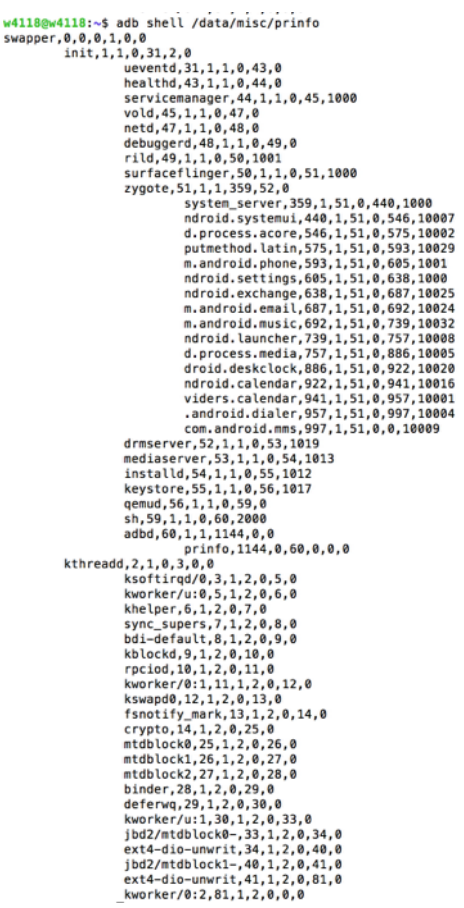
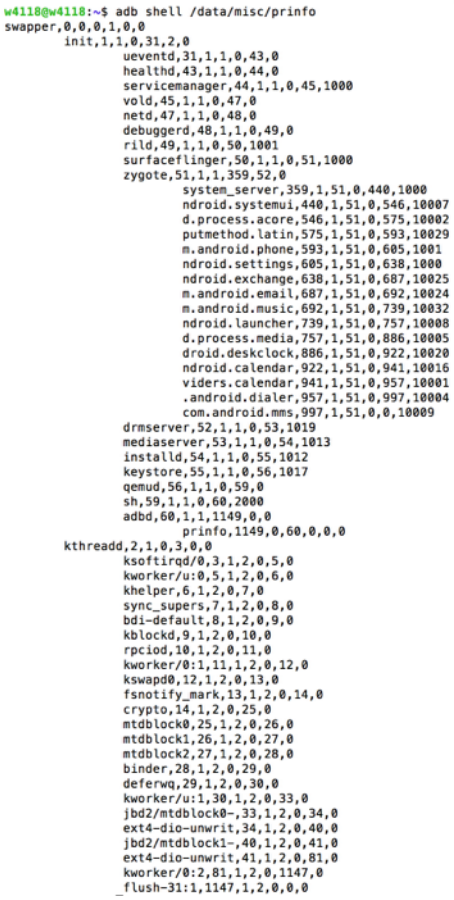
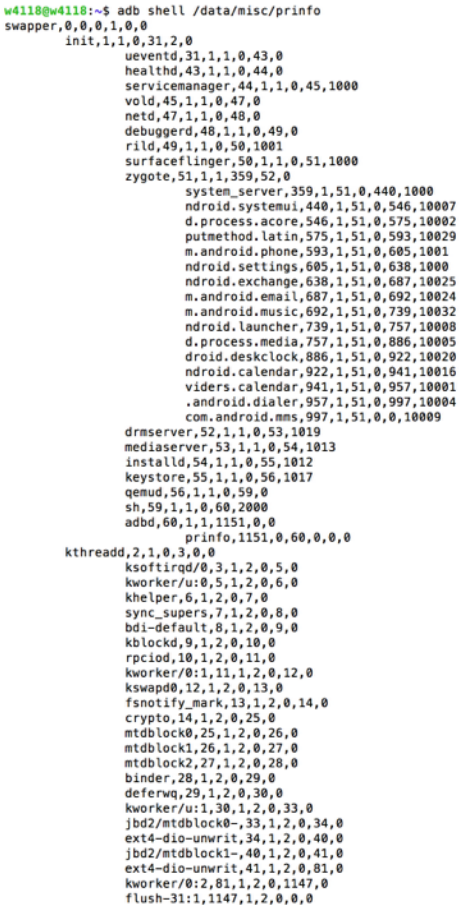
4, a)Run your test program several times. Which fields in the prinfo structure change? Which ones do not? Discuss why different fields might change with different frequency.

A:







The pid “prinfo” change each run. Because every time we use adb shell to create new instance of prinfo, the new pid will be re-assigned to prinfo. And the prinfo is actually the child process of adbd. So both child pid of adbd and pid of prinfo itself change.

Though it is expected to observe some of the "process state" changes, but in the few run of our program, we didn't see state field change.

4, b)Start the mobile web browser in the emulator, and re-run your test program. How many processes are started? What is/are the parent process(es) of the new process(es)? Close the browser (press the "Home" button). How many processes were destroyed? Discuss your findings.

A: When we start the web browser, the android.browser process is created. It’s child process of zygote. When close the browser(I mean go to app manage of Android to stop the browser), the android.browser will be destroyed. If we tap the Home button instead, the android.browser still exists. It may go to background mode, and later might be killed by android if resources are limited.

4, c)Notice that on the Android platform there is a process named zygote. Investigate this process and any children processes:
i)What is the purpose of this process?

Zygote is a daemon whose goal is to launch Apps, do the function similiar to fork(). Zygote is is the parent of all App processes in android.

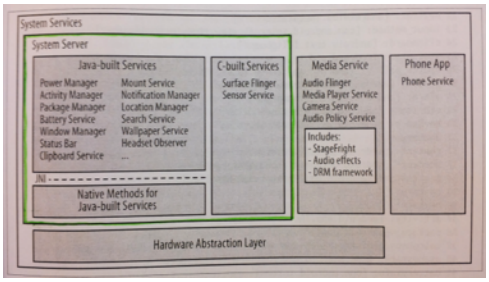
ii)Where is the zygote binary? If you can't find it, how might you explain its presence in your list of processes?

There's no independent zygote binary but zygote is started by /system/bin/app_process -Xzygote.
After kernel is loaded, init.rc is parsed and native services are started. With this, the /system/bin/app_process is run.
We can find some comments in app_process source(frameworks/base/cmds/app_process/app_main.cpp):

```
// --zygote : Start in zygote mode
// --start-system-server : Start the system server.
// --application : Start in application (stand alone, non zygote) mode.
// --nice-name : The nice name for this process.
//
// For non zygote starts, these arguments will be followed by
// the main class name. All remaining arguments are passed to
// the main method of this class.
//
// For zygote starts, all remaining arguments are passed to the zygote.
// main function.
```

This eventually calls AndroidRuntime.start() function, passing it the parameters com.android.internal.os.ZygoteInit and start-system-server. ZygoteInit.main() first registers the zygote socket. It initializes a lot of classes system-wide resources like drawables, xmls, etc. Then it calls startSystemServer() which forks a new process for com.android.server.SystemServer. Zygote process opens a socket /dev/socket/zygote to listen for requests for starting applications.

iii)Discuss some reasons why an embedded system might choose to use a process like the zygote.
A:
1, To save time start other applications. Once Zygote starts, it preloads all necessary Java classes and resources, starts System Server, initialize all the different System Services and the Activity Manager.



For System Service, we refer to the picture above. They are shared resources for other Apps. So when launch app, there's no need to start services again.

2, For the design pattern of android, zygote is to start Dalvik VM each time App launches. This is different from the way linux do. So it's necessary to take zygote the responsibility to manage all App processes.

3, Save memory. As we know, Android runs on Linux. The Linux Kernel implements a strategy call Copy On Write (COW). In the case of Android those libraries are not writable. This means that all process forked from Zygote are using the exact same copy of the system classes and resources. No matter how many applications are started the increase in memory usage will be a lot smaller.