consumed:  13
Consumed:  14
Consumed:  15
Consumed:  16
Consumed:  17
Produced:  18
Produced:  19
Produced:  20
Produced:  21
Produced:  22
Produced:  23
Consumed:  18
Consumed:  19
Consumed:  20
Consumed:  21
Consumed:  22
Consumed:  23
Produced:  24
Produced:  25
Produced:  26
Produced:  27
Produced:  28
Produced:  29
Consumed:  24
```

```
Consumed:  25
Consumed:  26
Consumed:  27
Consumed:  28
Consumed:  29
Produced:  30
Produced:  31
Produced:  32
Produced:  33
Produced:  34
Produced:  35
Consumed:  30
Consumed:  31
Consumed:  32
Consumed:  33
Consumed:  34
Consumed:  35
Produced:  36
Produced:  37
Produced:  38
Produced:  39
Produced:  40
Produced:  41
Consumed:  36
Consumed:  37
Consumed:  38
Consumed:  39
Consumed:  40
Consumed:  41
```

18 Implement peer-to-peer communication model using socket API.

prog_18/server.c

```
// Peer-to-peer UDP Server using Linux sockets API

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <signal.h>
#include <string.h>
#include <errno.h>

#define BUF_SIZE 1024

int udp_socket = -1;

void cleanup();

int main() {
    atexit(cleanup);
    signal(SIGINT, exit);

    int on = 1;
    char buf[BUF_SIZE];
    struct sockaddr_in server, client;

    printf("Server program.\nPress Ctrl+C to exit.\n\n");

    if((udp_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("Unable to get UDP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    setsockopt(udp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    inet_aton("127.0.0.1", &server.sin_addr);
    int server_size = sizeof(server);
    int client_size = sizeof(client);

    if(bind(udp_socket, (struct sockaddr*) &server, server_size) < 0) {
        printf("Unable to bind socket to server address: %s\n", strerror(errno));
        return -1;
    }

    while(true) {
        int buf_size = recvfrom(udp_socket, buf, BUF_SIZE, 0,
                                (struct sockaddr*) &client, &client_size);

        sendto(udp_socket, buf, buf_size, 0, (struct sockaddr*) &client, client_size);

        printf("[Echoed message from %s:%d] > %s\n",
                inet_ntoa(client.sin_addr), (int) (client.sin_port), buf);
    }

    return 0;
}

void cleanup() {
    printf("\nShutting down server.....\n");
    close(udp_socket);
}
```

prog_18/client.c

```
// Peer-to-peer UDP Client using Linux sockets API

#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <string.h>
#include <errno.h>

#define BUF_SIZE 1024

int main() {
    int udp_socket = -1;
    int on = 1;
    char buf_out[BUF_SIZE], buf_in[BUF_SIZE];
    struct sockaddr_in server, client;

    printf("Client program.\n\n");
```

```c

    if((udp_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("Unable to get UDP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    inet_aton("127.0.0.1", &server.sin_addr);
    int server_size = sizeof(server);

    printf("> ");
    scanf("%1000[^\n]%*c", buf_out);

    sendto(udp_socket, buf_out, strlen(buf_out)+1, 0, (struct sockaddr*) &server, server_size);
    recvfrom(udp_socket, buf_in, BUF_SIZE, 0, NULL, NULL);

    printf("\nSent: %s\n", buf_out);
    printf("Echo: %s\n", buf_in);

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

Output - server (server.c)

```
$ ./bin/server
Server program.
Press Ctrl+C to exit.

[Echoed message from 127.0.0.1:55222] > Hello, World!
[Echoed message from 127.0.0.1:35540] > Hello, Server!
```

Output - Client 1 (client.c)

```
$ ./bin/client
Client program.

> Hello, World!

Sent: Hello, World!
Echo: Hello, World!
```

Output - Client 2 (client.c)

```
$ ./bin/client
Client program.

> Hello, Server!

Sent: Hello, Server!
Echo: Hello, Server!
```

19 Solve the process synchronization on I/O using record locking mechanism.

record_locking.c

```
// Process Synchronization on I/O using Record Locking

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

#define BUF_SIZE 64
#define MIN(x, y) (((x) < (y)) ? (x) : (y))

int main(int argc,  char *argv[]) {
    if(argc != 2) {
        printf("Invalid arguments.\nUsage: %s <file>\n", argv[0]);
        return -1;
    }

    struct flock fl = {F_UNLCK, SEEK_SET, 0, 100, 0};
    int fsize, offset, fd, pid, read_size;
    char buf[BUF_SIZE];

    if((fd = open(argv[1], O_RDWR)) == -1) {
        printf("Unable to open %s: %s\n", argv[1], strerror(errno));
        return -1;
    }

    printf("Press any key to Lock the File\n");
    printf("***********************************\n");
    getchar();

    fl.l_type = F_WRLCK;
    fl.l_pid = getpid();
    if(fcntl(fd, F_SETLK, &fl) == -1) {
        printf("Cannot Set Exclusive Lock on %s: %s\n", argv[1], strerror(errno));
        close(fd);
        return -1;
    } else if(fl.l_type != F_UNLCK && fl.l_type != F_RDLCK)
        printf("%s has been Exclusively Locked by Process: %d\n", argv[1], fl.l_pid);
    else
        printf("%s is NOT Locked\n", argv[1]);


    printf("Press any key to release the lock\n");
    printf("***********************************\n");
    getchar();

    fl.l_type = F_UNLCK;
    printf("File has been Unlocked \n");

    fsize = lseek(fd, 0, SEEK_END);
    offset = fsize - MIN(BUF_SIZE, fsize);
    lseek(fd, offset, SEEK_SET);

    read_size = read(fd, buf, MIN(BUF_SIZE, fsize));
    buf[read_size] = '\0';
    printf("Last %d bytes:\n********************************************\n%s\n", read_size, buf);
```

```c

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

Output - Process 1 (record_locking.c)

```
$ ./bin/record_locking record_locking.c
Press any key to Lock the File
***********************************

record_locking.c has been Exclusively Locked by Process: 593
Press any key to release the lock
***********************************

File has been Unlocked
Last 64 bytes:
********************************************
**\n%s\n", read_size, buf);

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

Output - Process 2 (record_locking.c)

```
$ ./bin/record_locking record_locking.c
Press any key to Lock the File
***********************************

Cannot Set Exclusive Lock on record_locking.c: Resource temporarily unavailable
```

20 Implement I/O multiplexing using select system call.

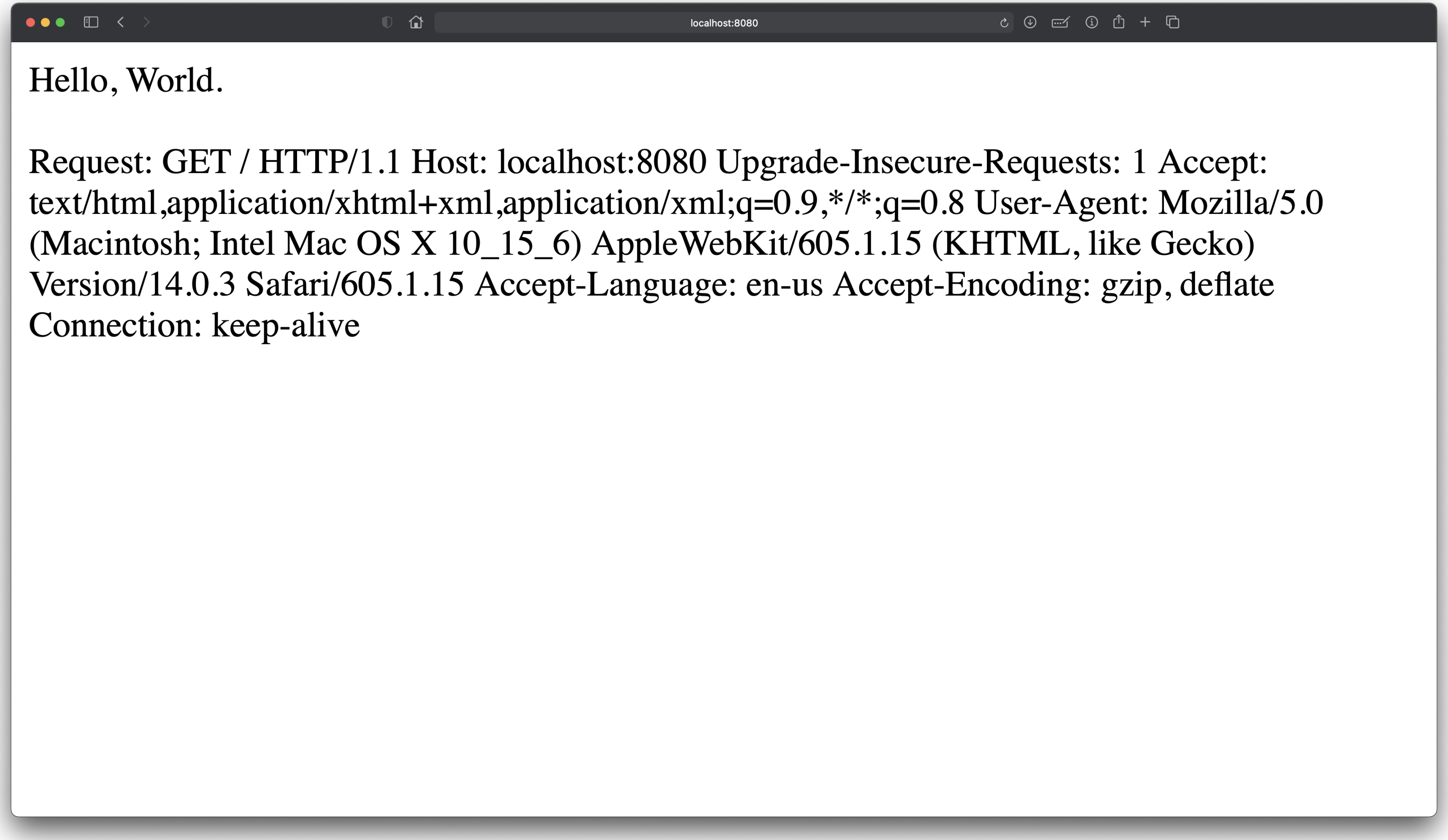
`http_server_select.c`

```c
// TCP Server that accepts requests and uses select() system call (I/O multiplexing) to
// handle requests.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/select.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <time.h>
#include <errno.h>
#include <string.h>

#define true (1)
#define BACKLOG 16
#define BUF_SIZE 1024

int main() {
    int tcp_socket = -1, conn_fd = -1, on = 1;
    char req[BUF_SIZE], res[BUF_SIZE];
    fd_set active_fd_set, read_fd_set;
    struct timeval tm;

    printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");
    if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
        return -1;
    }

    setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int server_addr_size = sizeof(server_addr);

    if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
        printf("Unable to bind socket to server address: %s\n", strerror(errno));
        return -1;
    }

    if(listen(tcp_socket, BACKLOG) < 0) {
        printf("Unable to listen to socket: %s\n", strerror(errno));
        return -1;
    }

    FD_ZERO(&active_fd_set);
    FD_SET(tcp_socket, &active_fd_set);

    while(true) {
        tm.tv_sec = 5; tm.tv_usec = 0;
        read_fd_set = active_fd_set;
        if(select(FD_SETSIZE, &read_fd_set, NULL, NULL, &tm) < 0) {
            printf("Erorr with select(): %s\n", strerror(errno));
            return -1;
        }

        for (int i = 0; i < FD_SETSIZE; i++) {
            if (FD_ISSET(i, &read_fd_set)) {
                if (i == tcp_socket) {
                    if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
                                        &server_addr_size)) < 0) {
                        printf("Unable to accept new connection: %s\n", strerror(errno));
                        continue;
                    }

                    printf("Connection Accepted... File Descriptor: %d\n", conn_fd);
                    FD_SET(conn_fd, &active_fd_set);
                } else {
                    if(recv(i, req, REQ_BUF_SIZE, 0) < 0) {
                        printf("Unable to read message: %s\n", strerror(errno));
                        close(i);
                        FD_CLR(i, &active_fd_set);
                        continue;
                    }

                    printf("Received Request... ");
                    sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World."
                            "<br/><br/>\nRequest: %s", req);

                    if(send(i, res, strlen(res), 0) < 0) {
                        printf("Unable to send message: %s\n", strerror(errno));
                        close(i);
                        FD_CLR(i, &active_fd_set);
                        continue;
```

```c
                    }
                    printf("Request Served. File Descriptor: %d\n", i);

                    close(i);
                    FD_CLR(i, &active_fd_set);
                }
            }
        }
    }

    return 0;
}
```

Terminal Output

```
$ ./bin/http_server_select
Very Simple HTTP Web Server.
Press Ctrl+C to exit.

Connection Accepted... File Descriptor: 4
Connection Accepted... File Descriptor: 5
Received Request... Request Served. File Descriptor: 5
Connection Accepted... File Descriptor: 5
Received Request... Request Served. File Descriptor: 4
Connection Accepted... File Descriptor: 4
Received Request... Request Served. File Descriptor: 5
Received Request... Request Served. File Descriptor: 4
```



```
Hello, World.

Request: GET / HTTP/1.1 Host: localhost:8080 Upgrade-Insecure-Requests: 1 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 User-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko)
Version/14.0.3 Safari/605.1.15 Accept-Language: en-us Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Output in Safari