TDT4186 Operating Systems
Practical Exercise 1 - Alarm

Håvard R. Krogstie

Magnus Lykke Dahlø

Herman Seternes

February 27, 2022

(a) We implemented the alarm program across three source files, with menu logic in `main.c`. The `readline()` function is a simple wrapper around `fgets(3)` that safely consumes whole lines, rejects the `'\n'` and returns the length.

The text printed by the menu is taken from the example, and the current time gets printed in the same format. Time formatting is done in `timeutil.c`. After the prelude, an infinite loop feeds user commands into a `switch` statement, with some cases requiring further input.

(b) The data structure for alarms is `typedef struct{ ... } Alarm;` defined in `alarm.h`. Here we also define `MAX_ALARMS` to be 20. A global array of 20 alarms is defined `alarm.c`.

An `Alarm` consists of a unix timestamp, a PID, and the chosen sound file (a number between 1-3). If the PID is equal to 0, that means the alarm is inactive, and can be overridden by a new alarm. Adding a new alarm thus begins by finding an open slot in the array. Parsing of timestamps is performed in `timeutil.c`, which results in a target unix timestamp.

When listing alarms, the `alarm` array first gets sorted in chronological order, while inactive alarms get placed in the back of the array. In the printed list, the 1-index of each alarm is printed, along with its target time. These indexes will **not** change until a new list is printed, so you can always use the last list printed to refer to active alarms. This comes in handy when removing alarms.

(c) Spawning the alarm

When an new alarm is created with target timestamp, the time difference is calculated. If it is in the future, the time left gets printed, and a new process gets spawned using `spawn_alarm`, which in turn uses `fork(2)`. The parent process takes the PID of the child, and stores it in the `alarms` array. All other data about the alarm was stored in the array *before* the call to fork.

Meanwhile, the child process begins a `sleep(3)` for the duration of the alarm. After the time is up, the child activates the alarm by printing and starting an audio player. Before using `execvp(3)` to launch the audio player `mpg123`, we need to detach the fork process from `stdout` and `stderr`. Otherwise `mpg123` will change the termios settings in the terminal, which affects the interactivity of the main process. We solve this by `freopen(3)`ing the streams to `/dev/null` before calling `execvp(3)`. A nice effect of this is that it becomes possible to stop the ringing sound early using the `"c"` option while the sounds is playing. The call to `execvp(3)` should never return, but we `_exit(2)` afterwards just in case.

(d) Canceling alarms

As discussed in section (b), the alarm numbering only changes when the list is printed, so the cancel command takes in a number from the previously printed list. We have stored the `pid` of each active alarm, so the parent process can use `kill(2)` to send a `SIGKILL` signal to the alarm being canceled. This will end the process prematurely. The original target time of the alarm gets printed to the user as well.

A caveat here is that alarms can finish, and have their alarm slot be replaced by new alarms. The numbers on the last printed list will then no longer refer to the alarms shown on the list. To prevent confusion, the program will not let you cancel alarms that have never been part of a list, instead generating a new list for you and asking you to try again.

(e) Catch The Zombies!

When an alarm fork terminates, it's either because the alarm has rung, or because it was killed. The fork then becomes a zombie, which the parent process must handle. This is done in `cleanup_zombies` in `alarm.c`. It performs the `waitpid(2)` syscall on every alarm where the `pid` is not 0. Using the `WNOHANG` flag, the syscall won't block the main process. If the alarm fork is terminated, either normally or by a signal, the main process removes the alarm from the list of active alarms by setting the `pid` = 0.

This cleanup function is performed at the top of every other public function in `alarm.c`, to make sure the alarm list only contains active alarms before using it.

An extra feature we added was a `cancel_all_alarms()` function that gets called before the program exits, registered using `atexit(3)`. This kills all active alarms, and waits for all children to terminate. Importantly, a fork will never call this, since it always gets terminated by a signal, replaced by `execvp(3)`, or exited using `_exit(2)`.

(f) A real alarm clock

We placed three different mp3 files inside the `audio/` folder. The file names are hard coded into the program. When scheduling an alarm, you must pick a number between 1-3, with 1 being the default. This is stored inside the `struct Alarm`, and gets read when the alarm fork is done sleeping.

The function `play_sound` takes this number, and loads the correct audio file path. `execvp(3)` is used to replace the alarm fork with a `mpg123` process image, which plays the sound file provided as an argument. We also provide the `-q` option, to prevent spam in stdout.

Test case documentation

(1) Scheduling input parsing

Launch the program, it should print the current time in your local time zone. Type "`s`" to schedule an alarm, for each of the following incorrect date + time inputs.

```
tomorrow
2022-02-23 15:06:08!
2022-02-23 15:06:68
```

Each attempt should tell you that the format is incorrect.

Now try entering a date that is in the past, like the year 0023:

```
23-02-23 15:06:08
```

It should parse it correctly, and ask you for choice of alarm sound. Press enter for default. It should then complain that alarms must be set in the future.

Type "`s`" again, this time enter a valid date+time, such as

```
2022-5-1 15:0:0
```

For choice of alarm sound, type "`gong`". It should complain that it's not a number.

Try again, this time pick the alarm sound number "`6`". It should complain that the choice of sound is invalid.

Finally we will enter a valid alarm configuration! Type "`s`" again. Enter sometime the next minute as the time, and pick the alarm sound "`3`". The alarm should now have been added, and print out the number of seconds until the target time.

Wait for that many seconds, the alarm should ring, both with `RING RING` in the terminal, and with a sound playing. We chose sound 3, which should be a few seconds of Halloween themed *Kahoot!* lobby music. Verify that the time of the ring is actually the provided time using your system clock.

Type "`x`" to quit.

(2) Listing alarms

Start the program, type "`s`" to schedule an alarm three minutes in the future. Pick the default sound this time, by pressing enter.

Type "`l`" for `list`, which should show the new alarm listed as alarm number 1.

Schedule a new alarm using "`s`", this time set the time one minute earlier than the previously scheduled alarm.

Type "`l`" again, which should show two alarms, in chronological order. In other words, alarm 1 and 2 should be the second and first scheduled alarms, respectively.

Schedule yet another alarm, this time a long time into the future. When printing the list using "`l`", the new alarm should be listed as alarm number 3.

Wait for the first alarm to ring. It should play the Windows XP starting sound.

Now list alarms again using "`l`". The list should now only contain two alarms, the first and third alarms we added. They should be numbered 1 and 2, respectively.

Type "`x`" to quit, which should tell you that those two alarms got cancelled, and display when they were supposed to ring. Wait for a minute to make sure that the alarm was actually cancelled.

(3) Canceling alarms

Schedule three alarms, one to ring in one minute, the other in two, the last in a day. Pick sound "`2`" for all alarms.

List them all using "`l`". They should be in chronological order.

Wait until the first alarm rings. it should be a classic alarm ringing sound.

Type "`c`" to cancel an alarm. When asked for number, try giving "`1`", "`gong`" and "" as options. They should all fail. The reason the first fails, is that alarm 1 has rung. Now try giving "`2`" as the alarm to cancel. It should work, and also tell you when alarm 2 was supposed to ring, in less than a minute.

Wait until alarm 2 was supposed to ring, to make sure it was actually cancelled.

Now type "`l`" again to list the active alarms. The list should now only have one alarm, the one that is a day away. It should now be alarm number `1`, since it is the only alarm.

Type "`c`" and select "`1`" to cancel it.

Finally type "`x`" to exit. It should not print any additional cancellations due to quitting.

(4) Canceling new alarms
Type "`s`" to schedule a new alarm, to ring a day from now. Do it again, to schedule an alarm 1 hour from now.

Now try using "`c`" to cancel alarm number 1. It should tell you that the alarm hasn't been listed before, and print out a list. On this list, the upcoming alarm has number 1, while tomorrows alarm has number 2.

Now try cancelling alarm number 1 again. This time it will work.

Use Ctrl-D to exit the program. This should tell you that tomorrow's alarm gets cancelled as well.