Operating Systems Design 19CS2106S
Session – 9 &10
ALM

1. Understand the internal algorithmsbehind the design of various xv6 file system calls. perform the given tests related to file system. Customize the usertests.c given in xv6 source code base and execute. submit the output of all the tests

OSD
ALM 9 & 10

190031187
Radhakrishna

```
1. (i) void openiputtest( void){
        printf( stdout, "iput test\n");
        if( mkdir( "iputdir" )<0){
            printf(stdout, mkdir iputdir failed \n");
        exit();
        }
        if(chdir( "iputdir" )<0){
            printf(stdout, "chdir iputdir failed\n");
            exit();
        }
        if(chdir("/")<0){
            printf(stdout, "chdir / failed\n");
            exit();
        }

        printf(stdout, "iput test ok \n");
    }
(ii) void opentest(void)
    { int fd;
        printf( stdout, "open test\n");
        fd= open( "echo", 0);
        if ( fd<0){
            printf(stdout, "open echo failed \n");
            exit();
        }
        close(fd);
        fd = open ( "doesnot exist", 0);
```

190031187

```
if (fd>=0){
printf(Stdout, "Opendoesnotexist succeded");
exit();
}
printf (stdout; "opentest ok \n");
}

(iii) void writetest (void)
{ int fd;
  int i;
  printf(stdout, "small file test \n");
fd=open("small", O_CREATE|O_RDWR);
if (fd>=0)
  printf (stdout, "create small succedded ok \n")
else {
  printf (stdout, "error create small failed \n")
  exit();
}
for (i=0; i<100; i++) {
  if (write(fd, "99999999", 10)!=10)
  { printf(stdout, "error: write 99 %d
                  new file failed\n", i);
    exit();
  }
}
printf (stdout, "writes ok\n");
close (fd);
```

190031187

```
fd = open("small", O_RDONLY);
if (fd >= 0)
    printf(stdout, "open small sucedded");
else {
    printf(stdout, "error: open small failed\n");
    exit();
}
i = read(fd, buf, 2000);
if (i == 2000)
    printf(stdout, "read succeded ok\n");
else {
    printf(stdout, "read failed\n");
    exit();
}
close(fd);
printf(stdout, "small file test ok\n");
```

osd-190031187@team-osd:~/xv6

```
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980


Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
190031187$ usertests
usertests starting
arg test passed
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
fourfiles ok
sharedfd test
sharedfd ok
```

2. Assume that a process A executes the following three function calls:

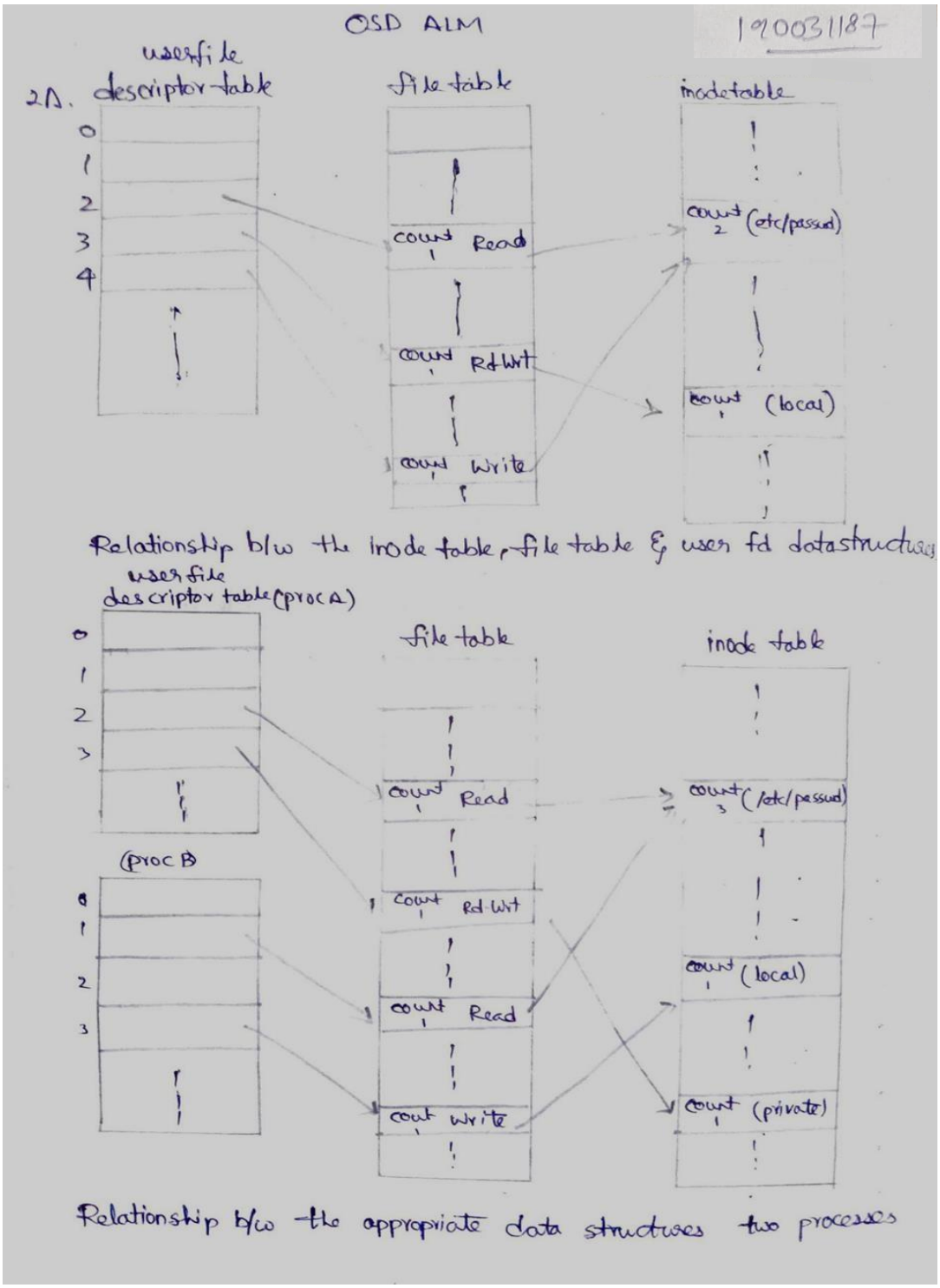
```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("local", O_RDWR);
fd3 = open("/etc/passwd", O_WRONLY);
```

For process A, show the relationship between the inode table, file table, and user filedescriptor data structures.

Suppose a secondprocess B executes the following code.

```
fd1 — open("/etc/passwd", O_RDONLY);
fd2 — open("private", O_RDONLY);
```

Draw the resulting picture that shows the relationship between the appropriate data structures whileboth processes (and no others) have the files open.



3. The following sequence of code has been observed in various programs:

```
dup2(fd, 0);
dup2(fd, 1);
dup2(fd, 2);
if (fd> 2)
close(fd);
```

To see why the `if` test is needed, assume that `fd`is 1 and draw a picture of what happens tothe three descriptor entries and the corresponding file table entry with each call to `dup2`.Then assume that `fd`is 3 and draw the same picture.

190031187

3.

If fd is 1, then dup2(fd,1) returns 1 without closing descriptor 1. After the three calls to dup2, all three descriptors point to same file table entry nothing needs to be closed

If fd is 3 however, after the three calls to dup2, four descriptors are point-ing to the same file table entry. In this case, we need to close descriptor 3.

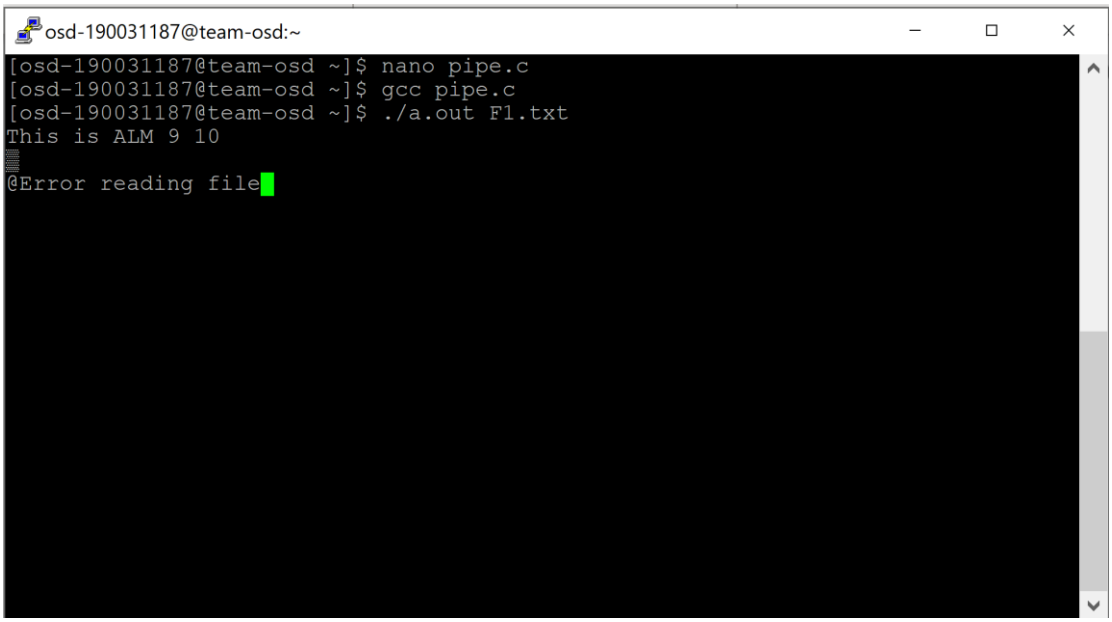
4. Write a program that create two pipes, send filename from command line to child process. In child read that file and send it back using pipe. Parent process should print the file. if error occur in child process error must be send to parent process.

```
4. #include <stdio.h>
   #include <unistd.h>
   #include <stdlib.h>
   #include <string.h>
   int main (int argc, char *argv[])
   {  int pipefd[2];
      char buff[100];
      int childpid;
      int size;
     File * file;
      if (argc != 2) {
          printf("usage:\n %s <filename>\n",
                                argv[0]);
```

```
exit (1);                                   190031187
}
if( pipe(pipefd)<0)
    perror("can't open pipe\n");
if ((childpid = fork()) == 0)
{
  sleep(1);
  size = read (pipefd[0], buff, sizeof(buff));
  file = fopen (buff, "r");
  if (file == NULL)
  {  write (pipefd[1], "Can't open file", 15);
     exit(1);
  }
  while (!feof (file)) {
     if ( fgets (buff, sizeof (buff), file) == NULL)
     {  write (pipfd[1], "Error reading file", 18);
     }
     else
        write (pipefd[1], buff, sizeof (buff));
  }

}
else if (childpid > 0) {
     size = strlen(argv[1]);
     if ( write( pipefd[1], argv[1], size) != size)
     {  perror( "Error writing to file\n");
     }
     wait (NULL);
```

```
     while(size = read (pipefd[0], buff, sizeof (buff))
     {   write (1, buff, size);
     }
  }
  exit (0);
}
```


osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano pipe.c
[osd-190031187@team-osd ~]$ gcc pipe.c
[osd-190031187@team-osd ~]$ ./a.out F1.txt
This is ALM 9 10
@Error reading file