CIT 5950 Recitation 10 - Pipe() and HW4

Welcome back to recitation! We're glad that you're here :)

Exercise 1

```
int main(int argc, char* argv[]) {
  int fd = open("antennas.txt", O_RDWR);
  pid_t pid =  fork();
  close(STDOUT_FILENO);
  if (pid == 0) {
     cout << "storm\n";
     dup2(fd, STDOUT_FILENO); //redirects STDOUT to the file
specified by fd
     cout << "static\n";
     exit(EXIT_SUCCESS);
  }
  waitpid(pid, nullptr, 0);
  cout << "sleep\n";
}
```

What is printed to the terminal and what is written to antennas.txt?

antennas.txt contains:
static

what was printed:

Nothing gets printed to the terminal since `STDOUT_FILENO` has been closed for both parent and child

Note:
Write function will fail if `STDOUT_FILENO` is closed. But cout<< has the buffer and might not be flushed yet, thus the behavior is undefined.

For `dup2(newfd, oldfd)`

- `newfd` must be a valid, open file descriptor.
- `oldfd` does not need to be open; if it is, `dup2` will close it without complaining. If it's not already open, `dup2` will just assign it the file descriptor `newfd`.

Using dup2 to copy a new file descriptor onto STDOUT_FILENO after it's been closed won't throw an error. The new file descriptor will take over the standard output stream.

Exercise 2: fill in the blanks

```
int main (int argc, char** argv) {
 // create a pipe to send input to program
 int in_pipe[2];
 pipe(in_pipe);

 pid_t pid = fork();

 if (pid == 0) {
   // child
   close(in_pipe[1]); // close writeend
   dup2(in_pipe[0], STDIN_FILENO); // replace stdin with read end of pipe
   close(in_pipe[0]); // close read end since it has been duplicated

   // exec the program "./numbers" with no command line args
   string command("./numbers");
   char* args[] = {"./numbers", nullptr};
   execvp(command.c_str(), args);

   // should NEVER get here
   return EXIT_FAILURE;
 } else {
 close(in_pipe[0]); // close read end

 // write inputs to the pipe
 string inputs = "30\n40\n50\n6";
 wrapped_write(to_echo, in_pipe[1]);

 // close pipe so that exec'd
 // program knows there is no more piped contents to read
 close(in_pipe[1]);

 // wait for child to finish
 waitpid(pid, nullptr, 0);
}
}
```

Exercise 3 What does this print? Does it terminate?

```
int main(int argc, char* argv[]) {
  array<int, 2> pipe_fds {-1, -1};
  pipe(pipe_fds.data());
  pid_t pid =  fork();
  if (pid == 0) {
     dup2(pipe_fds.at(0), STDIN_FILENO);
     close(pipe_fds.at(0));
     // cat should read from stdin till eof, printing everything
it reads
     vector<char*> args {"cat", nullptr};
     execvp(args.at(0), args.data());
  }
  write(pipe_fds.at(1), "the city in rain", strlen("the city in
rain"));
  close(pipe_fds.at(1));
  close(pipe_fds.at(0));
  waitpid(pid, nullptr, 0);
}
```

It prints `the city in rain`
However, it doesn't terminate since the child has its write end open, thus cat never reads an eof. To fix this, we should add `close(pipe_fds.at(1));` before calling `execvp(args.at(0), args.data());` in the child process.