Lecture 2 (UNIX System Calls part 2)

1 Inter Process Communication (IPC)

exit code is one way, between child and parent

2 Why is `fork` + `exec` good?

1. It seems wasteful to copy information of parent to just destroy the child via `exec`
2. Windows uses `create_process` instead
3. To ensure shell redirection and other features, the command requires 10 arguments
4. Also, `fork` can be made faster by just storing references and copying only when needed

3 More about FDT and Processes

1. Each process has its associated FDT
2. Interaction with FDT can only happen via syscalls
3. Each entry in FDT points to a certain resource and offset
4. `write` appends the offset on every successful write
5. The resource can be shared, called file sharing
6. To change the standard output, we do:

   ```
   close(1);
   open("foo"); // smallest empty entry in FDT is at index 1
   ```

7. Similar as above for standard error

To write simultaneously to same file by STDOUT and STDERR, we do the following:

```
close(1);
open("foo");
close(2);
dup(1); // opening foo again will not make pointers to same object, but dup will
```

The same command in shell looks like:

```
$ program > foo >2 &1
```

4 Pipe

1. Connecting the STDOUT of one program to STDIN of a different program
2. `pipe(int[] fdarray)` is the syntax of the syscall
3. Consider the example:

   ```
   int fdarray[2];
   pipe(fdarray);
   write(fdarray[1], "hello");
   read(fdarray[0], buf, 6); // "hello" is stored at buf
   ```

4. Notice that the pipe is created from right to left
5. `pipe` is used as follows

   ```
   pipe(fdarray);
   pid = fork(); // pipe is shared
   if (pid > 0) {
     write(fdarray[1], ...);
   } else {
     read(fdarray[0], ...);
   }
   ```

6. Now, this is how shell implements the following pipe command: `command1 | command2`

   ```
   int fdarray[2];
   if (pipe(fdarray) < 0) panic ("error");
   if ((pid = fork ()) == 0) {  // child (left end of pipe)
     close (1);
     tmp = dup (fdarray[1]);   // fdarray[1] is the write end, tmp will be 1
     close (fdarray[0]);       // close read end
     close (fdarray[1]);       // close fdarray[1]
     exec (command1, args1, 0);
   } else if (pid > 0) {        // parent (right end of pipe)
     close (0);
     tmp = dup (fdarray[0]);   // fdarray[0] is the read end, tmp will be 0
     close (fdarray[0]);
       close (fdarray[1]);       // close write end
       exec (command2, args2, 0);
   } else {
     printf ("Unable to fork\n");
   }
   ```

7. Pipe's size is limited, therefore, the commands are actually run together so that the pipe gets emptied along the way and it also helps with scheduling

8. Useful link (pipe man page)