Programming Assignment #3 **Due: 3/29**

1. [100 points] Using what you know about process relationships, process control, and file and directory operations, write a function `snapshot()` as described in the attached "man" page (q.v.).

   This function is intended for incorporation in a program you send to a (human) client. The idea is that at any point in its execution, your program can call `snapshot()` to create a compressed tarball (a "snapshot") containing a core dump, a matching executable, and a `README.txt` file containing a message from the program. The snapshot is suitable for attaching to a bug report to be sent to the developer for debugging.

   You will find source for test program for `snapshot()` in the `snapshot_t0.tgz` tarball available from the web page, along with a Makefile and header files. Un-*tar* this into a clean directory, add your own `snapshot.c` containing the function, and *make* will compile and link a simple test program, `snapshot_t0`, that produces a snapshot whose name (i.e., ***`ssname`***) is given on the command line.

   Notes:

   - Although you might think you could copy your program's executable by opening and copying `argv[0]`, this is not reliable. Instead, make use of the fact that on Linux systems there's a symlink for a process's executable in `/proc/`***`pid`***`/exe` where ***`pid`*** is the process ID. Open that link and copy it to a file.
   - `snapshot()` should fail if the snapshot directory already exists. (It assumes it's not the result of a `snapshot()` call so it avoids trashing it.)
   - `snapshot()` should overwrite any existing identically-named tarball. (It assumes that it's the result of a previous execution and that the user wants to update it.)
   - Use *setrlimit(2)* to allow an unlimited core dump size. (As a good developer, you should have set "`$ ulimit -c unlimited`", but your client might not.)
   - `snapshot()` does not call *exit(3)*. The same process may call `snapshot()` at several different points (possibly with different `ssnames`) and the process will continue to run.
   - Use OS calls and utility functions (i.e. manual sections 2 and 3) to do most of the work – Don't try to do everything from scratch! Requirement: You may use *system(3)* to invoke *tar(1)*, but you may not invoke any other system executables.
   - Your tarball should include a makefile that will make `snapshot_t0`. *Do not modify `snapshot_t0.c` in any way.*
   - The instructor will test your code with a different program than `snapshot_t0`. That program will "`#include`" `snapshot.h` (which you are not to modify) and compile and link in your `snapshot.c`.

SNAPSHOT(3) CptS 360 SNAPSHOT(3)

NAME

`snapshot` - generate a "snapshot" tarball of a running program suitable for debugging

SYNOPSIS

```
#include "snapshot.h"

int snapshot(char *ssname, char *progpath, char *readme);
```

DESCRIPTION

When `snapshot()` is called from a running program, it creates a *gzip(1)*-compressed tarball containing a copy of that program's executable, a core file, and a `README.txt` text file.

`ssname` is the name of a directory that will be temporarily created (if the calling process has permission to do so) that will hold those files. The name of the tarball is ***`ssname`***`.tgz` and it holds a directory named ***`ssname`*** whose contents are:

- ***`progname`***: An executable copy of the binary code for the currently-running executable. ***`progname`*** is ***`progpath`*** without any directory information. (hint: *basename(3)*) For this to work, ***`progpath`*** should usually be `argv[0]`, where `argv` is the first argument to `main()`.
- `core`: A core dump of the current program made by `snapshot`.
- `README.txt`: A file containing the contents (and only the contents) of the ***`readme`*** string. If ***`readme`*** does not end in a newline, it will be added to this file.

`ssname` and its contents will be deleted after the tarball is created.

EXAMPLE

A program *foo* calling

```
snapshot("snap_1", argv[0], "This is where I think the problem is.");
```

will create a gzipped tar file `snap_1.tgz`. If you look at this file from the shell, you will find:

```
$ tar -tzf snap_1.tgz
snap_1/foo
snap_1/core
snap_1/README.txt
```

If you then un-`tar` the contents and `cd` to the `snap_1` directory, you will be able to run

```
$ gdb foo core
```

and examine variables as usual. If you examine the contents of `README.txt` you get:

```
$ cat README.txt
This is where I think the problem is.
```

ERRORS

If `snapshot()` fails for any reason, it returns -1. Otherwise, it returns 0. If any of the system routines it calls sets `errno`, that value is unchanged.