Recitation 11: I/O Problems

Andrew Faulring 15213 Section A 18 November 2002

Logistics

- faulring@cs.cmu.edu
- Office hours
 - NSH 2504
 - Permanently moving to Tuesday 2–3
- What's left
 - Lab 6 Malloc: due on Thursday, 21 Nov
 - Lab 7 Proxy: due on Thursday, 5 Dec
 - Final Exam: 8:30am on Tuesday, 17 Dec, in Porter Hall 100

Today's Plan

- Robust I/O
- Chapter 11 Practice Problems

Why Use Robust I/O

- Handles interrupted system calls – Signal handlers
- Handles short counts
 - Encountering end-of-file (EOF) on reads (disk files)
 - Reading text lines from a terminal
 - Reading and writing network sockets or Unix pipes
- Useful in network programs
 - Subject to short counts
 - Internal buffering constraints
 - Long network delays
 - Unreliable

Rio: Unbuffered Input/Output

- · Transfer data directly between memory and a file
- No application level buffering
- Useful for reading/writing binary data to/from networks
 - (Though text strings are binary data.)

ssize_t rio_readn(int fd, void* usrbuf, size_t n)

- Reads n bytes from fd into usrbuf
- Only returns short on EOF

ssize_t rio_writen(int fd, void* usrbuf, size_t n)

- Writes n bytes from usrbuf to file fd
- Never returns short count

Rio: Buffered Input

void rio_readinitb(rio_t* rp, int fd);

- Called only once per open file descriptor
- Associates fd with a read buffer rp

ssize_t rio_readlineb(rio_t* rp, void* usrbuf, size_t maxlen)

- For reading lines from a text file only
- Read a line (stop on `\n') or maxlen-1 chars from file rp to usrbuf
- Terminate the text line with null (zero) character
- Returns number of chars read

ssize_t rio_readnb(rio_t* rp, void* usrbuf, size_t n);

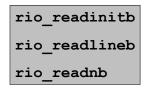
- For both text and binary files
- Reads n bytes from rp into usrbuf
- Result string is NOT null-terminated!
- Returns number of chars read

rio_readlineb

ssize t rio readlineb(rio t *rp, void *usrbuf, size t maxlen) £ int n, rc; char c, *bufp = usrbuf; for (n = 1; n < maxlen; n++) { if ((rc = rio read(rp, &c, 1)) == 1) { *bufp++ = c;if $(c == ' \setminus n')$ break; } else if (rc == 0) { if (n == 1)return 0; /* EOF, no data read */ else /* EOF, some data was read */ break; } else return -1; /* error */ *bufp = 0;return n;

Do not interleave

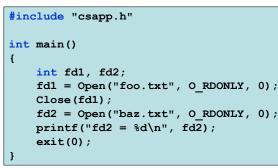
- Do not interleave calls on the same file descriptor to these two sets of functions
- Why?



rio_readn rio_writen

Rio Error Checking Problems from Chapter 11 • All functions have upper case equivalents • 11.1-11.5 (Rio readn ...), which call unix error if Handout contains the problems the function encounters an error Short reads are not errors - Also handles interrupted system calls - But does **not** ignore EPIPE errors, which are not fatal errors for Lab 7 Problem 11.1 Answer to 11.1 Default file descriptors: What is the output of the following

program?



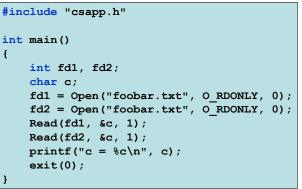
- stdin (descriptor 0)
- stdout (descriptor 1)
- stderr (descriptor 2)
- open always returns *lowest*, *unopened* descriptor
- First open returns 3. close frees it.
- So second open also returns 3.
- Program prints:
 - fd2 = 3

Kernel Structure for Open Files

- Descriptor table
 - One per process
 - Children inherit from parents
- File Table
 - The set of all open files
 - Shared by all processes
 - Reference count of number of file descriptors pointing to each entry
- V-node table
 - Contains information in the stat structure
 - Shared by all processes

Problem 11.2

Suppose that the disk file foobar.txt consists of the 6 ASCII characters "foobar". Then what is the output of the following program?



Answer to 11.2

- Two descriptors fd1 and fd2
- Two open file table entries, each with their own file positions for foobar.txt
- The read from fd2 also reads the first byte of foobar.txt
- So, the output is

c = f

and not

c = 0

Problem 11.3

As before, suppose the disk file foobar.txt consists of 6 ASCII characters "foobar". Then what is the output of the following program?

<pre>#include "csapp.h"</pre>
int main()
{
<pre>int fd;</pre>
char c;
<pre>fd = Open("foobar.txt", O_RDONLY, 0);</pre>
if(Fork() == 0) {
Read(fd, &c, 1);
exit(0);
}
Wait(NULL);
Read(fd, &c, 1);
printf("c = %c n", c);
<pre>exit(0);</pre>
}

Answer to 11.3

- Child inherits the parent's descriptor table.
- Child and parent share an open file table entry (refcount == 2).
- Hence they share a file position!
- The output is

c = 0

Problem 11.4

- How would you use dup2 to redirect standard input to descriptor 5?
- int dup2(int oldfd, int newfd);
 - Copies descriptor table entry oldfd to descriptor table entry newfd

Answer to 11.4

dup2(5,0);

or

dup2(5,STDIN_FILENO);

Problem 11.5

Assuming that the disk file foobar.txt consists of 6 ASCII characters "foobar". Then what is the output of the following program?

#include "csapp.h"
<pre>int main() { int fd1, fd2; char c; fd1 = Open("foobar.txt", O_RDONLY, 0); fd2 = Open("foobar.txt", O_RDONLY, 0); Read(fd2, &c, 1); Dup2(fd2, fd1); Read(fd1, &c, 1); printf("c = %c\n", c); exit(0); }</pre>
}

Answer to 11.5

• We are redirecting fd1 to fd2. So the second Read uses the file position offset of fd2.

c = 0