CS 341 #22 | *Pipes and Seeks*

> Warm up

Can you rewrite the following ...

```
1: char mesg[100];
2: sprintf("hello %d\n", 123);
3: write(1, mesg, strlen(mesg) );
```

using `asprintf`?

using `dprintf`?

> Remember this for later: **dup2(**existingfd**,** newfd**)**

```
1: int fd = open("log.txt", ...,...)
2: dup2(fd, 1);
3: write/fork/exec
```

> Remember this for later: **fdopen(**fd, ...**)**

```
1: int fd = open("log.txt", ...,...)
2: FILE* f = fdopen(fd,"w")
3: fprintf(f, "hello!");
```

> Pipes! (demo)

1. **How do you use unnamed pipe to send a message from the parent to the child?**

> Random access

2. What is `fseek` and `ftell`? How would you use them?

3. What happens to the other process if you `fclose` after forking?

4. What happens to the other process if you `fseek` before forking?

5. What happens to the other process if you `fseek` after forking?

6. Why does `pwrite` exist? When would you use it?

> Pipes (part 2)

7. What is a named pipe?

8. What signals can a pipe generate and when?

9. How would you modify your pipe code to send an integer value of a variable?

10. Why is it useful to close a pipe's unused filedescriptors after forking?

> Code Review: Can you improve this queue code?

```
int in, out, count;
void* buffer[16]

void enqueue(void* ptr) {
  pthread_mutex_lock (&m);
  while(count < 16) {/*loop*/}

  pthread_mutex_unlock(&m);
  p_cond_broadcast(&cv);
  count ++;
  buffer[(in++)%16 ] = ptr;
}
```

```
void* dequeue() {
 pthread_mutex_lock(&m);
 while(count  == 0) {/*loop*/}

 void* res = buffer[(out++)%16];
 p_cond_broadcast(&cv);
 pthread_mutex_unlock(&m);
 count --;
 return res;
}
```

> Altogether now... Build Descarte's Demon AKA an *autograder*!

```
 1:  int p[6];
 2:  pipe(p);
 3:  pipe(p + ___ )
 4:  pipe(p + ___ )
 5:  pid_t childid = fork();
 6:  if( childid ==0 ) {
 7:    dup2 (p[0] /*read from */, ___);
 8:    dup2 (p[3] /*write to*/,  ___);
 9:    dup2 (p[5] /*write to*/  ,___);
10:
11:    //Child should close 'in'(input), out(output) err(output)
12:    close(p[1]) close(p[2]);close(p[4]);
13:
14:   alarm(10) // Max 10 seconds  for test to run;
15:   execlp(prog, prog, NULL)
16:   }
```