Puzzle #5 Why is this 'broken'?

```
1:  int main(int argc, char**argv) {
2:    srand(time(NULL));
3:    pid_t child = fork();
4:    int r = rand() & 0xf;
5:    printf("%d: My random number is %d\n", getpid(), r);
6:  }
7:
```

Puzzle #6 What is this madness!?

```
1:  int main(int c, char **v)
2:  {
3:    while (--c > 1 && !fork());
4:    int val  = atoi(v[c]);
5:    sleep(val);
6:    printf("%d\n", val);
7:    return 0;
8:  }
9:
10:
```

deepest -

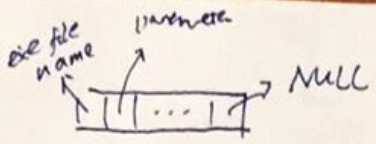
Example: *"Let is snow, let it snow!"*
`snowflake.c` attempts to create a snowstorm where every snowflake is a process *(found in /_shared/ in the CS 241 svn)*. Screen cursor logic is provided, simple API is:

- `int rows`: contains the number of rows of the terminal/console
- `int cols`: contains the number of columns of the terminal/console
- `gotoxy(x, y)`: moves cursor to a given `x`, `y` position

The key function, `snowflake()`:

```
1:  void snowflake() {
2:    srand((unsigned)time(NULL));
3:    int col = rand() % cols;
4:    int row = 0;
5:
6:    while (row < rows) {
7:      gotoxy(row, col);
8:      fprintf(stderr, "*");
9:      usleep(200000);
10:     gotoxy(row, col);
11:     fprintf(stderr, " ");
12:     row++;
13:   }
14: }
```

CS 241 | Lecture Handout #5
January 29, 2016



Puzzle #1 A most powerful program. How does it work?

```
1:  int main(int argc, char**argv) {
2:     printf("Executing %s ...\n", argv[1]);
3:     execvp( argv + 1, argv + 1);
4:     perror("Failed to be all powerful");
   }
```

Puzzle #2: Fix my getline implementation. What asserts might you add?

```
1:  ssize_t getline(char **lineptr, size_t *n, FILE
2:  *stream){
3:    what asserts would you add here?
4:    if(*lineptr == NULL) { *n = 256; *lineptr = malloc(*n);}
5:    size_t bytesread = 0;
6:    char c = 0;
7:    while( ferror(stream)==0 || feof(stream)==0 ){
8:        if (bytesread == *n) { /* extend buffer */ }
9:        c = fgetc(stream);
10:       (*lineptr)[bytesread] = c; // FIX ME
11:       if(c == '\n') return bytesread; // FIX ME
12:   }
13:    return -1; // error (e.g. end of file)
   }
```

assert?

FIX ME #1:

FIX ME #2:

Puzzle #3

What is wrong with the following?

```
1:  int main(int argc, char** argv) {
2:     char** lineptr;
3:     size_t size;
4:     size = getline(lineptr, &size, stdin);
5:     execlp(lineptr);
6:     return 0;
7:  }
```

Your notes about fork, exec

Environmental Variables

- What is getenv("HOME");
-
- What is getenv("PATH");
-
- What is getenv("USER");
-
- What is getenv("AWESOME");
- ;

Puzzle #4 An exec puzzle

What does the following example do? How does it work

```
1:
2:  int main() {
3:    close(1); // close standard out
4:    open("log.txt", O_RDWR | O_CREAT | O_APPEND, S_IRUSR |
5:  S_IWUSR);
6:    puts("Captain's log");
7:    chdir("/usr/include");
8:    execl("/bin/ls", "/bin/ls",".",(char*)NULL); // "ls ."
9:    perror("exec failed");
10:   return 0; // Not expected
11: }
12:
13:
```