CS61 Section Notes Solutions

Section 11 (Fall 2011)

File IO
Fork Review
Signals

Answers:

Question 1: read() and write() are system calls, which have very high overhead, so reading and writing one byte at a time will be very slow!

Question 2: Every time the program writes back to stdout it writes the full 512 byte buffer, even if it didn't fill it with the call to read(). We could fix this by checking the return value of read() more carefully. Both versions should also check for an error—if the read was interrupted, we may want to try again rather than immediately exiting.

Question 3: By keeping track of the reference counts to an open file, the OS can determine when no more file descriptors point to a given open file. When that occurs, the OS can close the file. Just like garbage collection, the reference counting enables the automatic reclamation of unused resources.

Question 4: 2 entries are added to each of the file descriptor table and the open file table, but there is only one entry in the v-node table corresponding to the file.

Question 5: dup2(fd1, fd2) closes the existing filehandle at fd2, and points fd2 to the open file that is pointed to by fd1, increasing its reference count. If fd2 was the only reference to the open file it was pointing to, then that file is closed and the entry removed from the open file table. But no entries are added to any of the tables! Also, if the second argument did not exist it will open a new file descriptor pointing to the first argument.

Program 1 outputs “f”. There are two file descriptors and two open files.

Program 2 outputs “o”. There are two file descriptors and only one open file.

Question 6: Here is the process hierarchy:

```
    +--------------
    |   “3” “0”  *
    |          *
    |          *
+-------------------*-----------
```

```
    |        wait        “4”  “0”
    |
    |
-------------------
            “2”
```

So A, C, and D are valid outputs.

Question 7: Here is the process hierarchy:

```
        +--------------
        |     “0”  “2”
        |
   +--------------------
   |          “1”  “2”
   |
   |         +--------
   |         |    “0”
   |         |
-------------------------
                  “1”
```

So, A, C, and E are possible.

Question 8: One thing to note is that even if we “miss” a SIGCHLD, those children are still reachable. We saw in lecture that one way to deal with multiple children exiting is to loop over a call to waitpid until all the exited children have been reaped. Giving waitpid the argument pid = -1 allows the loop to reap children with arbitrary pids.

Question 9:

```
sigset_t set;
sigemptyset(&set);
sigaddset(&set, SIGPWR);
sigprocmask(SIG_BLOCK,&set,NULL);
```

Question 10:

```
     val=7
    +-----------
    |        *
    | val=10   *        val=15
--------------------------------
       wait     handler
```

So, val = 15.

Question 11: First the main program installs handler1. After the fork(), the child installs handler2, replacing handler1. The child then signals the parent, so handler1 is called. This prints out “counter = 1”. It then signals the child, which calls handler2. This prints out “counter = 3” and exits. This causes the parent to return from the wait() and to print “counter = 5”. So the answer is:

```
counter = 1
counter = 3
counter = 5
```