Confine

This is an individual assignment. The goal of this assignment is to write a program that ``confines'' the behaviour of other programs. Such programs are useful in variety of situations -- examining malicious programs, running untrusted programs, and autograding.

You will write a program called `confine`, that runs another program with several restrictions. The program `confine` will be called as:

```
./confine /path/to/program-to-be-confined [args-to-program1 [args-to-program2...]]
```

The program being confined need not accept any arguments, so `./confine /path/to/program-to-be-confined` is valid too.

Example

An example of a call to confine will look like this:

```
./confine ./samples/sub1 1 2 3 0
```

This will cause `confine` to execute `./samples/sub1` with the arguments `1 2 3 0`.

Restrictions to be applied

The `confine` program should prevent the confined programs from:

1. Using more than 64MB of memory (64x1024x1024 bytes).
2. Creating a file greater than 4MB (4x1024x1024 bytes).
3. Running for more than 1 minute real time (i.e. wall-clock time).

Return value

The `confine` program should return the confined program's return value if the confined program ran normally without exceeding the restrictions.

It should return `127` if the confined program was terminated because it encountered a restriction.

It should return `128` if the confined program died because it encountered a bug.

It should also create a file `confine_result.txt`, in the following format:

1. First line contains confined program and its arguments
2. Second line contains "NORMAL", "TERMINATED", "TIMEOUT" depending on whether the confined program i) ran normally, ii) was terminated for any reason, iii) was terminated for a timeout respectively.

Recommended Implementation

This assignment requires you to use (among others) `fork`, `execve`, `waitpid`, `setitimer` system calls, as well as set up signal handlers using `sigaction`.

Read the manual pages, and ask questions on Blackboard if needed. Specific instructions are given below.

Execute programs specified on the command line

Write `confine` to execute the program specified on the command line.

The manual page for `execve` contains sample code for a program `myecho.c` that you can use to verify correctness. It also contains example code for how to execute a program, and you can adapt it for this assignment.

Detect normal and abnormal termination

Detect cases when the program exits normally and when it dies because of a bug (e.g. segmentation fault) using `wait`.

Return the appropriate value (the program's return value) or 128 if the program encountered a bug -- you'll know this is the case because the return values will be negative.

None of the programs to be confined in this assignment will ordinarily return a value greater than 64.

The programs in the included `samples/` directory all show various pathological behaviour. See the source code and the included `README.txt`.

Set limits on child processes

Use the Linux system calls `setrlimit` and `getrlimit` to set limits on memory, file size, and CPU time. Note, these limits should only apply to the confined process, not `confine` itself.

To do this, call `setrlimit` in the child process after `fork()` but before `execve()`.

Set timeouts on processes

The `setrlimit` CPU time restriction applies only when the program is actually running. In some instances, a program may not make progress and not consume CPU time -- for example, when it is waiting for input.

To detect and terminate such errant programs, you will need to set up a timer *before* running the program (using `setitimer` or `alarm` for example). If the program has not terminated by the time you receive SIGALRM, you should kill it using `kill` and `SIGKILL`. This set up should be done in the parent, before `fork()`.

Note, SIGALRM is not guaranteed to deliver a signal exactly on time, and that's okay.