

AUP Assignment 10

111703013 Akshay Rajesh Deodhar

12th November 2020

Q1

A pipe setup is given below that involves three processes. P is the parent process, and C1 and C2 are child processes, spawned from P. The pipes are named p1, p2, p3, and p4. Write a program that establishes the necessary pipe connections, setups, and carries out the reading/writing of the text in the indicated directions.

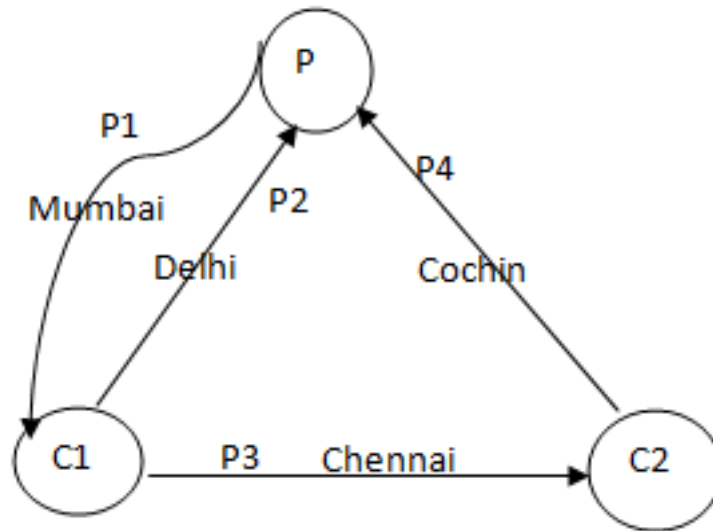


Figure 1: Setup of Pipes

```
1
2 #include <sys/types.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <errno.h>
6 #include <stdio.h>
7 #include <string.h>
8
9 #define M1 "Mumbai"
10 #define M2 "Delhi"
11 #define M3 "Chennai"
12 #define M4 "Cochin"
13
14 #define BUFLen 10
15
16 int main(void) {
17
```

```

18     int pid, ppid;
19
20     int pipes[5][2];
21
22     int n;
23     char buf[BUFLEN];
24
25     int i;
26
27     /* open 4 pipes */
28     for (i = 1; i <= 4; i++) {
29         if (pipe(pipes[i]) == -1) {
30             perror("pipe");
31             return errno;
32         }
33     }
34
35     if ((pid = fork()) == -1) {
36         perror("fork 1 failed");
37         return errno;
38     }
39     else if (!pid) {
40         /* C1 */
41
42         /* 1 W */
43         close(pipes[1][1]);
44
45         /* 2 R */
46         close(pipes[2][0]);
47
48         /* 3 R */
49         close(pipes[3][0]);
50
51         /* 4 RW */
52         close(pipes[4][0]);
53         close(pipes[4][1]);
54
55         pid = getpid();
56         ppid = getppid();
57
58         printf("%d is child of %d\n", pid, ppid);
59
60         if ((n = read(pipes[1][0], buf, BUFLen)) == -1) {
61             perror("read 1");
62             return errno;
63         }
64         printf("%d Read %s\n", pid, buf);
65         close(pipes[1][0]);
66
67         if (write(pipes[2][1], M2, strlen(M2) + 1) == -1) {
68             perror("write 2");
69             return errno;
70         }
71         printf("%d Wrote %s\n", pid, M2);

```

```

72         close(pipes[2][1]);
73
74         if (write(pipes[3][1], M3, strlen(M3) + 1) == -1) {
75             perror("write 3");
76             return errno;
77         }
78         printf("%d Wrote %s\n", pid, M3);
79         close(pipes[3][1]);
80
81         return 0;
82     }
83
84     if ((pid = fork()) == -1) {
85         perror("fork 2 failed");
86         return errno;
87     }
88     else if (!pid) {
89         /* C2 */
90
91         /* 1 RW */
92         close(pipes[1][0]);
93         close(pipes[1][1]);
94
95         /* 2 RW */
96         close(pipes[2][0]);
97         close(pipes[2][1]);
98
99         /* 3 W */
100        close(pipes[3][1]);
101
102        /* 4 R */
103        close(pipes[4][0]);
104
105        pid = getpid();
106        ppid = getppid();
107        printf("%d is child of %d\n", pid, ppid);
108
109        if ((n = read(pipes[3][0], buf, BUFLen)) == -1) {
110            perror("read 3");
111            return errno;
112        }
113        printf("%d Read %s\n", pid, buf);
114        close(pipes[3][0]);
115
116        if (write(pipes[4][1], M4, strlen(M4) + 1) == -1) {
117            perror("write 4");
118            return errno;
119        }
120        printf("%d Wrote %s\n", pid, M4);
121        close(pipes[4][1]);
122
123        return 0;
124    }

```

```

125      /* P */
126
127      /* close read end of 1 */
128      close(pipes[1][0]);
129
130      /* close write end of 2 */
131      close(pipes[2][1]);
132
133      /* close write end of 4 */
134      close(pipes[4][1]);
135
136      /* close both ends of 3 */
137      close(pipes[3][0]);
138      close(pipes[3][1]);
139
140      pid = getpid();
141      printf("%d is parent\n", pid);
142
143
144      if (write(pipes[1][1], M1, strlen(M1) + 1) == -1) {
145          perror("write 1");
146          return errno;
147      }
148      printf("%d Wrote %s\n", pid, M1);
149      close(pipes[1][1]);
150
151      if ((n = read(pipes[2][0], buf, BUFLLEN)) == -1) {
152          perror("read 2");
153          return errno;
154      }
155      printf("%d Read %s\n", pid, buf);
156      close(pipes[2][0]);
157
158      if ((n = read(pipes[4][0], buf, BUFLLEN)) == -1) {
159          perror("read 4");
160          return errno;
161      }
162      printf("%d Read %s\n", pid, buf);
163      close(pipes[4][0]);
164
165      return 0;
166  }
167

```

Output

```
akshay@akshay-inspiron5423 ~/.../lab/10_ipc master . /1
5841 is parent
5842 is child of 5841
5841 Wrote Mumbai
5842 Read Mumbai
5842 Wrote Delhi
5843 is child of 5841
5841 Read Delhi
5843 Read Chennai
5842 Wrote Chennai
5843 Wrote Cochin
5841 Read Cochin
akshay@akshay-inspiron5423 ~/.../lab/10_ipc master
```

Figure 2: Messages read and written by processes

Q2

Let P1 and P2 be two processes alternatively writing numbers from 1 to 100 to a file. Let P1 write odd numbers and p2, even. Implement the synchronization between the processes using FIFO.

```
1
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <stdio.h>
7 #include <errno.h>
8 #include <string.h>
9
10
11 #define FIFO1 "/tmp/aup_fifo1"
12 #define FIFO2 "/tmp/aup_fifo2"
13 #define FILENAME "/tmp/aup_file"
14 #define BUFLen 10
15
16 int main(void) {
17
18     int pid;
19     int fr, fw, fp;
20     int i;
21     char buf[BUFLen];
22
23     if (mkfifo(FIFO1, S_IRUSR | S_IWUSR) == -1) {
24         perror("mkfifo 1");
```

```

25         return errno;
26     }
27
28     if (mkfifo(FIFO2, S_IRUSR | S_IWUSR) == -1) {
29         perror("mkfifo 2");
30         return errno;
31     }
32
33     if ((fp = open(FILENAME, O_WRONLY | O_CREAT,
34         S_IRUSR | S_IWUSR)) == -1) {
35         perror("file");
36         return errno;
37     }
38
39     if ((pid = fork()) == -1) {
40         perror("fork");
41         return errno;
42     }
43     else if (pid) {
44         /* P1 */
45
46         if ((fw = open(FIFO1, O_WRONLY)) == -1) {
47             perror("write 1");
48             return errno;
49         }
50
51         if ((fr = open(FIFO2, O_RDONLY)) == -1) {
52             perror("read 2");
53             return errno;
54         }
55
56         i = 1;
57         while (i <= 100) {
58             sprintf(buf, "%d\n", i);
59
60             if (write(fp, buf, strlen(buf)) == -1) {
61                 perror("odd write");
62                 return errno;
63             }
64
65             if (write(fw, "*", 1) == -1) {
66                 perror("sync write odd");
67                 return errno;
68             }
69
70             if (read(fr, buf, 1) == -1) {
71                 perror("sync read odd");
72                 return errno;
73             }
74
75             i += 2;
76         }
77
78         close(fp);

```

```

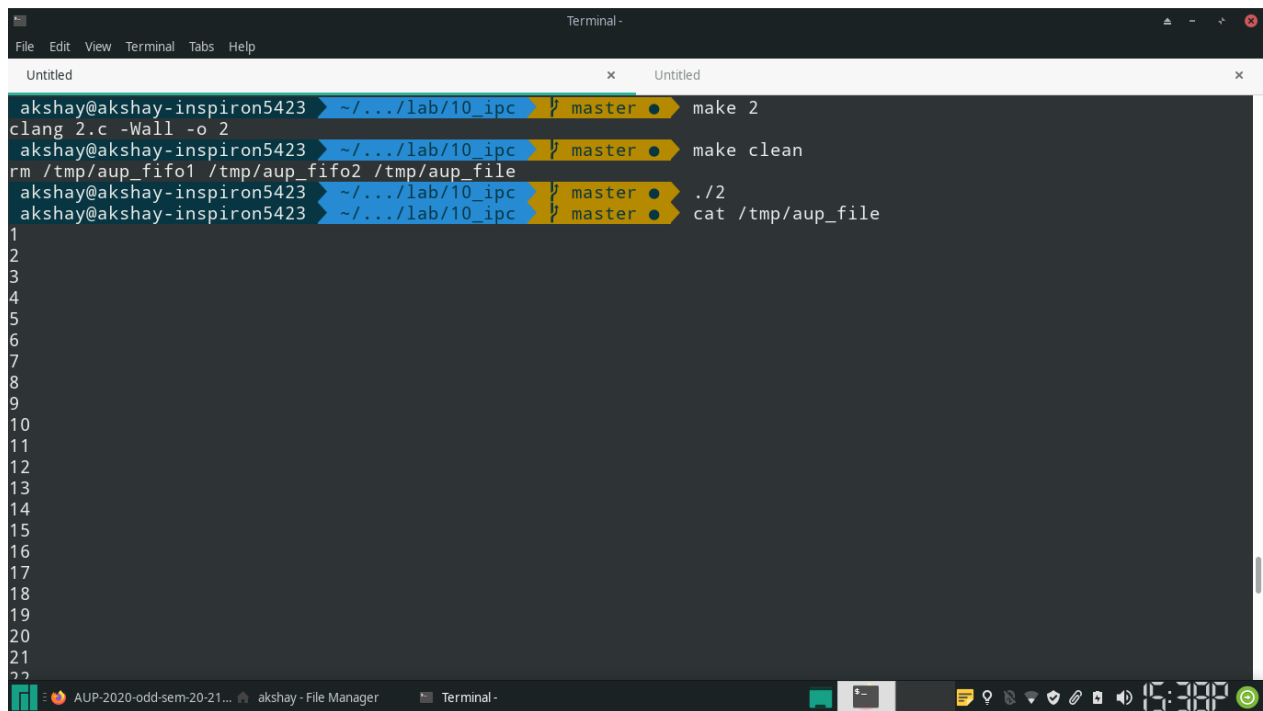
79         close(fw);
80         close(fr);
81
82         return 0;
83     }
84     else {
85         /* P1 */
86
87         if ((fr = open(FIFO1, O_RDONLY)) == -1) {
88             perror("read 1");
89             return errno;
90         }
91
92         if ((fw = open(FIFO2, O_WRONLY)) == -1) {
93             perror("write 2");
94             return errno;
95         }
96
97         i = 2;
98         while (i <= 100) {
99             if (read(fr, buf, 1) == -1) {
100                 perror("sync read even");
101                 return errno;
102             }
103
104             sprintf(buf, "%d\n", i);
105
106             if (write(fp, buf, strlen(buf)) == -1) {
107                 perror("odd write");
108                 return errno;
109             }
110
111             if (write(fw, "*", 1) == -1) {
112                 perror("sync write even");
113                 return errno;
114             }
115
116             i += 2;
117         }
118
119         close(fp);
120         close(fw);
121         close(fr);
122
123         return 0;
124     }
125 }

```

Note:

The program uses two FIFOs, /tmp/aup_fifo1 and /tmp/aup_fifo2. The file which is used for writing the numbers is /tmp/aup_file.

Output



A terminal window titled "Terminal -" with a menu bar (File, Edit, View, Terminal, Tabs, Help) and two tabs labeled "Untitled". The terminal shows a series of commands and their outputs, with line numbers 1 through 22 on the left. The commands are: `clang 2.c -Wall -o 2`, `make 2`, `make clean`, `rm /tmp/aup_fifo1 /tmp/aup_fifo2 /tmp/aup_file`, `./2`, and `cat /tmp/aup_file`. The output of `cat /tmp/aup_file` is a series of numbers 1 through 22, indicating synchronized writes to a shared file. The terminal window is part of a desktop environment with a taskbar at the bottom showing icons for "AUP-2020-odd-sem-20-21...", "akshay - File Manager", and "Terminal -". The system clock shows 15:38.

```
1 akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ● clang 2.c -Wall -o 2
2
3 akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ● make 2
4
5 akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ● make clean
6
7 rm /tmp/aup_fifo1 /tmp/aup_fifo2 /tmp/aup_file
8
9 akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ● ./2
10
11 akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ● cat /tmp/aup_file
12
13
14
15
16
17
18
19
20
21
22
```

Figure 3: Synchronized writes to shared file

Q3

Implement a producer-consumer setup using shared memory and semaphore. Ensure that data doesn't get over-written by the producer before the consumer reads and displays on the screen. Also ensure that the consumer doesn't read the same data twice.

Code

```
1
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <semaphore.h>
8  #include <errno.h>
9  #include <stdio.h>
10
11 #define BUF_SIZE 5
12 #define N_ITEMS 10
13
14
15 int main(void) {
16
17     int *buf;
18     sem_t *sem_fill;
19     sem_t *sem_empty;
20     int pid;
21     int i;
22
23     printf("Maximum number of elements in buffer: %d\n", BUF_SIZE);
24     printf("Number of items to be produced and consumed: %d\n", N_ITEMS);
25
26     if ((buf = (int *)mmap(NULL,
27                             BUF_SIZE * sizeof(int),
28                             PROT_READ | PROT_WRITE,
29                             MAP_SHARED | MAP_ANONYMOUS,
30                             -1,
31                             0)) == (void *)-1) {
32         perror("mmap 1");
33         return errno;
34     }
35
36     if ((sem_fill = (sem_t *)mmap(NULL,
37                                   sizeof(sem_t),
38                                   PROT_READ | PROT_WRITE,
39                                   MAP_SHARED | MAP_ANONYMOUS,
40                                   -1,
41                                   0)) == (void *)-1) {
42         perror("mmap fill");
43         return errno;
44     }
45
46     if (sem_init(sem_fill, 1, 0) == -1) {
```

```

47     perror("init fill");
48     return errno;
49 }
50
51 if ((sem_empty = (sem_t *)mmap(NULL,
52     sizeof(sem_t),
53     PROT_READ | PROT_WRITE,
54     MAP_SHARED | MAP_ANONYMOUS,
55     -1,
56     0)) == (void *)-1) {
57     perror("mmap 2");
58     return errno;
59 }
60
61 if (sem_init(&sem_empty, 1, BUF_SIZE) == -1) {
62     perror("init empty");
63     return errno;
64 }
65
66 if ((pid = fork()) == -1) {
67     perror("fork");
68     return errno;
69 }
70 else if (pid) {
71     /* parent, producer */
72
73     for (i = 0; i < N_ITEMS; i++) {
74         if (sem_wait(&sem_empty) == -1) {
75             perror("wait in producer");
76             return errno;
77         }
78
79         buf[i % BUF_SIZE] = i;
80         printf("Writing %d into buffer\n", i);
81
82         if (sem_post(&sem_fill) == -1) {
83             perror("post in producer");
84             return errno;
85         }
86     }
87 }
88 else {
89     /* child, consumer */
90
91     for (i = 0; i < N_ITEMS; i++) {
92
93         if (sem_wait(&sem_fill) == -1) {
94             perror("wait in consumer");
95             return errno;
96         }
97
98         printf("Read %d from buffer\n", buf[i % BUF_SIZE]);
99

```

```

100         if (sem_post(sem_empty) == -1) {
101             perror("post in consumer");
102             return errno;
103         }
104     }
105 }
106
107 return 0;
108 }

```

Note

There is 1 producer and 1 consumer, 10 items are sent through the shared memory in total, capacity of the shared memory is 5 items

Output

```

Terminal -
File Edit View Terminal Tabs Help
Untitled x Untitled x
akshay@akshay-inspiron5423 ~/.../lab/10_ipc master ./.3
Maximum number of elements in buffer: 5
Number of items to be produced and consumed: 10
Writing 0 into buffer
Writing 1 into buffer
Writing 2 into buffer
Writing 3 into buffer
Writing 4 into buffer
Read 0 from buffer
Read 1 from buffer
Read 2 from buffer
Read 3 from buffer
Read 4 from buffer
Writing 5 into buffer
Writing 6 into buffer
Writing 7 into buffer
Read 5 from buffer
Writing 8 into buffer
Read 6 from buffer
Writing 9 into buffer
Read 7 from buffer
Read 8 from buffer
Read 9 from buffer
akshay@akshay-inspiron5423 ~/.../lab/10_ipc master

```

Figure 4: Items written and read without deadlock