

# 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. . . . .	1
2	Create a new file with 0666 access permissions and enable the close-on-exec flag. . . . .	4
3	Change the file control information while setting O_SYNC flag . . . . .	5
4	Write a C Program to implement a UNIX 'ls -ls file1' command using File related API & stat structure. . . . .	6
5	Write a C Program to implement a UNIX 'ls -ls dir1' command using directory related system calls . . . . .	11
6	Write a C program which creates a child process and the parent waits for child's exit. . . . .	15
7	Write a C program to demonstrate the difference between the fork and vfork system calls. . . . .	18
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. . . . .	20
9	Implement 'ls   wc -l -c -w' command using pipe and exec functions. . . . .	23
10	Establish bidirectional communication between sender program and receiver program using multiple FIFO's. . . . .	25
11	Implement SVR based Message Queue IPC mechanism to establish asynchronous communication between two communicating processes. . . . .	31
12	Program to demonstrate IPC by implementing Server-Client Model to perform 'isEven' & 'isPrime' operations . . . . .	36
13	Implement Shared Memory based communication model . . . . .	45
14	Implement client/server model using socket API. . . . .	51
15	Implement concurrent server using fork based model while avoiding the zombie state of the client. . . . .	57
16	Implement a concurrent server model using pthread API. . . . .	62
17	Solve the producer consumer problem using pthread API. . . . .	67
18	Implement peer-to-peer communication model using socket API. . . . .	71
19	Solve the process synchronization on I/O using record locking mechanism. . . . .	76
20	Implement I/O multiplexing using select system call. . . . .	79

## 1 Implement 'cp' and 'mv' shell commands using file related system calls.

copy.c

```
1 // Copies contents of <source file> into <destination file>
2
3 #include <stdio.h>
4 #include <sys/types.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 #define BUF_SIZE 1024
9
10 int main(int argc, char *argv[]) {
11     if(argc != 3) {
12         printf("Invalid arguments.\nUsage: copy <source file> <destination file>\n");
13         return -1;
14     }
15
16     ssize_t retIn, retOut;
17     char buffer[BUF_SIZE];
18
19     int srcID = open(argv[1], O_RDONLY);
20     if(srcID == -1) {
21         printf("Unable to open %s\n", argv[1]);
22         return -1;
23     }
24
25     int destID = open(argv[2], O_WRONLY | O_CREAT, 0644);
26     if(destID == -1) {
27         printf("Unable to open %s\n", argv[2]);
```

```
28     return -1;
29 }
30
31 while((retIn = read(srcID, &buffer, BUF_SIZE)) > 0) {
32     retOut = write(destID, &buffer, retIn);
33     if(retOut != retIn) {
34         printf("Unable to copy files\n");
35         return -1;
36     }
37 }
38
39 close(srcID);
40 close(destID);
41
42 printf("Successfully copied %s to %s\n", argv[1], argv[2]);
43 return 0;
44 }
```

### Output

```
1 $ ./bin/copy source.txt dest.txt
2 Successfully copied source.txt to dest.txt
```

### move.c

```
1 // Moves <source file> into <destination file>
2
3 #include <stdio.h>
4 #include <sys/types.h>
5 #include <fcntl.h>
```

```
6  #include <unistd.h>
7
8  #define BUF_SIZE 1024
9
10 int main(int argc, char *argv[]) {
11     if(argc != 3) {
12         printf("Invalid arguments.\nUsage: move <source file> <destination file>\n");
13         return -1;
14     }
15
16     if((link(argv[1], argv[2])) == -1) {
17         printf("Unable to move file: %s\n", argv[1]);
18         return -1;
19     }
20     if((unlink(argv[1])) == -1) {
21         printf("Unable to move file: %s\n", argv[1]);
22         unlink(argv[2]);
23         return -1;
24     }
25
26     printf("Successfully moved %s to %s\n", argv[1], argv[2]);
27     return 0;
28 }
```

### Output

```
1  $ ./bin/move source.txt src.txt
2  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

```
1 // Creates a file named "close-on-exec.txt" with permissions 666 and close-on-exec flag set.
2
3 #include <stdio.h>
4 #include <sys/types.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 int main(void) {
9     int fd;
10    if((fd = open("close-on-exec.txt", O_WRONLY|O_CREAT, 0666)) == -1) {
11        printf("Unable to open the file.");
12        return -1;
13    }
14
15    int old_flags = fcntl(fd, F_GETFD);
16    printf("Old flags: %d\n", old_flags);
17
18    fcntl(fd, F_SETFD, FD_CLOEXEC);
19
20    int new_flags = fcntl(fd, F_GETFD);
21    printf("New flags: %d\n", new_flags);
22
23    return 0;
24 }
```

## Output

```
1 $ ls
2 bin close_on_exec.c
3
4 $ ./bin/close_on_exec
5 Old flags: 0
6 New flags: 1
7
8 $ ls
9 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

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <dirent.h>
4  #include <fcntl.h>
5  #include <unistd.h>
6  #include <sys/types.h>
7  #include <sys/stat.h>
8  #include <pwd.h>
9  #include <grp.h>
10 #include <time.h>
11
12 typedef struct {
13     long int block_no;
14     char type;
15     char *perms;
16     long int nlink;
17     char *uname;
18     char *gname;
19     long int size;
20     char *time_str;
21     char *name;
22 } file_info;
23
24 int getFileInfo(file_info*, char*);
25 int getFileInfoAt(file_info*, char*, int);
26
```



```
27 int getFileInfoAt(file_info *info, char* pathname, int dirfd) {
28     struct stat st;
29     struct passwd *pwd;
30     struct group *grp;
31
32     char type;
33     if(fstatat(dirfd, pathname, &st, 0) < 0) {
34         return -1;
35     }
36
37     char *time_str = (char*) calloc(20, sizeof(char));
38     char *perms = (char*) calloc(10, sizeof(char));
39
40     pwd = getpwuid(st.st_uid);
41     grp = getgrgid(st.st_gid);
42
43     long int block_no = st.st_blocks / 2;
44     strftime(time_str, 20, "%b %d %H:%M", localtime(&st.st_mtime));
45
46     if(S_ISREG(st.st_mode))
47         type = '-';
48     else if(S_ISDIR(st.st_mode)) {
49         type = 'd';
50         block_no = 0;
51     } else if(S_ISCHR(st.st_mode))
52         type = 'c';
53     else if(S_ISBLK(st.st_mode))
54         type = 'b';
55     else if(S_ISFIFO(st.st_mode))
56         type = 'p';
```

```
57     else if(S_ISLNK(st.st_mode))
58         type = 'l';
59     else if(S_ISSOCK(st.st_mode))
60         type = 's';
61
62     perms[0] = (S_IRUSR & st.st_mode) ? 'r' : '-';
63     perms[1] = (S_IWUSR & st.st_mode) ? 'w' : '-';
64     perms[2] = (S_IXUSR & st.st_mode) ? 'x' : '-';
65     perms[3] = (S_IRGRP & st.st_mode) ? 'r' : '-';
66     perms[4] = (S_IWGRP & st.st_mode) ? 'w' : '-';
67     perms[5] = (S_IXGRP & st.st_mode) ? 'x' : '-';
68     perms[6] = (S_IROTH & st.st_mode) ? 'r' : '-';
69     perms[7] = (S_IWOTH & st.st_mode) ? 'w' : '-';
70     perms[8] = (S_IXOTH & st.st_mode) ? 'x' : '-';
71     perms[9] = '\\0';
72
73     info->block_no = block_no;
74     info->type = type;
75     info->perms = perms;
76     info->nlink = st.st_nlink;
77     info->uname = pwd->pw_name;
78     info->gname = grp->gr_name;
79     info->size = st.st_size;
80     info->time_str = time_str;
81     info->name = pathname;
82
83     return 0;
84 }
85
86 int getFileInfo(file_info *info, char* pathname) {
```

```
87     return getFileInfoAt(info, pathname, AT_FDCWD);
88 }
```

ls\_file.c

```
1  // Displays file info like 'ls -ls'
2
3  #include <stdio.h>
4  #include <math.h>
5  #include "ls_helper.h"
6
7  #define MAX(a,b) (((a)>(b))?(a):(b))
8
9  int main(int argc, char *argv[]) {
10     if(argc <= 1) {
11         printf("Invalid arguments.\nUsage: ls_file <file> [<file> <file>]\n");
12         return -1;
13     }
14
15     // Variables for formatting
16     int p_blockno = 0;
17     int p_nlink = 0;
18     int p_size = 0;
19     file_info infos[argc-1];
20
21     for(int i = 1; i < argc; i++) {
22         if(getFileInfo(&infos[i-1], argv[i]) < 0) {
23             printf("Unable to retrieve File Details");
24             return -1;
25         }
26     }
```

```
26     p_blockno = MAX((int)log10(infos[i-1].block_no)+1, p_blockno);
27     p_nlink = MAX((int)log10(infos[i-1].nlink)+1, p_nlink);
28     p_size = MAX((int)log10(infos[i-1].size)+1, p_size);
29 }
30
31
32 for(int i = 0; i < (argc-1); i++) {
33     printf("%*ld %c%s %*ld %s %s %*ld %s %s\n", p_blockno, infos[i].block_no,
34         infos[i].type, infos[i].perms, p_nlink, infos[i].nlink, infos[i].uname,
35         infos[i].gname, p_size, infos[i].size, infos[i].time_str, infos[i].name);
36 }
37 }
```

### Output

```
1 $ ls -ls copy.c
2 4 -rw-r--r-- 1 ubuntu root 1032 Jan  3 18:29 copy.c
3
4 $ ./bin/ls_file copy.c
5 4 -rw-r--r-- 1 ubuntu root 1032 Jan 03 18:29 copy.c
6
7 $ ls -ls ls_file.c ls_dir.c
8 4 -rw-r--r-- 1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
9 4 -rw-r--r-- 1 ubuntu root 1140 Dec 22 14:20 ls_file.c
10
11 $ ./bin/ls_file ls_file.c ls_dir.c
12 4 -rw-r--r-- 1 ubuntu root 1140 Dec 22 14:20 ls_file.c
13 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

```
1 // Displays directory info like 'ls -ls'
2
3 #include <stdio.h>
4 #include <math.h>
5 #include "ls_helper.h"
6
7 #define true (1)
8 #define false (0)
9 #define MAX_ENTRIES 1024
10 #define MIN(a,b) (((a)<(b))?(a):(b))
11 #define MAX(a,b) (((a)>(b))?(a):(b))
12
13 int main(int argc, char *argv[]) {
14     if(argc != 2) {
15         printf("Invalid arguments.\nUsage: ls_dir <dir> \n");
16         exit(1);
17     }
18
19     // Variables for formatting
20     int p_blockno = 0;
21     int p_nlink = 0;
22     int p_size = 0;
23
24     // Variables
```

```
25  int len = 0, total_blocks = 0;
26  int dirfd = -1;
27  DIR *dir = NULL;
28  struct dirent *dirent;
29  file_info infos[MAX_ENTRIES];
30
31  if((dir = opendir(argv[1])) == NULL) {
32      printf("Unable to open %s.\n", argv[1]);
33      exit(1);
34  }
35
36  if((dirfd = open(argv[1], O_RDONLY)) < 0) {
37      printf("Unable to open %s.\n", argv[1]);
38      closedir(dir);
39      exit(1);
40  }
41
42  while(true) {
43      if((dirent = readdir(dir)) == NULL)
44          break;
45
46      if(dirent->d_name[0] == '.')
47          continue;
48
49
50      if(getFileInfoAt(&infos[len], dirent->d_name, dirfd) < 0) {
51          printf("Unable to retrieve File Details");
52          return -1;
53      }
54  }
```

```
55     total_blocks += infos[len].block_no;
56     p_blockno = MAX((int)log10(infos[len].block_no)+1, p_blockno);
57     p_nlink = MAX((int)log10(infos[len].nlink)+1, p_nlink);
58     p_size = MAX((int)log10(infos[len].size)+1, p_size);
59     len++;
60 }
61
62 printf("total %d\n", total_blocks);
63 for(int i = 0; i < len; i++) {
64     printf("%*ld %c%s %*ld %s %s %*ld %s %s\n", p_blockno, infos[i].block_no,
65         infos[i].type, infos[i].perms, p_nlink, infos[i].nlink, infos[i].uname,
66         infos[i].gname, p_size, infos[i].size, infos[i].time_str, infos[i].name);
67 }
68
69 if(dirfd != -1) {
70     close(dirfd);
71 }
72 if(dir != NULL) {
73     closedir(dir);
74 }
75
76 return 0;
77 }
```

## Output

```
1 $ ./bin/ls_dir .
2 total 80
3 0 drwxr-xr-x 25 ubuntu root 800 Jan 03 18:33 bin
4 4 -rw-r--r-- 1 ubuntu root 574 Dec 22 16:05 close_on_exec.c
5 4 -rw-r--r-- 1 ubuntu root 1032 Jan 03 18:29 copy.c
6 4 -rw-r--r-- 1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
7 4 -rw-r--r-- 1 ubuntu root 1140 Dec 22 14:20 ls_file.c
8 4 -rw-r--r-- 1 ubuntu root 2258 Dec 22 14:37 ls_helper.h
9 4 -rw-r--r-- 1 ubuntu root 666 Dec 22 16:05 move.c
10 4 -rw-r--r-- 1 ubuntu root 53 Dec 22 06:20 source.txt
11
12 $ls -ls .
13 total 80
14 0 drwxr-xr-x 25 ubuntu root 800 Jan 3 18:33 bin
15 4 -rw-r--r-- 1 ubuntu root 574 Dec 22 16:05 close_on_exec.c
16 4 -rw-r--r-- 1 ubuntu root 1032 Jan 3 18:29 copy.c
17 4 -rw-r--r-- 1 ubuntu root 1919 Dec 22 14:34 ls_dir.c
18 4 -rw-r--r-- 1 ubuntu root 1140 Dec 22 14:20 ls_file.c
19 4 -rw-r--r-- 1 ubuntu root 2258 Dec 22 14:37 ls_helper.h
20 4 -rw-r--r-- 1 ubuntu root 666 Dec 22 16:05 move.c
21 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

```
1 // Creates a child processes and the parent waits until its children exit.
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <math.h>
6 #include <unistd.h>
7 #include <sys/wait.h>
8
9 #define MAX_CHILDS 5
10
11 int main() {
12     for(int i = 0; i < MAX_CHILDS; i++)
13         if(!fork()) {
14             printf("Parent(pid) %d -> Child(pid) %d:\t", getppid(), getpid());
15             srand(getpid());
16
17             int n = 100;
18             long sum = 0;
19             for(int i = 0; i < n; i++) {
20                 sum += rand();
21             }
22             printf("Sum of %d random numbers = %ld\n", n, sum);
23
24             exit(0);
25         }
26
27     for(int i = 0; i < MAX_CHILDS; i++) {
```

```
28     int wstatus;
29
30     int cid = wait(&wstatus);
31
32     if(WIFEXITED(wstatus))
33         printf("\033[0;32m> child(pid): %d of parent (pid): %d exited normally with status: "
34               "%d\033[0m\n", cid, getpid(), WEXITSTATUS(wstatus));
35     else
36         printf("\033[0;31m> child(pid): %d of parent (pid): %d exited abnormally\033[0m\n",
37               cid, getpid());
38 }
39
40 return 0;
41 }
```

## Output

```
1 $ ./bin/parent_child_wait
2 parent(pid) 193 -> Child(pid) 194:      Sum of 100 random numbers = 112349801883
3 parent(pid) 193 -> Child(pid) 195:      Sum of 100 random numbers = 110287567662
4 parent(pid) 193 -> Child(pid) 196:      Sum of 100 random numbers = 117306909504
5 parent(pid) 193 -> Child(pid) 197:      Sum of 100 random numbers = 100113144996
6 > child(pid): 194 of parent (pid): 193 exited normally with status: 0
7 > child(pid): 195 of parent (pid): 193 exited normally with status: 0
8 parent(pid) 193 -> Child(pid) 198:      Sum of 100 random numbers = 99795688877
9 > child(pid): 196 of parent (pid): 193 exited normally with status: 0
10 > child(pid): 197 of parent (pid): 193 exited normally with status: 0
11 > child(pid): 198 of parent (pid): 193 exited normally with status: 0
12
13 $ ./bin/parent_child_wait
14 parent(pid) 199 -> Child(pid) 200:      Sum of 100 random numbers = 97012711868
15 parent(pid) 199 -> Child(pid) 201:      Sum of 100 random numbers = 113382333867
16 > child(pid): 200 of parent (pid): 199 exited normally with status: 0
17 > child(pid): 201 of parent (pid): 199 exited normally with status: 0
18 parent(pid) 199 -> Child(pid) 202:      Sum of 100 random numbers = 108751118111
19 parent(pid) 199 -> Child(pid) 203:      Sum of 100 random numbers = 117941568918
20 parent(pid) 199 -> Child(pid) 204:      Sum of 100 random numbers = 102851697320
21 > child(pid): 202 of parent (pid): 199 exited normally with status: 0
22 > child(pid): 203 of parent (pid): 199 exited normally with status: 0
23 > 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

```
1  // Demonstrates the difference between the fork and vfork system calls.
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/wait.h>
6  #include <unistd.h>
7
8  void forktest();
9  void vforktest();
10
11 int main() {
12     forktest();
13     vforktest();
14
15     return 0;
16 }
17
18 void forktest() {
19     int a = 3, b = 2;
20
21     if(!fork()) {
22         a++;
23         b++;
24
25         printf("fork child    -> a: %d; b: %d\n", a, b);
26         exit(0);
27     }
```

```
28
29     int cid = wait(NULL);
30     printf("fork parent -> a: %d; b: %d\n", a, b);
31 }
32
33 void vforktest() {
34     int a = 3, b = 2;
35
36     if(!vfork()) {
37         a++;
38         b++;
39
40         printf("vfork child -> a: %d; b: %d\n", a, b);
41         // Make sure this is _exit() and not exit()
42         _exit(0);
43     }
44
45     printf("vfork parent -> a: %d; b: %d\n", a, b);
46 }
```

### Output

```
1 $ ./bin/fork_vfork
2 fork child -> a: 4; b: 3
3 fork parent -> a: 3; b: 2
4 vfork child -> a: 4; b: 3
5 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

```
1 // Creates a child process and registers a signal handler to get exit status of child process
2 // asynchronously
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <signal.h>
7 #include <unistd.h>
8 #include <sys/types.h>
9 #include <sys/wait.h>
10 #include <time.h>
11
12 #define true (1)
13 #define false (0)
14
15 void signalHandler(int);
16
17 int stop = false;
18
19 int main() {
20     srand(time(0));
21
22     if(!fork()) {
23         sleep(rand() % 15 + 1);
24         exit((rand() % 100));
25     }
26 }
```

```
27     signal(SIGCHLD, signalHandler);
28
29     int i = 1;
30     while (!stop) {
31         printf("\rWaiting for Child Exit Signal..... %d", i);
32         fflush(stdout);
33         sleep(1);
34         i++;
35     }
36
37     return 0;
38 }
39
40 void signalHandler(int signum) {
41     printf("\nChild Exit Signal Received.\n");
42
43     int wstatus;
44     int cid = wait(&wstatus);
45     if(WIFEXITED(wstatus))
46         printf("\033[0;32m> child(pid): %d of parent (pid): %d exited normally with status: "
47             "%d\033[0m\n", cid, getpid(), WEXITSTATUS(wstatus));
48     else
49         printf("\033[0;31m> child(pid): %d of parent (pid): %d exited abnormally\033[0m\n",
50             cid, getpid());
51     stop = true;
52 }
```

## Output

```
1 $ ./bin/async_child_wait
2 Waiting for Child Exit Signal.... 5s
3 Child Exit Signal Received.
4 > child(pid): 182 of parent (pid): 181 exited normally with status: 4
5
6 $ ./bin/async_child_wait
7 Waiting for Child Exit Signal.... 2s
8 Child Exit Signal Received.
9 > child(pid): 184 of parent (pid): 183 exited normally with status: 52
10
11 $ ./bin/async_child_wait
12 Waiting for Child Exit Signal.... 9s
13 Child Exit Signal Received.
14 > 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

```
1 // Implements `ls | wc -l -w -c` using pipe() and exec()
2
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <sys/types.h>
7 #include <sys/wait.h>
8
9 int main() {
10     int pipefd[2];
11
12     if(pipe(pipefd) < 0) {
13         printf("Unable to create pipe. Exiting...");
14         return -1;
15     }
16
17     if(!fork()) {
18         dup2(pipefd[1], STDOUT_FILENO);
19         close(pipefd[0]);
20         close(pipefd[1]);
21
22         execl("/usr/bin/ls", "ls", ".", NULL);
23         _exit(1);
24     }
25
26     dup2(pipefd[0], STDIN_FILENO);
27     close(pipefd[0]);
```

```
28     close(pipefd[1]);
29
30     execl("/usr/bin/wc", "wc", "-l", "-c", "-w", NULL);
31     wait(NULL);
32
33     return 0;
34 }
```

### Output

```
1 $ ./bin/pipe_exec
2      27      27      346
3
4 $ ls | wc -l -c -w
5      27      27      346
```

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

fifo\_server.c

```
1 // Bidirectional communication between sender and receiver programs using multiple FIFOs.
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <fcntl.h>
7 #include <unistd.h>
8 #include <sys/types.h>
9 #include <sys/stat.h>
10 #include <signal.h>
11
12 #define true (1)
13 #define false (0)
14 #define BUF_SIZE 1024
15
16 void cleanup(int);
17
18 int server_fd = -1, client_fd = -1;
19
20 int main() {
21     signal(SIGINT, cleanup);
22
23     char *serverPath = "/tmp/fifo_ex_server";
24     char *clientPath = "/tmp/fifo_ex_client";
25
26     char buffer_in[BUF_SIZE], buffer_out[BUF_SIZE];
```

```
27
28     if(mkfifo(serverPath, 0666) < 0) {
29         printf("Error creating FIFO file. Exiting...\n");
30         return -1;
31     }
32
33     printf("Server Program.\n\n");
34     while(true) {
35         printf("> ");
36         scanf("%1000[^\n]%%c", buffer_out);
37
38         if((server_fd = open(serverPath, O_WRONLY)) == -1) {
39             printf("Error opening FIFO file. Exiting...\n");
40             break;
41         }
42
43         write(server_fd, buffer_out, strlen(buffer_out)+1);
44         close(server_fd);
45         server_fd = -1;
46
47         if((client_fd = open(clientPath, O_RDONLY)) == -1) {
48             printf("Error opening FIFO file. Exiting...\n");
49             break;
50         }
51
52         read(client_fd, &buffer_in, BUF_SIZE);
53         printf("Client: %s\n", buffer_in);
54         close(client_fd);
55         client_fd = -1;
56     }
```

```
57
58     return 0;
59 }
60
61 void cleanup(int signum) {
62     printf("\nClosing FIFO Files and performing cleanup...\n");
63     if(server_fd != -1) close(server_fd);
64     if(client_fd != -1) close(client_fd);
65     exit(0);
66 }
```

fifo\_client.c

```
1  // Bidirectional communication between sender and receiver programs using multiple FIFOs.
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <fcntl.h>
7  #include <unistd.h>
8  #include <sys/types.h>
9  #include <sys/stat.h>
10 #include <signal.h>
11
12 #define true (1)
13 #define false (0)
14 #define BUF_SIZE 1024
15
16 void cleanup(int);
17
```

```
18 int server_fd = -1, client_fd = -1;
19
20 int main() {
21     signal(SIGINT, cleanup);
22
23     char *serverPath = "/tmp/fifo_ex_server";
24     char *clientPath = "/tmp/fifo_ex_client";
25
26     char buffer_in[BUF_SIZE], buffer_out[BUF_SIZE];
27
28     if(mkfifo(clientPath, 0666) < 0) {
29         printf("Error creating FIFO file. Exiting...\n");
30         return -1;
31     }
32
33     printf("Client Program.\n\n");
34     while(true) {
35         if((server_fd = open(serverPath, O_RDONLY)) == -1) {
36             printf("Error opening FIFO file. Exiting...\n");
37             break;
38         }
39
40         read(server_fd, &buffer_in, BUF_SIZE);
41         printf("Server: %s\n", buffer_in);
42         close(server_fd);
43         server_fd = -1;
44
45         printf("> ");
46         scanf("%1000[^\n]%*c", buffer_out);
47     }
```

```
48     if((client_fd = open(clientPath, O_WRONLY)) == -1) {
49         printf("Error opening FIFO file. Exiting...\n");
50         break;
51     }
52
53     write(client_fd, buffer_out, strlen(buffer_out)+1);
54     close(client_fd);
55     client_fd = -1;
56 }
57
58 return 0;
59 }
60
61 void cleanup(int signum) {
62     printf("\nClosing FIFO Files and performing cleanup...\n");
63     if(server_fd != -1) close(server_fd);
64     if(client_fd != -1) close(client_fd);
65     exit(0);
66 }
```

## Output - fifo\_server.c

```
1 $ ./bin/fifo_server
2 Server Program.
3
4 > Hello, World!
5 Client: Hello, Server!
6 > How are You?
7 Client: Im Fine
8 > Bye
9 Client: Bye
10 > ^C
11 Closing FIFO Files and performing cleanup...
```

## Output - fifo\_client.c

```
1 $ ./bin/fifo_client
2 Client Program.
3
4 Server: Hello, World!
5 > Hello, Server!
6 Server: How are You?
7 > Im Fine
8 Server: Bye
9 > Bye
10 ^C
11 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

```
1 // Creates a Message Queue and communicates with mq_client_sync.c
2
3 #include <stdio.h>
4 #include <string.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/msg.h>
8
9 #define true (1)
10 #define false (0)
11 #define BUF_SIZE (1024)
12
13 typedef struct {
14     long msg_type;
15     char msg_text[BUF_SIZE];
16 } message;
17
18 int main() {
19     key_t key;
20     int msg_id;
21
22     printf("Server Program.\nType \".exit\" to exit.\n\n");
23
24     if((key = ftok("mq_server_sync.c", 69)) < 0) {
25         printf("Error creating key using ftok(). Exiting...\n");
26         return -1;
```

```
27     }
28
29     if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
30         printf("Error getting message queue using msgget(). Exiting...\n");
31         return -1;
32     }
33
34     message msg;
35     while(true) {
36         printf("> ");
37         scanf("%1000[^\n]*c", msg.msg_text);
38         msg.msg_type = 1;
39         msgsnd(msg_id, &msg, BUF_SIZE, 0);
40         if(strcmp(msg.msg_text, ".exit") == 0) break;
41
42         msgrcv(msg_id, &msg, BUF_SIZE, 2, 0);
43         printf("Client: %s\n", msg.msg_text);
44         if(strcmp(msg.msg_text, ".exit") == 0) break;
45     }
46
47     msgctl(msg_id, IPC_RMID, NULL);
48     return 0;
49 }
```

mq\_client\_sync.c

```
1 // Creates a Message Queue and communicates with mq_server_sync.c
2
3 #include <stdio.h>
4 #include <string.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/msg.h>
8
9 #define true (1)
10 #define false (0)
11 #define BUF_SIZE (1024)
12
13 typedef struct {
14     long msg_type;
15     char msg_text[BUF_SIZE];
16 } message;
17
18 int main() {
19     key_t key;
20     int msg_id;
21
22     printf("Client Program.\nType \".exit\" to exit.\n\n");
23
24     if((key = ftok("mq_server_sync.c", 69)) < 0) {
25         printf("Error creating key using ftok(). Exiting...\n");
26         return -1;
27     }
28
29     if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
```

```
30     printf("Error getting message queue using msgget(). Exiting...\n");
31     return -1;
32 }
33
34 message msg;
35 while(true) {
36     msgrcv(msg_id, &msg, BUF_SIZE, 1, 0);
37     printf("Server: %s\n", msg.msg_text);
38     if(strcmp(msg.msg_text, ".exit") == 0) break;
39
40     printf("> ");
41     scanf("%1000[^\n]%*c", msg.msg_text);
42     msg.msg_type = 2;
43     msgsnd(msg_id, &msg, BUF_SIZE, 0);
44     if(strcmp(msg.msg_text, ".exit") == 0) break;
45 }
46
47 return 0;
48 }
```

## Output - mq\_server\_sync.c

```
1 $ ./bin/mq_server_sync
2 Server Program.
3 Type ".exit" to exit.
4
5 > Hello, World
6 Client: Hello, Server
7 > This is a Message
8 Client: 200 OK
9 > .exit
```

## Output - mq\_client\_sync.c

```
1 $ ./bin/mq_client_sync
2 Client Program.
3 Type ".exit" to exit.
4
5 Server: Hello, World
6 > Hello, Server
7 Server: This is a Message
8 > 200 OK
9 Server: .exit
```

```
1 $ ipcs -q
2
3 ----- Message Queues -----
4 key          msqid      owner      perms      used-bytes  messages
5 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 from 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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/msg.h>
7  #include <signal.h>
8
9  typedef struct {
10     long req_type;
11     int req_data;
12     int req_service;
13 } request;
14
15 typedef struct {
16     long res_type;
```

```
17     int res_data;  
18 } response;  
19  
20 #define REQ_SIZE (2 * sizeof(int))  
21 #define RES_SIZE (sizeof(int))  
22  
23 #define MSG_TYPE_REQ 1  
24 #define MSG_TYPE_RES_EVEN 2  
25 #define MSG_TYPE_RES_PRIME 3  
26  
27 #define REQ_SRV_EVEN 1  
28 #define REQ_SRV_PRIME 2
```

prog\_12/server.c

```
1 // Creates a Message Queue and acts as a Server providing "isPrime" & "isEven" services.  
2  
3 #include <math.h>  
4 #include "message.h"  
5  
6 void sigint(int);  
7 int isEven(int);  
8 int isPrime(int);  
9  
10 key_t key = -1;  
11 int msg_id = -1;  
12  
13 int main() {  
14     signal(SIGINT, sigint);  
15 }
```

```
16 printf("Server Program.\nPress Ctrl+C to exit.\n\n");
17
18 if((key = ftok("message.h", 69)) < 0) {
19     printf("Error creating key using ftok(). Exiting...\n");
20     return -1;
21 }
22
23 if((msg_id = msgget(key, 0666 | IPC_CREAT)) < 0) {
24     printf("Error getting message queue using msgget(). Exiting...\n");
25     return -1;
26 }
27
28 request req;
29 response res;
30
31 while(true) {
32     msgrcv(msg_id, &req, REQ_SIZE, MSG_TYPE_REQ, 0);
33
34     if(req.req_service == REQ_SRV_EVEN) {
35         printf("Received Request: isEven(%d)... ", req.req_data);
36
37         res.res_type = MSG_TYPE_RES_EVEN;
38         res.res_data = isEven(req.req_data);
39         msgsnd(msg_id, &res, RES_SIZE, 0);
40
41         printf("Processed.\n");
42     } else if(req.req_service == REQ_SRV_PRIME) {
43         printf("Received Request: isPrime(%d)... ", req.req_data);
44
45         res.res_type = MSG_TYPE_RES_PRIME;
```



```
46         res.res_data = isPrime(req.req_data);
47         msgsnd(msg_id, &res, RES_SIZE, 0);
48
49         printf("Processed.\n");
50     } else {
51         printf("Received Request: Unknown... Discarding.\n");
52     }
53 }
54
55 return 0;
56 }
57
58 void sigint(int signum) {
59     printf("\nDeleting message queue and Stopping Server.....\n");
60     if(msg_id != -1) msgctl(msg_id, IPC_RMID, NULL);
61     exit(0);
62 }
63
64 int isEven(int n) {
65     return n % 2 == 0;
66 }
67
68 int isPrime(int n) {
69     if(n < 2) return false;
70
71     int max = (int)sqrt(n);
72     for(int i = 2; i <= max; i++) {
73         if(n%i == 0) return false;
74     }
75 }
```

```
76     return true;
77 }
```

prog\_12/client\_is\_even.c

```
1  // Using Message Queue, created by server.c, seeks "isEven" service.
2
3  #include "message.h"
4
5  int main() {
6      key_t key = -1;
7      int msg_id = -1;
8
9      printf("Client (isEven) Program.\nPress Ctrl+C to exit.\n\n");
10
11     if((key = ftok("message.h", 69)) < 0) {
12         printf("Error creating key using ftok(). Exiting...\n");
13         return -1;
14     }
15
16     if((msg_id = msgget(key, 0666)) < 0) {
17         printf("Error getting message queue using msgget().\nTry running the Server first.\n"
18             "Exiting...\n");
19         return -1;
20     }
21
22     request req;
23     response res;
24     req.req_type = MSG_TYPE_REQ;
25 }
```

```
26 while(true) {
27     printf("Enter a Number: ");
28     scanf("%d", &req.req_data);
29     req.req_service = REQ_SRV_EVEN;
30     if(msgsnd(msg_id, &req, REQ_SIZE, 0) < 0) {
31         printf("Unable to send request to Server process.\n\n");
32         continue;
33     }
34
35     if(msgrcv(msg_id, &res, RES_SIZE, MSG_TYPE_RES_EVEN, 0) < 0) {
36         printf("Unable to receive response from Server process.\n\n");
37         continue;
38     }
39
40     if(res.res_data)
41         printf("%d is an even no.\n\n", req.req_data);
42     else
43         printf("%d is an odd no.\n\n", req.req_data);
44 }
45
46 return 0;
47 }
```

prog\_12/client\_is\_prime.c

```
1 // Using Message Queue, created by server.c, seeks "isPrime" service.
2
3 #include "message.h"
4
5 int main() {
```

```
6  key_t key = -1;
7  int msg_id = -1;
8
9  printf("Client (isPrime) Program.\nPress Ctrl+C to exit.\n\n");
10
11 if((key = ftok("message.h", 69)) < 0) {
12     printf("Error creating key using ftok(). Exiting...\n");
13     return -1;
14 }
15
16 if((msg_id = msgget(key, 0666)) < 0) {
17     printf("Error getting message queue using msgget().\nTry running the Server first.\n"
18           "Exiting...\n");
19     return -1;
20 }
21
22 request req;
23 response res;
24 req.req_type = MSG_TYPE_REQ;
25
26 while(true) {
27     printf("Enter a Number: ");
28     scanf("%d", &req.req_data);
29     req.req_service = REQ_SRV_PRIME;
30
31     if(msgsnd(msg_id, &req, REQ_SIZE, 0) < 0) {
32         printf("Unable to send request to Server process.\n\n");
33         continue;
34     }
35 }
```

```
36     if(msgrcv(msg_id, &res, RES_SIZE, MSG_TYPE_RES_PRIME, 0) < 0) {
37         printf("Unable to receive response from Server process.\n\n");
38         continue;
39     }
40     if(res.res_data)
41         printf("%d is a prime no.\n\n", req.req_data);
42     else
43         printf("%d is not a prime no.\n\n", req.req_data);
44 }
45
46 return 0;
47 }
```

Output - Process 1 (server.c)

```
1  $ ./bin/server
2  Server Program.
3  Press Ctrl+C to exit.
4
5  Received Request: isEven(5)... Processed.
6  Received Request: isEven(8)... Processed.
7  Received Request: isEven(0)... Processed.
8  Received Request: isPrime(5)... Processed.
9  Received Request: isPrime(25)... Processed.
10 Received Request: isPrime(-10)... Processed.
11 Received Request: isEven(-3)... Processed.
12 Received Request: isPrime(0)... Processed.
13 ^C
14 Deleting message queue and Stopping Server.....
```

## Output - Process 2 (client\_is\_prime.c)

```
1 $ ./bin/client_is_prime
2 Client (isPrime) Program.
3 Press Ctrl+C to exit.
4
5 Enter a Number: 5
6 5 is a prime no.
7
8 Enter a Number: 25
9 25 is not a prime no.
10
11 Enter a Number: -10
12 -10 is not a prime no.
13
14 Enter a Number: 0
15 0 is not a prime no.
16
17 Enter a Number: ^C
```

## Output - Process 3 (client\_is\_even.c)

```
1 $ ./bin/client_is_even
2 Client (isEven) Program.
3 Press Ctrl+C to exit.
4
5 Enter a Number: 5
6 5 is an odd no.
7
8 Enter a Number: 8
9 8 is an even no.
10
11 Enter a Number: 0
12 0 is an even no.
13
14 Enter a Number: -3
15 -3 is an odd no.
16
17 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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <signal.h>
4  #include <string.h>
5  #include <errno.h>
6  #include <sys/ipc.h>
7  #include <sys/shm.h>
8  #include <sys/sem.h>
9
10 #define MAX_COUNT 25
11
12 #define SEM_COUNT 2
13 #define SEM_READ 0
14 #define SEM_WRITE 1
15 #define SEMOP_WAIT -1
16 #define SEMOP_RELEASE 1
17
18 void initialize();
19
20 key_t key = -1;
21 int shm_id = -1, sem_id = -1;
22
```

```
23 void initialize() {
24     if((key = ftok("shm_helper.c", 69)) < 0) {
25         printf("Error creating key using ftok(): %s\n", strerror(errno));
26         exit(-1);
27     }
28
29     if((shm_id = shmget(key, sizeof(int), 0600 | IPC_CREAT)) < 0) {
30         printf("Error getting shm_id using shmget(): %s\n", strerror(errno));
31         exit(-1);
32     }
33
34     if((sem_id = semget(key, SEM_COUNT, 0600 | IPC_CREAT)) < 0) {
35         printf("Error getting sem_id using semget(): %s\n", strerror(errno));
36         exit(-1);
37     }
38 }
```

prog\_13/shm\_writer.c

```
1 // Gets a Shared Memory (creates if doesn't exists) and writes MAX_COUNT (defined in
2 // "shm_helper.h") random numbers and uses Semaphores for synchronization of shm access
3
4 #include "shm_helper.h"
5
6 void cleanup();
7
8 int *buf = NULL;
9
10 int main() {
11     struct sembuf sbuf;
```



```
12     sbuf.sem_flg = SEM_UNDO;
13
14     atexit(cleanup);
15     signal(SIGINT, exit);
16     initialize();
17
18     if((buf = (int*) shmat(shm_id, NULL, 0)) == NULL) {
19         printf("Error attaching shm using shmat(): %s\n", strerror(errno));
20         return -1;
21     }
22
23     printf("Writing %d Random Numbers.\n", MAX_COUNT);
24     for(int i = 0; i < MAX_COUNT; i++) {
25         sbuf.sem_num = SEM_READ;
26         sbuf.sem_op = SEMOP_WAIT;
27         semop(sem_id, &sbuf, 1);
28
29         *buf = rand();
30         printf("\033[0;32m Wrote: %3d\033[0m\n", *buf);
31
32         sbuf.sem_num = SEM_WRITE;
33         sbuf.sem_op = SEMOP_RELEASE;
34         semop(sem_id, &sbuf, 1);
35     }
36     printf("Done.....\n");
37
38     return 0;
39 }
40
41 void cleanup() {
```

```
42     if(buf != NULL) shmdt(buf);
43     if(shm_id != -1) shmctl(shm_id, IPC_RMID, NULL);
44     if(sem_id != -1) semctl(sem_id, 0, IPC_RMID);
45 }
```

prog\_13/shm\_reader.c

```
1 // Gets a Shared Memory (creates if doesn't exists) and reads MAX_COUNT (defined in
2 // "shm_helper.h") random numbers and uses Semaphores for synchronization of shm access
3
4 #include "shm_helper.h"
5
6 void cleanup();
7
8 int *buf = NULL;
9
10 int main() {
11     struct sembuf sbuf;
12     sbuf.sem_flg = SEM_UNDO;
13
14     atexit(cleanup);
15     signal(SIGINT, exit);
16     initialize();
17
18     if((buf = (int*) shmat(shm_id, NULL, 0)) == NULL) {
19         printf("Error attaching shm using shmat(): %s\n", strerror(errno));
20         return -1;
21     }
22
23     printf("Reading %d Random Numbers.\n", MAX_COUNT);
```

```
24     for(int i = 0; i < MAX_COUNT; i++) {
25         sbuf.sem_num = SEM_READ;
26         sbuf.sem_op = SEMOP_RELEASE;
27         semop(sem_id, &sbuf, 1);
28
29         sbuf.sem_num = SEM_WRITE;
30         sbuf.sem_op = SEMOP_WAIT;
31         semop(sem_id, &sbuf, 1);
32
33         printf("\033[0;31m Read: %3d\033[0m\n", *buf);
34     }
35     printf("Done.....\n");
36
37     return 0;
38 }
39
40 void cleanup() {
41     if(buf != NULL) shmdt(buf);
42 }
```

## Output - Writer Process (shm\_writer.c)

```
1 $ ./bin/shm_writer
2 Writing 25 Random Numbers.
3   Wrote: 1804289383
4   Wrote: 846930886
5   Wrote: 1681692777
6   Wrote: 1714636915
7   Wrote: 1957747793
8   Wrote: 424238335
9   Wrote: 719885386
10  Wrote: 1649760492
11  Wrote: 596516649
12  Wrote: 1189641421
13  Wrote: 1025202362
14  Wrote: 1350490027
15  Wrote: 783368690
16  Wrote: 1102520059
17  Wrote: 2044897763
18  Wrote: 1967513926
19  Wrote: 1365180540
20  Wrote: 1540383426
21  Wrote: 304089172
22  Wrote: 1303455736
23  Wrote: 35005211
24  Wrote: 521595368
25  Wrote: 294702567
26  Wrote: 1726956429
27  Wrote: 336465782
28 Done.....
```

## Output - Reader Process (shm\_reader.c)

```
1 $ ./bin/shm_reader
2 Reading 25 Random Numbers.
3   Read: 1804289383
4   Read: 846930886
5   Read: 1681692777
6   Read: 1714636915
7   Read: 1957747793
8   Read: 424238335
9   Read: 719885386
10  Read: 1649760492
11  Read: 596516649
12  Read: 1189641421
13  Read: 1025202362
14  Read: 1350490027
15  Read: 783368690
16  Read: 1102520059
17  Read: 2044897763
18  Read: 1967513926
19  Read: 1365180540
20  Read: 1540383426
21  Read: 304089172
22  Read: 1303455736
23  Read: 35005211
24  Read: 521595368
25  Read: 294702567
26  Read: 1726956429
27  Read: 336465782
28 Done.....
```

## 14 Implement client/server model using socket API.

prog\_14/server.c

```
1 // TCP Server using Linux sockets API
2
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <sys/socket.h>
6 #include <netinet/in.h>
7 #include <errno.h>
8 #include <string.h>
9
10 #define true (1)
11 #define false (0)
12 #define BACKLOG 16
13 #define BUF_SIZE 1024
14
15 int main() {
16     int tcp_socket = -1, conn_fd = -1;
17     char msg[BUF_SIZE];
18
19     printf("Server Program. (Simple Message Echo Server)\nPress Ctrl+C to exit.\n\n");
20
21     if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
22         printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
23         return -1;
24     }
25
26     struct sockaddr_in server_addr;
27     server_addr.sin_family = AF_INET;
```

```
28 server_addr.sin_port = htons(8080);
29 server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
30 int server_addr_size = sizeof(server_addr);
31
32 if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
33     printf("Unable to bind socket to server address: %s\n", strerror(errno));
34     return -1;
35 }
36
37 if(listen(tcp_socket, BACKLOG) < 0) {
38     printf("Unable to listen to socket: %s\n", strerror(errno));
39     return -1;
40 }
41
42 while(true) {
43     if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
44                          &server_addr_size)) < 0) {
45         printf("Unable to accept new connection: %s\n", strerror(errno));
46         break;
47     }
48
49     if(recv(conn_fd, msg, BUF_SIZE, 0) < 0) {
50         printf("Unable to read message: %s\n", strerror(errno));
51         break;
52     }
53
54     printf("Received: %s; Echoing back message... ", msg);
55     if(send(conn_fd, msg, BUF_SIZE, 0) < 0) {
56         printf("Unable to send message: %s\n", strerror(errno));
57         break;
58     }
59 }
```

```
58     }
59
60     printf("Message Sent.\n");
61     fflush(stdout);
62
63     if(conn_fd != -1) close(conn_fd);
64     conn_fd = -1;
65 }
66
67 return 0;
68 }
```

prog\_14/client.c

```
1  // TCP Client using Linux sockets API
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <signal.h>
6  #include <stdlib.h>
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <errno.h>
10 #include <string.h>
11
12 #define true (1)
13 #define false (0)
14 #define BACKLOG 16
15 #define BUF_SIZE 1024
16
```

```
17 int main() {
18     int tcp_socket = -1;
19     char msg[BUF_SIZE];
20
21     printf("Client Program.\nPress Ctrl+C to exit.\n\n");
22
23     struct sockaddr_in server_addr;
24     server_addr.sin_family = AF_INET;
25     server_addr.sin_port = htons(8080);
26     server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
27     int server_addr_size = sizeof(server_addr);
28
29     while(true) {
30         printf("> ");
31         scanf("%1000[^\n]%", msg);
32
33         if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
34             printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
35             return -1;
36         }
37
38         if(connect(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
39             printf("Unable to get new connection: %s\n", strerror(errno));
40             break;
41         }
42
43         if(send(tcp_socket, msg, BUF_SIZE, 0) < 0) {
44             printf("Unable to send message.: %s\n", strerror(errno));
45             break;
46         }
```



```
47
48     if(recv(tcp_socket, msg, BUF_SIZE, 0) < 0) {
49         printf("Unable to read message.: %s\n", strerror(errno));
50         break;
51     }
52
53     printf("Server: %s\n", msg);
54
55     if(tcp_socket != -1) close(tcp_socket);
56     tcp_socket = -1;
57 }
58
59 if(tcp_socket != -1) close(tcp_socket);
60 return 0;
61 }
```

## Output - Server (server.c)

```
1 $ ./bin/server
2 Server Program. (Simple Message Echo Server)
3 Press Ctrl+C to exit.
4
5 Received: Hello, Server; Echoing back message... Message Sent.
6 Received: Hello, Server!; Echoing back message... Message Sent.
7 Received: This is Client 1; Echoing back message... Message Sent.
8 Received: Good Bye; Echoing back message... Message Sent.
9 Received: This is Client 2; Echoing back message... Message Sent.
10 Received: Byeee; Echoing back message... Message Sent.
```

## Output - Client 1 (client.c)

```
1 $ ./bin/client
2 Client Program.
3 Press Ctrl+C to exit.
4
5 > Hello, Server
6 Server: Hello, Server
7 > This is Client 1
8 Server: This is Client 1
9 > Good Bye
10 Server: Good Bye
11 > ^C
```

## Output - Client 2 (client.c)

```
1 $ ./bin/client
2 Client Program.
3 Press Ctrl+C to exit.
4
5 > Hello, Server!
6 Server: Hello, Server!
7 > This is Client 2
8 Server: This is Client 2
9 > Byeee
10 Server: Byeee
11 > ^C
```

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

http\_server\_fork.c

```
1 // TCP Server that accepts requests and creates child process to handle requests.
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <signal.h>
7 #include <sys/socket.h>
8 #include <netinet/in.h>
9 #include <sys/wait.h>
10 #include <errno.h>
11 #include <string.h>
12
13 #define true (1)
14 #define false (0)
15 #define BACKLOG 16
16 #define REQ_BUF_SIZE 1024
17 #define RES_BUF_SIZE 2048
18
19 void sigchld(int);
20
21 int main() {
22     signal(SIGCHLD, sigchld);
23
24     int tcp_socket = -1, conn_fd = -1, on = 1;
25     char req[REQ_BUF_SIZE], res[RES_BUF_SIZE];
26
```

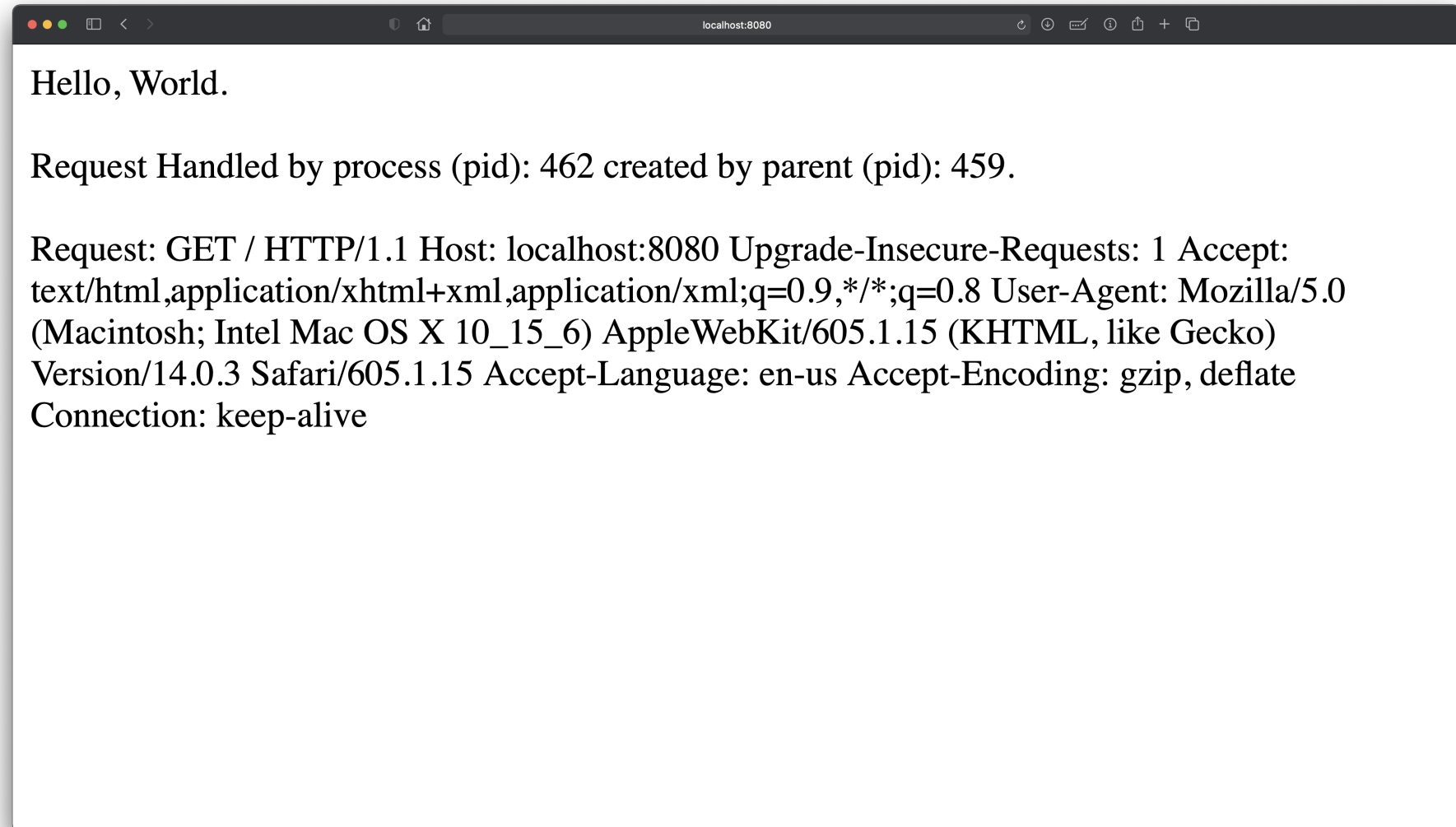
```
27 printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");
28
29 if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
30     printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
31     return -1;
32 }
33
34 setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
35
36 struct sockaddr_in server_addr;
37 server_addr.sin_family = AF_INET;
38 server_addr.sin_port = htons(8080);
39 server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
40 int server_addr_size = sizeof(server_addr);
41
42 if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
43     printf("Unable to bind socket to server address: %s\n", strerror(errno));
44     return -1;
45 }
46
47 if(listen(tcp_socket, BACKLOG) < 0) {
48     printf("Unable to listen to socket: %s\n", strerror(errno));
49     return -1;
50 }
51
52 while(true) {
53     if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
54                          &server_addr_size)) < 0) {
55         printf("Unable to accept new connection: %s\n", strerror(errno));
56         continue;
```

```
57     }
58
59     if(!fork()) {
60         close(tcp_socket);
61
62         if(recv(conn_fd, req, REQ_BUF_SIZE, 0) < 0) {
63             printf("Unable to read message: %s\n", strerror(errno));
64             close(conn_fd);
65             _exit(0);
66         }
67
68         printf("Received Request... ");
69
70         sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World.<br/><br/>\n"
71             "Request Handled by process (pid): %d created by parent (pid): %d.<br/><br/>\n"
72             "Request: %s", getpid(), getppid(), req);
73
74         if(send(conn_fd, res, strlen(res), 0) < 0) {
75             printf("Unable to send message: %s\n", strerror(errno));
76             close(conn_fd);
77             _exit(0);
78         }
79
80         close(conn_fd);
81         printf("Request Served. (cid: %d)\n", getpid());
82         _exit(0);
83     }
84
85     close(conn_fd);
86 }
```

```
87
88     return 0;
89 }
90
91 void sigchld(int signum) {
92     wait(NULL);
93 }
```

#### Terminal Output

```
1 $ ./bin/http_server_fork
2 Very Simple HTTP Web Server.
3 Press Ctrl+C to exit.
4
5 Received Request... Request Served. (cid: 462)
6 Received Request... Request Served. (cid: 463)
7 Received Request... Request Served. (cid: 464)
8 ^C
```



Output in 🍏 Safari

## 16 Implement a concurrent server model using pthread API.

http\_server\_pthread.c

```
1  // TCP Server that accepts requests and creates a thread using pthread to handle requests.
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <pthread.h>
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <sys/wait.h>
10 #include <errno.h>
11 #include <string.h>
12
13 #define true (1)
14 #define false (0)
15 #define BACKLOG 16
16 #define REQ_BUF_SIZE 1024
17 #define RES_BUF_SIZE 2048
18
19 void* handleRequest(void*);
20
21 int main() {
22     int tcp_socket = -1, conn_fd = -1, on = 1;
23
24     printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");
25
26     if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
27         printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
```



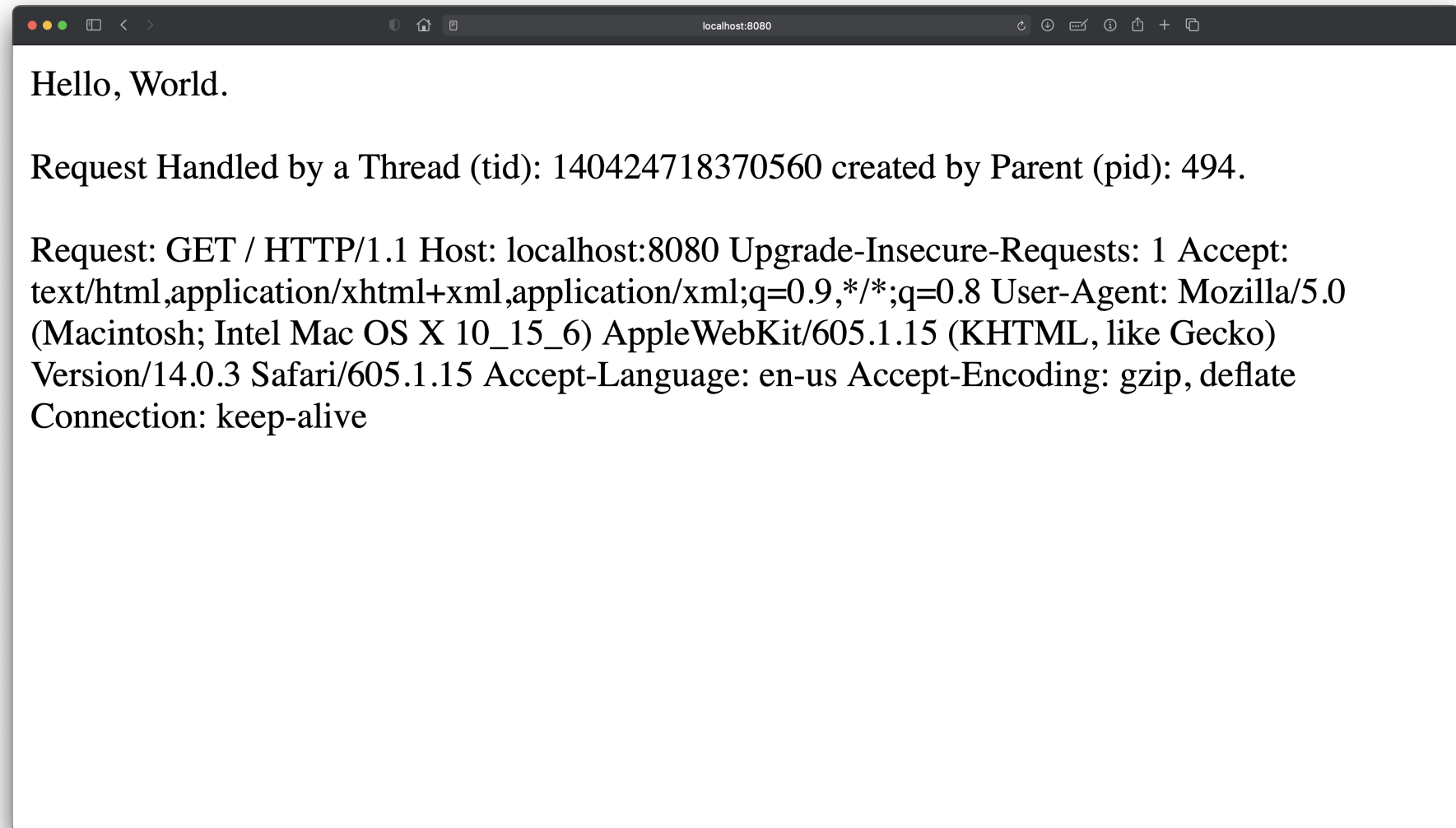
```
28     return -1;
29 }
30
31 setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
32
33 struct sockaddr_in server_addr;
34 server_addr.sin_family = AF_INET;
35 server_addr.sin_port = htons(8080);
36 server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
37 int server_addr_size = sizeof(server_addr);
38
39 if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
40     printf("Unable to bind socket to server address: %s\n", strerror(errno));
41     return -1;
42 }
43
44 if(listen(tcp_socket, BACKLOG) < 0) {
45     printf("Unable to listen to socket: %s\n", strerror(errno));
46     return -1;
47 }
48
49 while(true) {
50     if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
51                          &server_addr_size)) < 0) {
52         printf("Unable to accept new connection: %s\n", strerror(errno));
53         continue;
54     }
55
56     pthread_t tid;
57     int *new_conn_fd = (int*)malloc(sizeof(conn_fd));
```


```
58     *new_conn_fd = conn_fd;
59     if(pthread_create(&tid, NULL, handleRequest, (void*) new_conn_fd) != 0) {
60         printf("Unable to create new thread: %s\n", strerror(errno));
61         close(conn_fd);
62         free(new_conn_fd);
63         continue;
64     }
65
66     if(pthread_detach(tid) != 0) {
67         printf("Unable to detach new thread: %s\n", strerror(errno));
68         continue;
69     }
70 }
71
72 return 0;
73 }
74
75 void* handleRequest(void *new_conn_fd) {
76     int conn_fd = *((int*)new_conn_fd);
77     char req[REQ_BUF_SIZE], res[RES_BUF_SIZE];
78
79     if(recv(conn_fd, req, REQ_BUF_SIZE, 0) < 0) {
80         printf("Unable to read message: %s\n", strerror(errno));
81         close(conn_fd);
82         free(new_conn_fd);
83         pthread_exit(-1);
84     }
85
86     printf("Received Request... ");
87
```

```
88     sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World.<br/><br/>\n"
89             "Request Handled by a Thread (tid): %ld created by Parent (pid): %d.<br/><br/>\n"
90             "Request: %s", pthread_self(), getpid(), req);
91
92     if(send(conn_fd, res, strlen(res), 0) < 0) {
93         printf("Unable to send message: %s\n", strerror(errno));
94         close(conn_fd);
95         free(new_conn_fd);
96         pthread_exit(-1);
97     }
98
99     close(conn_fd);
100    printf("Request Served. (tid: %ld)\n", pthread_self());
101    free(new_conn_fd);
102    pthread_exit(0);
103 }
```

### Terminal Output

```
1 $ ./bin/http_server_pthread
2 Very Simple HTTP Web Server.
3 Press Ctrl+C to exit.
4
5 Received Request... Request Served. (tid: 140424718370560)
6 Received Request... Request Served. (tid: 140351225526016)
7 Received Request... Request Served. (tid: 140424709977856)
8 ^C
```



Output in  Safari

## 17 Solve the producer consumer problem using pthread API.

prod\_con\_thread.c

```
1 // Solves Producer - Consumer problem using pthreads and Mutex
2
3 #include <stdio.h>
4 #include <pthread.h>
5
6 #define BUF_SIZE 6
7 #define MAX_COUNT 42
8
9 int buf[BUF_SIZE];
10 int size = 0, front = 0, rear = 0;
11 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
12 pthread_cond_t m_produce = PTHREAD_COND_INITIALIZER;
13 pthread_cond_t m_consume = PTHREAD_COND_INITIALIZER;
14
15 void insert();
16 int delete();
17 void* producer(void*);
18 void* consumer(void*);
19
20 int main() {
21     pthread_t producer_tid, consumer_pid;
22
23     pthread_create(&producer_tid, NULL, producer, NULL);
24     pthread_create(&consumer_pid, NULL, consumer, NULL);
25
26     pthread_join(producer_tid, NULL);
27     pthread_join(consumer_pid, NULL);
28 }
```

```
28
29     return 0;
30 }
31
32 void insert(int data) {
33     buf[rear] = data;
34     rear = (rear + 1) % BUF_SIZE;
35     size++;
36 }
37
38 int delete() {
39     int data = buf[front];
40     front = (front + 1) % BUF_SIZE;
41     size--;
42     return data;
43 }
44
45 void* producer(void* arg) {
46     for(int i = 0; i < MAX_COUNT; i++) {
47         pthread_mutex_lock(&mutex);
48         while(size == BUF_SIZE) pthread_cond_wait(&m_produce, &mutex);
49         insert(i);
50         printf("\033[0;32m Produced: %3d\033[0m\n", i);
51         pthread_cond_signal(&m_consume);
52         pthread_mutex_unlock(&mutex);
53     }
54
55     return NULL;
56 }
57
```

```
58 void* consumer(void* arg) {
59     for(int i = 0; i < MAX_COUNT; i++) {
60         pthread_mutex_lock(&mutex);
61         while(size == 0) pthread_cond_wait(&m_consume, &mutex);
62         int data = delete();
63         printf("\033[0;31m Consumed: %3d\033[0m\n", data);
64         pthread_cond_signal(&m_produce);
65         pthread_mutex_unlock(&mutex);
66     }
67
68     return NULL;
69 }
```

```
1 $./bin/prod_con_pthread
2 Produced: 0
3 Produced: 1
4 Produced: 2
5 Produced: 3
6 Produced: 4
7 Produced: 5
8 Consumed: 0
9 Consumed: 1
10 Consumed: 2
11 Consumed: 3
12 Consumed: 4
13 Consumed: 5
14 Produced: 6
15 Produced: 7
16 Produced: 8
17 Produced: 9
18 Produced: 10
19 Produced: 11
20 Consumed: 6
21 Consumed: 7
22 Consumed: 8
23 Consumed: 9
24 Consumed: 10
25 Consumed: 11
26 Produced: 12
27 Produced: 13
28 Produced: 14
29 Produced: 15
```

## Output

```
1 Produced: 16
2 Produced: 17
3 Consumed: 12
4 Consumed: 13
5 Consumed: 14
6 Consumed: 15
7 Consumed: 16
8 Consumed: 17
9 Produced: 18
10 Produced: 19
11 Produced: 20
12 Produced: 21
13 Produced: 22
14 Produced: 23
15 Consumed: 18
16 Consumed: 19
17 Consumed: 20
18 Consumed: 21
19 Consumed: 22
20 Consumed: 23
21 Produced: 24
22 Produced: 25
23 Produced: 26
24 Produced: 27
25 Produced: 28
26 Produced: 29
27 Consumed: 24
```

```
1 Consumed: 25
2 Consumed: 26
3 Consumed: 27
4 Consumed: 28
5 Consumed: 29
6 Produced: 30
7 Produced: 31
8 Produced: 32
9 Produced: 33
10 Produced: 34
11 Produced: 35
12 Consumed: 30
13 Consumed: 31
14 Consumed: 32
15 Consumed: 33
16 Consumed: 34
17 Consumed: 35
18 Produced: 36
19 Produced: 37
20 Produced: 38
21 Produced: 39
22 Produced: 40
23 Produced: 41
24 Consumed: 36
25 Consumed: 37
26 Consumed: 38
27 Consumed: 39
28 Consumed: 40
29 Consumed: 41
```



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

prog\_18/server.c

```
1  // Peer-to-peer UDP Server using Linux sockets API
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <stdbool.h>
6  #include <unistd.h>
7  #include <sys/socket.h>
8  #include <netinet/udp.h>
9  #include <arpa/inet.h>
10 #include <signal.h>
11 #include <string.h>
12 #include <errno.h>
13
14 #define BUF_SIZE 1024
15
16 int udp_socket = -1;
17
18 void cleanup();
19
20 int main() {
21     atexit(cleanup);
22     signal(SIGINT, exit);
23
24     int on = 1;
25     char buf[BUF_SIZE];
26     struct sockaddr_in server, client;
27
```

```
28 printf("Server program.\nPress Ctrl+C to exit.\n\n");
29
30 if((udp_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
31     printf("Unable to get UDP Socket using socket(): %s\n", strerror(errno));
32     return -1;
33 }
34
35 setsockopt(udp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
36
37 server.sin_family = AF_INET;
38 server.sin_port = htons(8080);
39 inet_aton("127.0.0.1", &server.sin_addr);
40 int server_size = sizeof(server);
41 int client_size = sizeof(client);
42
43 if(bind(udp_socket, (struct sockaddr*) &server, server_size) < 0) {
44     printf("Unable to bind socket to server address: %s\n", strerror(errno));
45     return -1;
46 }
47
48 while(true) {
49     int buf_size = recvfrom(udp_socket, buf, BUF_SIZE, 0,
50                             (struct sockaddr*) &client, &client_size);
51
52     sendto(udp_socket, buf, buf_size, 0, (struct sockaddr*) &client, client_size);
53
54     printf("[Echoed message from %s:%d] > %s\n",
55           inet_ntoa(client.sin_addr), (int) (client.sin_port), buf);
56 }
57
```

```
58     return 0;
59 }
60
61 void cleanup() {
62     printf("\nShutting down server.....\n");
63     close(udp_socket);
64 }
```

prog\_18/client.c

```
1  // Peer-to-peer UDP Client using Linux sockets API
2
3  #include <stdio.h>
4  #include <stdbool.h>
5  #include <unistd.h>
6  #include <sys/socket.h>
7  #include <netinet/udp.h>
8  #include <arpa/inet.h>
9  #include <string.h>
10 #include <errno.h>
11
12 #define BUF_SIZE 1024
13
14 int main() {
15     int udp_socket = -1;
16     int on = 1;
17     char buf_out[BUF_SIZE], buf_in[BUF_SIZE];
18     struct sockaddr_in server, client;
19
20     printf("Client program.\n\n");
```

```
21
22     if((udp_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
23         printf("Unable to get UDP Socket using socket(): %s\n", strerror(errno));
24         return -1;
25     }
26
27     server.sin_family = AF_INET;
28     server.sin_port = htons(8080);
29     inet_aton("127.0.0.1", &server.sin_addr);
30     int server_size = sizeof(server);
31
32     printf("> ");
33     scanf("%1000[^\n]%*c", buf_out);
34
35     sendto(udp_socket, buf_out, strlen(buf_out)+1, 0, (struct sockaddr*) &server, server_size);
36     recvfrom(udp_socket, buf_in, BUF_SIZE, 0, NULL, NULL);
37
38     printf("\nSent: %s\n", buf_out);
39     printf("Echo: %s\n", buf_in);
40
41     close(udp_socket);
42     return 0;
43 }
```

## Output - server (server.c)

```
1 $ ./bin/server
2 Server program.
3 Press Ctrl+C to exit.
4
5 [Echoed message from 127.0.0.1:55222] > Hello, World!
6 [Echoed message from 127.0.0.1:35540] > Hello, Server!
```

## Output - Client 1 (client.c)

```
1 $ ./bin/client
2 Client program.
3
4 > Hello, World!
5
6 Sent: Hello, World!
7 Echo: Hello, World!
```

## Output - Client 2 (client.c)

```
1 $ ./bin/client
2 Client program.
3
4 > Hello, Server!
5
6 Sent: Hello, Server!
7 Echo: Hello, Server!
```

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

record\_locking.c

```
1 // Process Synchronization on I/O using Record Locking
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <errno.h>
7 #include <fcntl.h>
8 #include <unistd.h>
9
10 #define BUF_SIZE 64
11 #define MIN(x, y) (((x) < (y)) ? (x) : (y))
12
13 int main(int argc, char *argv[]) {
14     if(argc != 2) {
15         printf("Invalid arguments.\nUsage: %s <file>\n", argv[0]);
16         return -1;
17     }
18
19     struct flock fl = {F_UNLCK, SEEK_SET, 0, 100, 0};
20     int fsize, offset, fd, pid, read_size;
21     char buf[BUF_SIZE];
22
23     if((fd = open(argv[1], O_RDWR)) == -1) {
24         printf("Unable to open %s: %s\n", argv[1], strerror(errno));
25         return -1;
26     }
27 }
```

```
28 printf("Press any key to Lock the File\n");
29 printf("*****\n");
30 getchar();
31
32 fl.l_type = F_WRLCK;
33 fl.l_pid = getpid();
34 if(fcntl(fd, F_SETLK, &fl) == -1) {
35     printf("Cannot Set Exclusive Lock on %s: %s\n", argv[1], strerror(errno));
36     close(fd);
37     return -1;
38 } else if(fl.l_type != F_UNLCK && fl.l_type != F_RDLCK)
39     printf("%s has been Exclusively Locked by Process: %d\n", argv[1], fl.l_pid);
40 else
41     printf("%s is NOT Locked\n", argv[1]);
42
43
44 printf("Press any key to release the lock\n");
45 printf("*****\n");
46 getchar();
47
48 fl.l_type = F_UNLCK;
49 printf("File has been Unlocked \n");
50
51 fsize = lseek(fd, 0, SEEK_END);
52 offset = fsize - MIN(BUF_SIZE, fsize);
53 lseek(fd, offset, SEEK_SET);
54
55 read_size = read(fd, buf, MIN(BUF_SIZE, fsize));
56 buf[read_size] = '\0';
57 printf("Last %d bytes:\n*****\n%s\n", read_size, buf);
```

```

58
59     close(fd);
60     return 0;
61 }

```

### Output - Process 1 (record\_locking.c)

```

1  $ ./bin/record_locking record_locking.c
2  Press any key to Lock the File
3  *****
4
5  record_locking.c has been Exclusively Locked by
6  ↳ Process: 593
7  Press any key to release the lock
8  *****
9
10 File has been Unlocked
11 Last 64 bytes:
12 *****
13 **\n%s\n", read_size, buf);
14
15     close(fd);
16     return 0;
17 }

```

### Output - Process 2 (record\_locking.c)

```

1  $ ./bin/record_locking record_locking.c
2  Press any key to Lock the File
3  *****
4
5  Cannot Set Exclusive Lock on record_locking.c:
6  ↳ Resource temporarily unavailable

```



## 20 Implement I/O multiplexing using select system call.

http\_server\_select.c

```
1 // TCP Server that accepts requests and uses select() system call (I/O multiplexing) to
2 // handle requests.
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #include <signal.h>
8 #include <sys/select.h>
9 #include <sys/socket.h>
10 #include <netinet/in.h>
11 #include <sys/wait.h>
12 #include <time.h>
13 #include <errno.h>
14 #include <string.h>
15
16 #define true (1)
17 #define BACKLOG 16
18 #define BUF_SIZE 1024
19
20 int main() {
21     int tcp_socket = -1, conn_fd = -1, on = 1;
22     char req[BUF_SIZE], res[BUF_SIZE];
23     fd_set active_fd_set, read_fd_set;
24     struct timeval tm;
25
26     printf("Very Simple HTTP Web Server.\nPress Ctrl+C to exit.\n\n");
27     if((tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
```

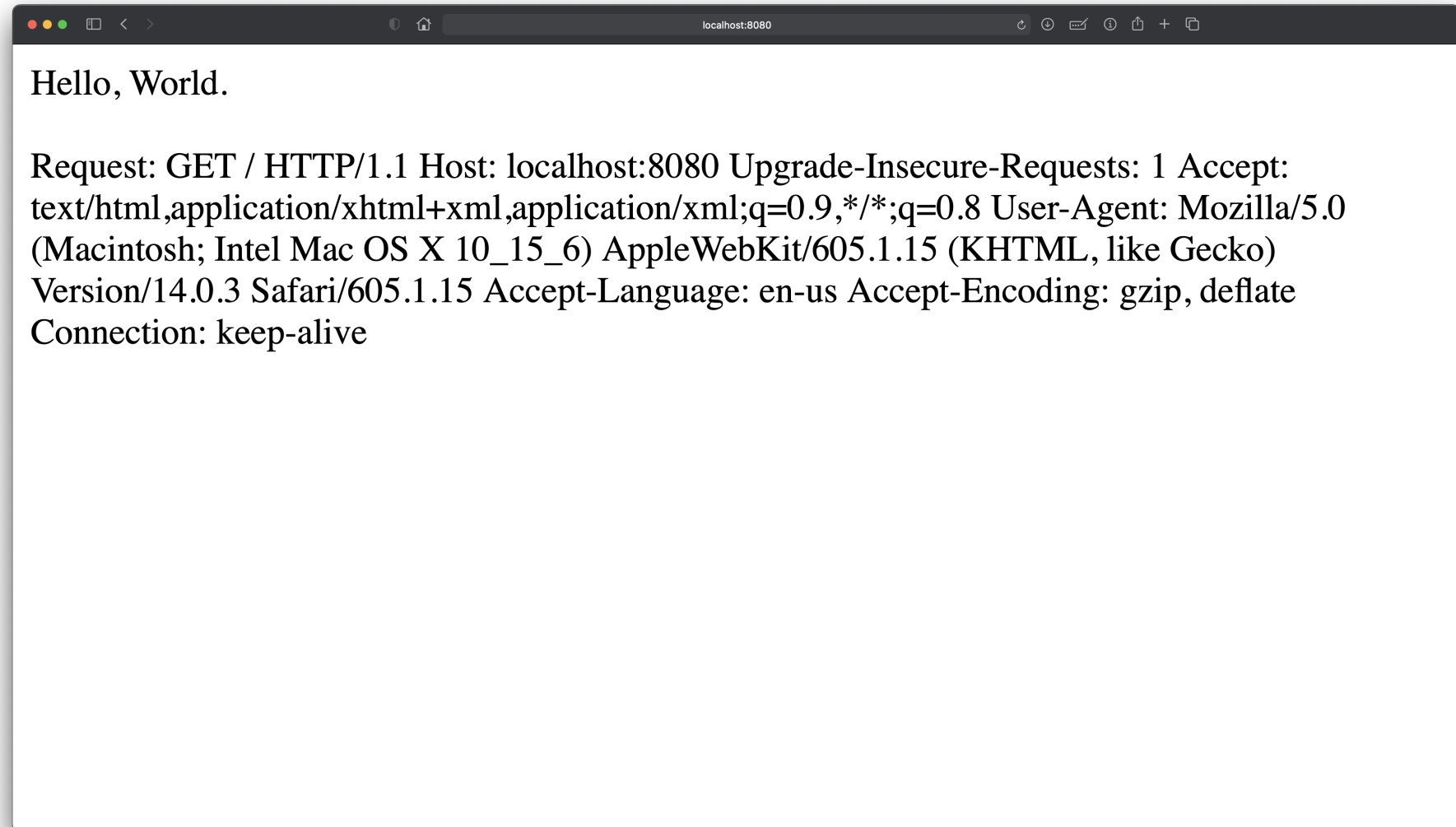
```
28     printf("Unable to get IPv4 TCP Socket using socket(): %s\n", strerror(errno));
29     return -1;
30 }
31
32 setsockopt(tcp_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
33 struct sockaddr_in server_addr;
34 server_addr.sin_family = AF_INET;
35 server_addr.sin_port = htons(8080);
36 server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
37 int server_addr_size = sizeof(server_addr);
38
39 if(bind(tcp_socket, (struct sockaddr*) &server_addr, server_addr_size) < 0) {
40     printf("Unable to bind socket to server address: %s\n", strerror(errno));
41     return -1;
42 }
43
44 if(listen(tcp_socket, BACKLOG) < 0) {
45     printf("Unable to listen to socket: %s\n", strerror(errno));
46     return -1;
47 }
48
49 FD_ZERO(&active_fd_set);
50 FD_SET(tcp_socket, &active_fd_set);
51
52 while(true) {
53     tm.tv_sec = 5; tm.tv_usec = 0;
54     read_fd_set = active_fd_set;
55     if(select(FD_SETSIZE, &read_fd_set, NULL, NULL, &tm) < 0) {
56         printf("Errorr with select(): %s\n", strerror(errno));
57         return -1;
```


```
58     }
59
60     for (int i = 0; i < FD_SETSIZE; i++) {
61         if (FD_ISSET(i, &read_fd_set)) {
62             if (i == tcp_socket) {
63                 if((conn_fd = accept(tcp_socket, (struct sockaddr*) &server_addr,
64                                     &server_addr_size)) < 0) {
65                     printf("Unable to accept new connection: %s\n", strerror(errno));
66                     continue;
67                 }
68
69                 printf("Connection Accepted... File Descriptor: %d\n", conn_fd);
70                 FD_SET(conn_fd, &active_fd_set);
71             } else {
72                 if(recv(i, req, REQ_BUF_SIZE, 0) < 0) {
73                     printf("Unable to read message: %s\n", strerror(errno));
74                     close(i);
75                     FD_CLR(i, &active_fd_set);
76                     continue;
77                 }
78
79                 printf("Received Request... ");
80                 sprintf(res, "HTTP/1.1 200 OK\nContent-Type: text/html\r\n\nHello, World."
81                         "<br/><br/>\nRequest: %s", req);
82
83                 if(send(i, res, strlen(res), 0) < 0) {
84                     printf("Unable to send message: %s\n", strerror(errno));
85                     close(i);
86                     FD_CLR(i, &active_fd_set);
87                     continue;
```

```
88         }
89         printf("Request Served. File Descriptor: %d\n", i);
90
91         close(i);
92         FD_CLR(i, &active_fd_set);
93     }
94 }
95 }
96 }
97
98 return 0;
99 }
```

### Terminal Output

```
1 $ ./bin/http_server_select
2 Very Simple HTTP Web Server.
3 Press Ctrl+C to exit.
4
5 Connection Accepted... File Descriptor: 4
6 Connection Accepted... File Descriptor: 5
7 Received Request... Request Served. File Descriptor: 5
8 Connection Accepted... File Descriptor: 5
9 Received Request... Request Served. File Descriptor: 4
10 Connection Accepted... File Descriptor: 4
11 Received Request... Request Served. File Descriptor: 5
12 Received Request... Request Served. File Descriptor: 4
```



Output in  Safari