Assignment 8

(1) **(3 pt)** Your program functions successfully and correctly as the requirements mentioned above.

Answer: The program is run as follows:

```
$ ./assignment8
Child incrementing, value: 1
Parent incrementing, value: 2
Child incrementing, value: 3
          .
          .
          .
Child incrementing, value: 99
Parent incrementing, value: 100
```

The program creates a file named `num.txt`, and performs the required incrementation task on it.

```
$ cat num.txt
100
```

The actual output of `cat` does not include a trailing newline.

We also check the output of `ps` to make sure that no updates to the file are continuing in the background:

```
$ ps
  PID TT  STAT    TIME COMMAND
93423  0  Ss   0:00.08 -sh (sh)
93598  0  R+   0:00.00 ps
```

(2) **(1 pt)** Describe your implementation in your report.

Answer: We start by opening the file `num.txt` in `w+` mode, which will truncate the file if it exists, and create the file if it doesn't exist. This will allow both reading and writing to the file.

```
FILE *numfile = fopen("num.txt","w+");
```

We write `"0"` to the file using the file descriptor from `fileno`. `pwrite` is used because it is atomic and does not cause buffering issues later on.

```
int fn = fileno(numfile);
pwrite(fn,"0",1,0);
```

Forking is done and the child process id is saved:

```
pid_t pid = fork();
```

Before the parent and child start operation, we make sure that they are set up to communicate with each other using user-defined interrupts, namely `SIGUSR1` and `SIGUSR2`.

```
TELL_WAIT();
```

The child process starts first, incrementing to odd numbers, so it increments the counter, tells the parent using an interrupt, and waits for the parent. This happens repeatedly in a while loop.

```
int n = increment_counter(numfile);
printf("Child incrementing, value: %i\n",n);
TELL_PARENT();
WAIT_PARENT();
```

The parent process increments to even numbers, so we check whether the file is at 100 in the parent. The parent waits for the child first, and then continues execution.

```
WAIT_CHILD();
int n = increment_counter(numfile);
printf("Parent incrementing, value: %i\n",n);
if (n == 100) {
    kill(pid, SIGKILL);
    exit(0);
}
TELL_CHILD(pid);
```

`kill(pid, SIGKILL)` makes sure that the child process is terminated so it is not left waiting forever. `exit(0)` then ends the program.