dumb-init for script

Problem

So you got your command that you would like to run it using a docker container, problem is once you got your program running using whatever mean possible, you see that it is running, but when you want the program to finish because it is say a web service, you notice that it takes a significant amount of time for the container to stop. You wonder to yourself, you read about `docker stop` sending sigterm then after a grace period of 10 seconds it will send sigkill to forcefully kill the process within the container.

You learned about that if the main program is ran under shell form, i.e. no JSON array, then your main process will be spawned by a shell, with that shell being PID of 1. Shell by itself doesn't handle sigterm signal, so you thought to yourself, let's make our main program the main process by switching to exec form.

Now you got your program running under exec form, and you want to stop it again, but there it is, it doesn't stop immediately either?! You know that your program is running under PID 1 so the sigterm signal should reach the main process, and it should gracefully stop because you tested locally. What's going on here?

Special PID 1

It turns out Linux kernel treats PID 1 as a special case, and applies different rules on how it handle signals. For a normal process if it doesn't register its own handlers for SIGTERM then it will use the default implementation for handling SIGTERM which is to kill the process.

However, for PID 1, kernel will NOT fallback to the default behavior. Therefore, SIGTERM will have no effect on the process. This is because PID 1 is run by the init process, and this is kind of safety mechanism to prevent people from accidentally sending SIGTERM to the init process.

This is why your program will not react to the SIGTERM signal, because it is treated as the "init" process by being PID of 1.

Resolution

To solve this, we would need an init process to take care of the signal forwarding and reaping of the process for us. Luckily we can do it in two ways:

1. Use the `--init` flag to use `Tini` as our init process
2. Use `#!/usr/bin/dumb-init` in your script to start a script with an init process that also takes care of the signal forwarding. Note that `dumb-init` needs to be installed in the image

before it can be used because it is a separate program.

Revision #1
Created 2023-07-10 00:29:31 UTC by Tamarine
Updated 2023-07-10 02:48:35 UTC by Tamarine