Assignment 1

Due January 26^{th} 2009

Interprocess communication, signals, process creation

The problem

Your task is to implement a simple system utility called

alert

which will send a wakeup message to the screen at the time given as argument.

The utility consists of a program called alert will be invoked (i.e. started from the very beginning) every time a command is issued to it. This is an example of a script that could be used to test your program:

```
gcc -o alert alert.c
./alert wakeup 12
./alert wakeup 2
./alert cancel 77
```

It is probable that your implementation of alert will spawn new processes from inside alert using fork().

alert will accept the following arguments from the command line:

alert wakeup time asks for a message to be displayed on standard error in time seconds.

alert cancel time cancels a previous request to wakeup after time seconds. Note that the time argument identifies a wakeup request made earlier.

alert change time1 time2

the third case is just a combination of the first 2.

Some rules

1. All the times are in seconds; for simplicity these times are not absolute, but relative to the current time.

2. Incorrect arguments are ignored without any ado.
3. There is one alert per shell (i.e., terminal window, for the purpose of this assignment).
4. There may be more than one copy of alert in action at any given time, provided they are created in different shells. These copies may have the same current directory, etc.
5. There may be more than one request for a wakeup with the same time argument; each cancel cancels only one request (not all).
6. You cannot use any system facility (such as cron);
7. The only system calls allowed are: file manipulation calls (open(), close(), write(), lseek(), etc.) plus fork(), kill(), sleep(), alarm(), wait(), getpid(), getppid(), etc.
8. While the kill() call can be used freely, all signals sent must be caught by the receiver. Consequently, you cannot use SIGKILL.

Example

Make sure to look at this example before you start programming.

The time is:	alert is called with these arguments:	this happens:
0	wakeup 5	do it in 5 seconds
4	wakeup 6	do it in 6 seconds
5		wakeup 5 is heard
7	cancel 6	cancels the request made 3 seconds earlier
12	blah blah	ignored
17	change 7 4	ignored
19	wakeup 2	
21		wakeup 2 is heard
24	cancel 2	ignored
28	wakeup 9	
30	wakeup 9	different starting points
31	cancel 9	refers to the first wakeup 9 (made at time 28)
33	change 9 12	deletes the request made at time 30 and asks for a wakeup in 12 seconds
45		wakeup 12 is heard

Hints

- This assignment is hard to implement without a fork() although it could be done.
- The simplest ways to have a program do something at a given moment are: (a) sleep() and usleep() (b) alarm() (c) select().
- There are many ways of organising your program. Two seem the easiest:
  - Keep a database (file) with the descriptions of the wakeup requests that are pending. Fork a separate process for each request; then use the file to identify the process that needs to be killed in case of a cancellation. If you use a database, make sure to use a lock to guarantee mutual exclusion.
  - Have a *dæmon* (a never–ending process) which handles all the requests. This approach requires some advanced knowledge of inter–process communication. If you use a dormant *dæmon*, make sure that each invocation of alert is able to identify its own *dæmon* by checking its ppid (in case more than one shell is running alert).

Useful links:

- Beej (see sections 2 and 3)
- forking code
- use of signals

Assignment requirements

Your solution must satisfy these requirements:

1. A message appears on standard error when the wakeup call is due.
2. When a process receives a signal, it must send a message to standard error saying that it received it (this requirement is needed for testing purposes). The message gives the pid of the process and the signal name. This requirement cannot be circumvented by having the process sending the signal issue the message; it must be the receiving process.

3. The processes implementing alert must delete all the files that they created and that are no longer needed. If the example above is used to test your software, there should be no files left slightly after time +45 (e.g. at time +46).

Submission rules

Submission rules are posted. They must be followed.

Grading

The assignment is worth 10 marks which are distributed as follows:

	action	marks
1	alert exits without waiting	1
2	Wakeup message appears at the right time	1
3	Wakeup appears on stderr	1
4	Cancel does cancel	2
5	Cancel message appears on stderr	2
6	Cancel removes oldest Wakeup	1
7	lock properly set and removed	2
8	no files left if not needed	2

Steps 1–7 form the basic assignment; if done perfectly, they are worth 10 points. Step 8 is a bit more difficult and should be treated as an optional bonus.