About initramfs and creating initramfs

What is initramfs ?

The word "initramfs" is made up of three words *"initial" "RAM based" "file system"*

This is nothing but a file system hierarchy (which you just studied in the last lecture, all those directory structure), made to live in the RAM of the device by compressing it *(we use compression because RAM is precious, we cannot use whole RAM just to store the FS)* and during booting, Linux mounts this file system as the initial file system. That means you just need RAM to mount the FS and get going with the complete boot Process.

Does that mean usage of initramfs is compulsory? Used everywhere, every time ?

No, not necessarily. Using initramfs is optional.

So, why do we need initramfs ?

Let's understand this with an example.

Let's say you have a product and your product has USB interfaces, mass storage devices like SD card and let's say you also have networking peripherals like Ethernet and also display.

Now, to operate this wide range of peripherals, all the device drivers must be in place right?

That means all the drivers must be loaded in to the kernel space.
And along with the drivers, some peripherals may require the firmware binaries to operate.

One idea is make sure that all those drivers are **"built in"** in to the kernel, that's not a great idea because that makes your Linux kernel specific to your product and it will drastically increase the Linux kernel image size.

Another good way is, you come up with the minimal file system, where you store all your drivers and firmware, and load that FS in to the RAM and ask the Linux to mount that file system during boot(Thanks to kernel boot arguments , you can use the kernel boot arguments to indicate kernel that your FS resides in RAM)

When the kernel mounts that file system from RAM, it loads all the required drivers for your product and all the peripherals of your product are ready to operate, because the drivers are in place.

after That you can even get rid of this RAM based file system and use (switch to) some other advanced file system which resides on your other memory devices like eMMC/SD card or even you can mount from the network.

So, basically initramfs embedded into the kernel and loaded at an early stage of the boot process, where it gives all the minimal requirements to boot the Linux kernel successfully on the board just from RAM without worrying about other peripherals. And what you should store in initramfs is left to your product requirements, you may store all the important drivers and firmware, you may keep your product specific scripts, early graphic display logos, etc.

How to keep initramfs in to RAM?

There are 2 ways,

1) You can make initramfs "built in" in to the Linux Kernel during compilation (i will show you later in this course) , so when the Linux starts booting , it will place the initramfs in the RAM and mounts as the initial root file system and continues.

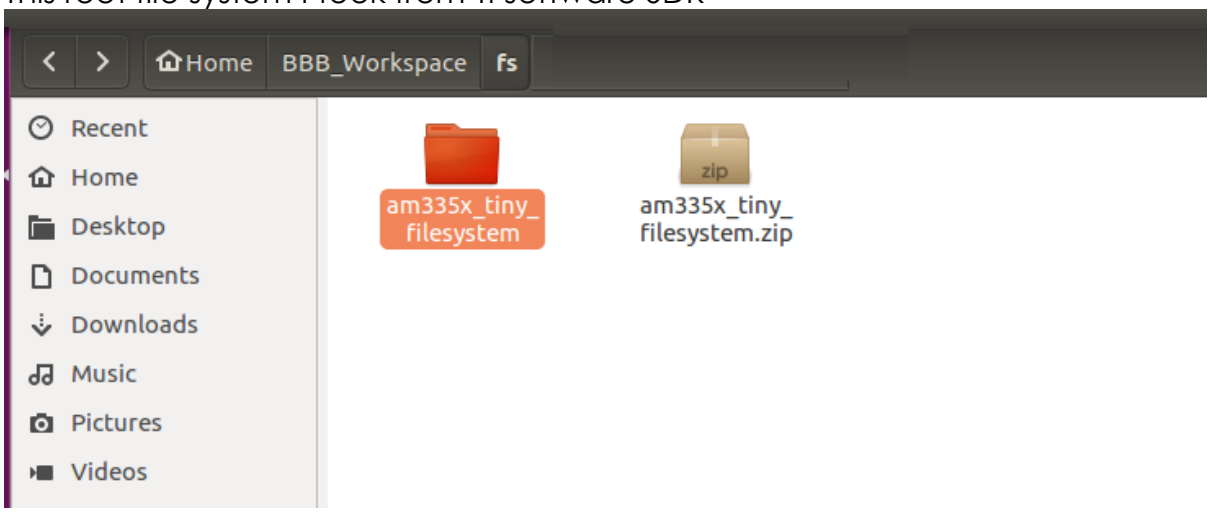
2) You can load the initramfs from some other sources in to the RAM of your board and tell the Linux Kernel about it (that is , at what RAM address initramfs is present) via the kernel boot arguments.

We will see both methods in this course as we make a progress!

How to generate the innitramfs ?

Follow these steps to generate the initramfs
Reproduce these steps at your desk, because we are going to use the generated initramfs in our course as we move along.

Step 1: Download and extract the root file system attached with this lecture. This root file system I took from TI software SDK



Step 2: Get in to the extracted folder and run the below 2 commands from the "terminal"

```
# find . | cpio -H newc -o > ../initramfs.cpio
# cat ../initramfs.cpio | gzip > ../initramfs.gz
```

In the first command we are generating a **cpio** archive and then **gz** archive. Refer the image below.

```
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$ find . | cpio -H newc -o > ../initramfs.cpio
11593 blocks
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$ ls -l ../
total 9076
drwxrwxr-x 17 kiran kiran    4096 Jun  8 16:41 am335x_tiny_filesystem
-rw-rw-r--  1 kiran kiran 3350311 Jun  8 16:42 am335x_tiny_filesystem.zip
-rw-rw-r--  1 kiran kiran 5935616 Jun  8 16:46 initramfs.cpio
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$ cat ../initramfs.cpio | gzip > ../initramfs.gz
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$ ls -l ../
total 12112
drwxrwxr-x 17 kiran kiran    4096 Jun  8 16:41 am335x_tiny_filesystem
-rw-rw-r--  1 kiran kiran 3350311 Jun  8 16:42 am335x_tiny_filesystem.zip
-rw-rw-r--  1 kiran kiran 5935616 Jun  8 16:46 initramfs.cpio
-rw-rw-r--  1 kiran kiran 3105573 Jun  8 16:47 initramfs.gz
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$
```

You will end up with the file "**initramfs.gz**" which is about 3MB.

STEP 3 : install mkImage command

For that run "**apt-get install u-boot-tools**" on your terminal software, this will install all the u-boot related tools along with **mkImage** tool.

 # apt-get install u-boot-tools

STEP 4: Make our initramfs , uboot friendly by attaching the uboot header with load address and other info .

Run the below command.

mkimage -A arm -O Linux -T ramdisk -C none -a 0x80800000 -n "Root Filesystem" -d ../initramfs.gz ../initramfs

```
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$ mkimage -A arm -O linux -T ramdisk -C none -a 0x808
00000 -n "Root Filesystem" -d ../initramfs.gz ../initramfs
Image Name:   Root Filesystem
Created:      Thu Jun  8 16:50:37 2017
Image Type:   ARM Linux RAMDisk Image (uncompressed)
Data Size:    3105573 Bytes = 3032.79 kB = 2.96 MB
Load Address: 80800000
Entry Point:  80800000
kiran@kiran-fastbitlab:~/BBB_Workspace/fs/am335x_tiny_filesystem$
```

Great now you will end up with a file "**initramfs**" which also includes the **uboot header** and this file we will be used as ram based file system whenever required.