CSC B09 Assignment 3, Winter 2018: The Great Ancestor Process

Due by the end of Friday March 23, 2018; no late assignments without written explanation.

We've discussed that the only way new processes come into existence in unix is via "fork()". However, one process has to be created upon system startup as a special case. This is process ID number 1, and it runs a program called "init" which is responsible for starting other fundamental system processes.

"Init" does a bit more than that: It monitors these processes and when one exits, init starts another.

Thus this works well for login-oriented processes: Init starts a program "getty" which initializes the terminal and prints an appropriate message; when the user types their logname, getty execs the "login" program (without forking, so the login program has the same process ID number which init is still monitoring); when you log in successfully, login execs your shell (still with the same process ID number); and when you log out, init notices that that process ID number no longer exists and starts a new getty on that terminal.

In this assignment you will write a program much like "init", although for testing you will run programs which are slightly more fun than getty. You will read from a configuration file whose name must be specified on the command-line ("init" defaults to reading /etc/inittab, but that default would be pointless for this assignment). Your program takes an optional '–r' parameter which indicates the "run-level", which is a single character whose meaning is described below. If this parameter is not present, the run-level defaults to 3. Use getopt() to parse the command-line, and output a fatal error message if the subsequent number of file name parameters is not exactly 1.

The configuration file has one entry per line, with fields separated by colons. Comments are introduced by '#' and run to the end of the line; and blank lines are permitted (that is, they must be ignored, rather than yielding an error message).

If the line is not empty (nor is solely a comment), it contains three colon-separated fields (the real *inittab*'s first field is omitted here):

- The first field is a list of which "run-levels" the given process should be executed for, as a string of individual run-level characters. For example, "23" indicates that the process should be run in run-levels 2 and 3. If this first field on the line is empty, then the line applies to all run-levels. Otherwise, if the current run-level as specified by the –r option is not listed, your program will discard this line after parsing.
- The second field specifies the re-spawning behaviour for this process. For this assignment, the contents of this field must be either the string "once", which means that the process should be executed once and your program will ignore whether or when it terminates; or the string "respawn", which indicates that your program should watch for the process to terminate and then start a new one. Any other value is a fatal error.
- The third field is a command-line to be executed with "sh –c *string*".

You may impose a maximum of 100 entries, but if this is exceeded you must exit with an appropriate error message rather than exceeding array bounds. (In real life we would build a dynamic data structure of some kind, such as a linked list, but you don't have to do that for this assignment; you'll do that in assignment four.)

In the directory /cmsfaculty/ajr/b09/a3 there is an example configuration file, and some sample programs you can list in your configuration file, some of which perform slightly cute services. You do not submit a configuration file; those files are just for your own testing. For *our* testing, your program will be run with a completely different configuration file and it must obey the instructions in that file.

The supplied sample program "listener" allows you to offer any program as a network service; it takes a command-line parameter saying what program to offer. You must be *extremely careful* about what programs you use in this manner! Anyone on the planet who connects to that service will be running a program under your linux account with your permissions. That's why all of the commands in the sample program "sillyshell" are fake or trivial (in real life, we'd make them log in first... and we'd be very

(over)

cautious about any possible security vulnerabilities in that too!). For the parameter to the "listener" program in your configuration file, I suggest you restrict yourself to /bin/date, simple echo commands, /cmsfaculty/ajr/eliza/eliza (which I've examined for network security issues, albeit not recently), or /cmsfaculty/ajr/b09/a3/sillyshell.

Please see the Q&A web page for suggestions on dealing with '#' comments (which is actually quite easy), blank lines, and the basic field parsing (you can't use strtok() because there can be empty fields; but again, it is a fairly easy parsing operation).

To submit

Your file must be named "myinit.c", and as usual, must compile on the UTSC linux machines with no warning or error messages. Once you are satisfied with your file, you can submit it for grading with the command

```
submit -c cscb09w18 -a a3 myinit.c
```

You may still change your file and resubmit it any time up to the due time. You can check that your file has been submitted with the command

```
submit -l -c cscb09w18 -a a3
```

And as before, you can resubmit with –f.

Please see the assignment Q&A web page at

```
https://mathlab.utsc.utoronto.ca/courses/cscb09w18/a3/qna.html
```

for other reminders, and answers to common questions.