**Guide to Yocto Build Process**

Index

# YOCTO setup , generate image and Deploy QEMU

1. **Procedure to add binary utility in Busy box**

# Procedure to add custom applications in Yocto build system

1. **Procedure to create custom shared library and library dependent application**

# Procedure to change Kernel version in Yocto build system

1. **Procedure to change kernel configuration with menuconfig**

# Procedure to apply the kernel patch to existing kernel

1. **Procedure to add custom kernel module to the root file system**

# **Yocto Setup and Image building**

# **$ sudo apt update**

# **$ sudo apt install git git-lfs tar python3 python3-pip gcc**

# **$ sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm python3-subunit mesa-common-dev zstd liblz4-tool**

# **$ sudo apt install make python3-pip**

# **$ sudo pip3 install sphinx sphinx\_rtd\_theme pyyaml**

# **$ mkdir yocto**

# **$ cd yocto**

# **$ git clone git://git.yoctoproject.org/poky -b mickledore**

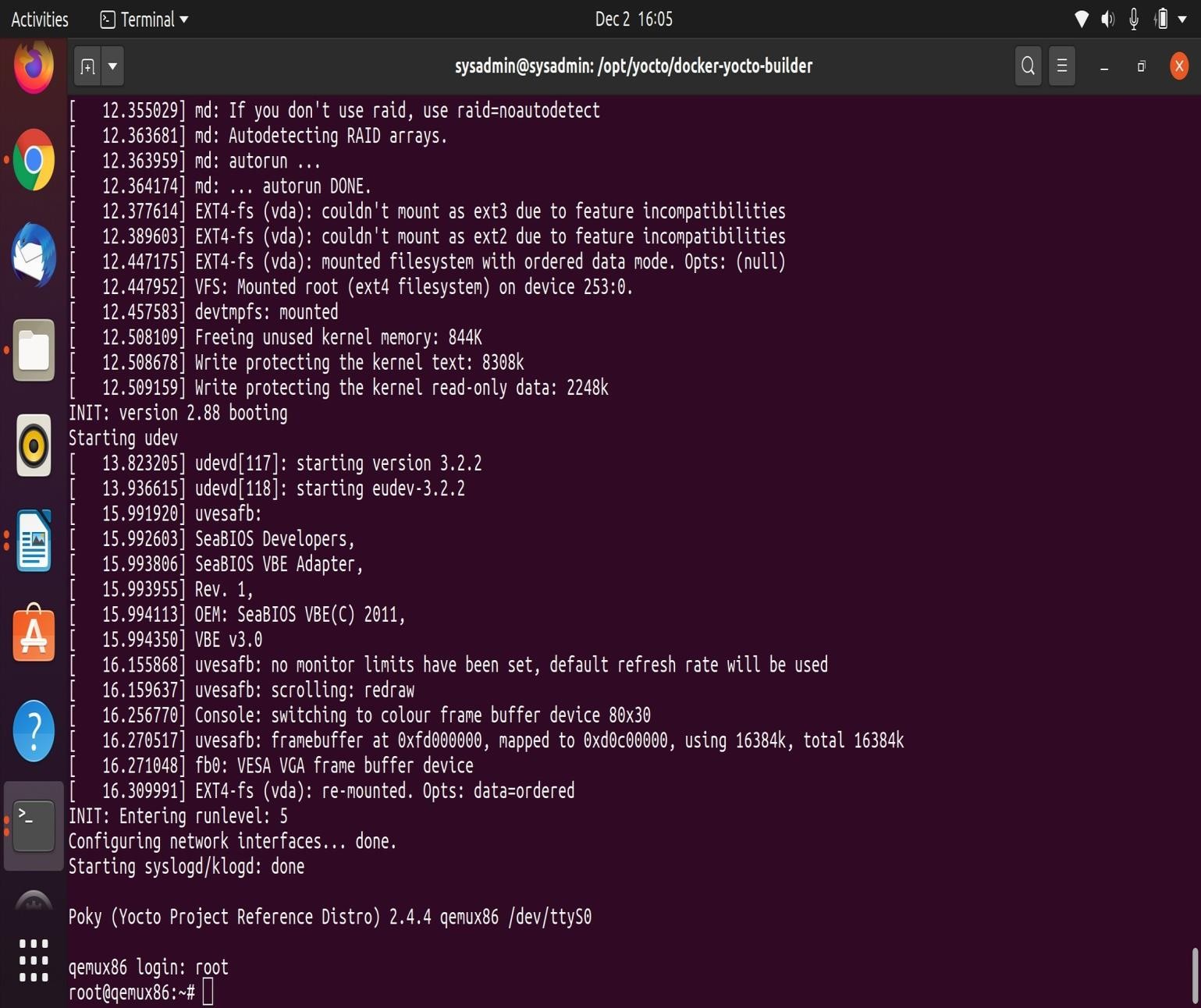
# **$ cd poky**

# **$ source oe-init-build-env**

# **$ bitbake core-image-sato --runall=fetch**

# **$ bitbake core-image-sato**

# **$ runqemu qemux86-64 nographic**



# Procedure to add binary utility in Busy box

1. **STEPS TO ADD BINARY UTILITY IN BUSY BOX**

# Procedure to add tftpd utility to busybox:

* 1. For adding the tftpd utility into the busybox first we found out where the configurations and patch files located in the system.
  2. For finding where the configurations files located for busybox follow the below steps
     1. Issue below commands to get configurations

$bitbake busybox -c menuconfig

$bitbake busybox -c diffconfig

**/home/rama/yocto/poky/build/tmp/work/core2-64-poky-linux/busybox/1.31.1-r0/ fragment.cfg**

$bitbake -e busybox | grep ^B= //Gives the path where busybox located

3 Get into the directory **opt/yocto/poky/meta/recipes-core/busybox/busybox/**

**cp /home/rama/yocto/poky/build/tmp/work/core2-64-poky-linux/busybox/1.31.1-r0/fragment.cfg enable-tftpd.cfg**

1. Locate the .bb file for the busybox to include this configuration file we created for tftpd.Then we need to add the the below line in the /**opt/yocto/poky/meta/recipes-core/busybox/ busybox\_1.24.1.bb** file

**SRC\_URI += "file://enable-tftpd.cfg"**

1. Once the above steps are done we need to build the core-minimal image for that execute the below commands.

$source /opt/yocto/poky/oe-init-build-env

$bitbake core-image-sato

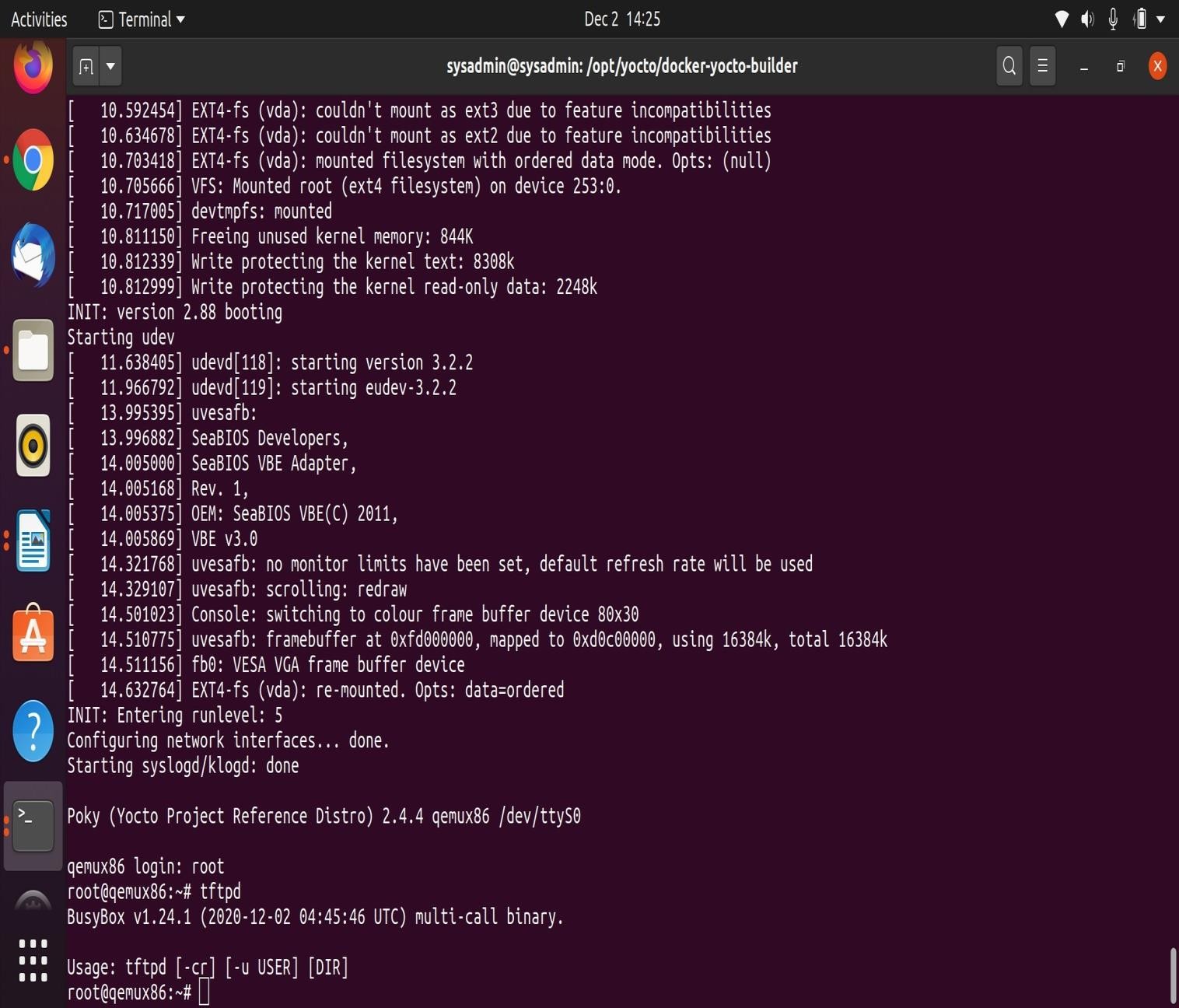
1. Once qemu is up and running try to check tftpd command working or not by given below command.

root@qemux86~# tftpd

**To clean busy box:**

$bitbake busybox -f -c cleanall

$bitbake core-image-sato



# Procedure to add custom applications in Yocto build system

1. **STEPS FOR ADDING THE CUSTOM APPLICATION TOYOCTO**
2. Here we have added the calculator application to the qemu core-image-minimal by creating the meta-layer in yocto framework.
3. For creating the meta layer first move to the poky directory inside the yocto source code and create a directory with **meta-layername**.

Execute the below commands for creating the meta-directory container@~cd /opt/yocto/poky/

container@~mkdir meta-example container@~cd meta-example

1. Once we have created the meta-example directory then we need to add the configurations required to the meta layer we have created.
   1. Create the conf directory and create a file layer.conf insideit. This will define the paths to the recipes.

Execute the below commands for creating the layer.conf and copy the below content into it and save it.

conatiner@~/opt/yocto/poky/meta-example$mkdir conf conatiner@~/opt/yocto/poky/meta-example/conf $vi layer.conf

layer.conf

# We have a conf and classes directory, add to BBPATH BBPATH .= “:${LAYERDIR}”

# We have recipes-\* directories, add to BBFILES BBFILES += “${LAYERDIR}/recipes-\*/\*/\*.bb \

${LAYERDIR}/recipes-\*/\*/\*.bbappend \

${LAYERDIR}/recipes-\*/\*.bb \

${LAYERDIR}/recipes-\*/\*.bbappend ”

BBFILE\_COLLECTIONS += “example”

BBFILE\_PATTERN\_example = “^${LAYERDIR}/” BBFILE\_PRIORITY\_example = “1”

LAYERVERSION\_example = “1”

1. Now we need to create the recipe folder and we will add the calculator recipe to that follow the below steps for creating the recipe.

conatiner@~/opt/yocto/poky/meta-example$mkdir recipes-example conatiner@~/opt/yocto/poky/meta-example$cd recipes-example

* 1. Create a folder with the calculator (application name) and create a calculator.bb file insideit copy the below content into it and save it.

conatiner@~opt/yocto/poky/meta-example/recipes-example$mkdir calculator conatiner@~opt/yocto/poky/meta-example/recipes-example$cd calculator conatiner@~opt/yocto/poky/meta-example/recipes-example/calculator$vi calculator.bb

calculator.bb

DESCRIPTION = "Example Calculator application for Yocto build." SECTION = "examples"

DEPENDS = "/opt/yocto/poky/meta-example/recipes-example/calculator/helloworld\_1.0.bb" LICENSE = "CLOSED"

FILESEXTRAPATHS\_prepend := "${THISDIR}/src:" SRC\_URI = "file://calculator.c"

S = "${WORKDIR}"

do\_compile() {

${CC} ${CFLAGS} ${LDFLAGS} ${WORKDIR}/calculator.c -o calculator

}

do\_install() {

# create the /usr/bin folder in the rootfs with default permissions install -d ${D}${bindir}

# install the application into the /usr/bin folder with default permissions install ${WORKDIR}/calculator ${D}${bindir}

}

* 1. Once the above commands are executed get back to the recipes-example directory and create the src(source) directory and add the calculator.c file into it and below content into it.

conatiner@~opt/yocto/poky/meta-example/recipes-example$mkdir src conatiner@~opt/yocto/poky/meta-example/recipes-example/src$vi calculator.c

calculator.c

#include <stdio.h> int main()

{

int num1,num2;

float result;

char ch; //to store operator choice

printf("Enter first number: "); scanf("%d",&num1); printf("Enter second number: "); scanf("%d",&num2);

printf("Choose operation to perform (+,-,\*,/,%): "); scanf(" %c",&ch);

result=0; switch(ch)

{

case '+':

result=num1+num2; break;

case '-':

result=num1-num2; break;

case '\*':

result=num1\*num2; break;

case '/':

result=(float)num1/(float)num2; break;

case '%': result=num1%num2; break;

default:

printf("Invalid operation.\n");

}

printf("Result: %d %c %d = %f\n",num1,ch,num2,result); return 0;

}

1. Once we have added the content for the recipes-example we need to append our application to core minimal image for that we will create recipes-image folder and add the configuration to it.

conatiner@~opt/yocto/poky/meta-example$mkdir recipes-image conatiner@~opt/yocto/poky/meta-example$cd recipes-image

* 1. Create the file name with core-image-minimal.bbappend and the below configuration into it. conatiner@~opt/yocto/poky/meta-example/recipes-image$ vi core-image-minimal.bbappend

core-image-minimal.bbappend IMAGE\_INSTALL \_append = "calculator"

1. Once the above steps are done we need to source this will take us into build folder. conatiner@~opt/yocto/poky$source oe-init-build-env
2. When we are in the build folder we need to add our custom layer to the bblayers.conf for that need to add path of our new layer in conf/bblayers.conf

conatiner@~opt/yocto/poky/build$cd conf conatiner@~opt/yocto/poky/build/conf$vi bblayers.conf

bblayers.conf

POKY\_BBLAYERS\_CONF\_VERSION is increased each time build/conf/bblayers.conf # changes incompatibly

POKY\_BBLAYERS\_CONF\_VERSION = "2"

BBPATH = "${TOPDIR}" BBFILES ?= ""

BBLAYERS ?= " \

/opt/yocto/poky/meta \

/opt/yocto/poky/meta-poky \

/opt/yocto/poky/meta-yocto-bsp \

/opt/yocto/poky/meta-example \ >(added this line)

"

1. Move back to the build folder and build the image using the bitbake command conatiner@~opt/yocto/poky/build$bitbake core-image-minimal
2. Once the minimal image is built for the qemu run the command and login as root and check whether our application is added or not

conatiner@~opt/yocto/poky/build$runqemu qemux86 nographic

# 4.Procedure to create custom shared library and library dependent application

**IV.STEPS FOR ADDING THE CUSTOM SHARED LIBRARY TO YOCTO AND CUSTOM APPLICATION ACCESSING THOSE FUNCTIONS**

1. Here we have added the libsample shared library, shared header and sample application that calling those functions to the qemu core-image-minimal by creating the meta-layer in yocto framework.
2. For creating the meta layer first move to the poky directory inside the yocto source code and create a directory with **meta-layername**.

Execute the below commands for creating the meta-directory container@~cd /opt/yocto/poky/

container@~mkdir meta-example container@~cd meta-example

1. Once we have created the meta-example directory then we need to add the configurations required to the meta layer we have created.
   1. Create the conf directory and create a file layer.conf insideit. This will define the paths to the recipes.

Execute the below commands for creating the layer.conf and copy the below content into it and save it.

layer.conf

conatiner@~/opt/yocto/poky/meta-example$mkdir conf conatiner@~/opt/yocto/poky/meta-example/conf $vi layer.conf

# We have a conf and classes directory, add to BBPATH BBPATH .= “:${LAYERDIR}”

# We have recipes-\* directories, add to BBFILES BBFILES += “${LAYERDIR}/recipes-\*/\*/\*.bb \

${LAYERDIR}/recipes-\*/\*/\*.bbappend \

${LAYERDIR}/recipes-\*/\*.bb \

${LAYERDIR}/recipes-\*/\*.bbappend ”

BBFILE\_COLLECTIONS += “example”

BBFILE\_PATTERN\_example = “^${LAYERDIR}/” BBFILE\_PRIORITY\_example = “1”

LAYERVERSION\_example = “1”

1. Now we need to create the recipe folder and we will add the recipes for custom library and application to that follow the below steps for creating the recipe.

conatiner@~/opt/yocto/poky/meta-example$mkdir recipes-example conatiner@~/opt/yocto/poky/meta-example$cd recipes-example

* 1. Create a two folder’s with the libsample and sampleapp (application’s name) and enter into libsample where we will create recipes for our shared library and shared header.

conatiner@~opt/yocto/poky/meta-example/recipes-example$mkdir libsample conatiner@~opt/yocto/poky/meta-example/recipes-example$cd libsample

* 1. Next we will create a recipe for the shared library and shared header for that create a libsample.bb file and copy the below content into it save and exit.

conatiner@~opt/yocto/poky/meta-example/recipes-example/libsample$vi libsample.bb

libsample.bb #

# This file is the libsample recipe.

#

SUMMARY = "Simple libsample application" SECTION = "libs"

LICENSE = "CLOSED"

SRC\_URI = "file://libsample.c \ file://libsample.h \ file://Makefile "

S = "${WORKDIR}"

PACKAGE\_ARCH = "${MACHINE\_ARCH}"

PROVIDES = "sample"TARGET\_CC\_ARCH += "${LDFLAGS}"

do\_install() {

install -d ${D}${libdir} install -d ${D}${includedir}

oe\_libinstall -so libsample ${D}${libdir}

install -d -m 0655 ${D}${includedir}/SAMPLE

install -m 0644 ${S}/\*.h ${D}${includedir}/SAMPLE/

}

FILES\_${PN} = "${libdir}/\*.so.\*

${includedir}/\*" FILES\_${PN}-dev = "${libdir}/\*.so"

* 1. Once the above commands are executed get back to the libsample directory and create the files(source) directory and add the libsample.c libsample.h and Makefile file’s into it and copy the below corresponding file content into it.

conatiner@~opt/yocto/poky/meta-example/recipes-example/libsample$mkdir files conatiner@~opt/yocto/poky/meta-example/recipes-example/libsamples$cd files conatiner@~opt/yocto/poky/meta-example/recipes-example/libsamples/files$vi libsample.c

libsample.c #include <stdio.h>

#include "libsample.h"

void function()

{

printf("Hello World!\n");

}

void samplelib()

{

printf("Hello From Global edge!!! -- samplelib !\n");

}

* + 1. Create a libsample.h file by below command copy the below content save and exit. conatiner@~opt/yocto/poky/meta-example/recipes-example/libsamples/files$

libsample.h

void samplelib(); void function();

* + 1. Create a Makefile file by below command copy the below content save and exit. conatiner@~opt/yocto/poky/meta-example/recipes-example/libsamples/files$vi Makefile

Makefile

LIBSOURCES=\*.c OUTS = \*.o

NAME := sample MAJOR = 1.0

MINOR = 1

VERSION = $(MAJOR).$(MINOR)

all: lib$(NAME).so lib$(NAME).so.$(VERSION): $(OUTS)

$(CC) $(LDFLAGS) $(OUTS) -shared -Wl,-soname,lib$(NAME).so.$(MAJOR) -o lib$ (NAME).so.$(VERSION)

lib$(NAME).so: lib$(NAME).so.$(VERSION)

rm -f lib$(NAME).so.$(MAJOR) lib$(NAME).so

ln -s lib$(NAME).so.$(VERSION) lib$(NAME).so.$(MAJOR) ln -s lib$(NAME).so.$(MAJOR) lib$(NAME).so

%.o: %.c

$(CC) $(CFLAGS) -c -fPIC $(LIBSOURCES)

clean:

rm -rf \*.o \*.so \*.so.\*

1. Once we have created the recipes for the library now we will create a folder for the sample application and add recipes for that sampleapp inside it.

Get back to the recipes-example folder and execute the below commands

conatiner@~opt/yocto/poky/meta-example/recipes-example$mkdir sampleapp conatiner@~opt/yocto/poky/meta-example/recipes-example$cd sampleapp

* 1. Once we are in the sampleapp directory we will create the recipe (.bb) file for our application execute the below commands and copy the content into sampleapp.bb file save and exit.

conatiner@~opt/yocto/poky/meta-example/recipes-example/sampleapp$vi sampleapp.bb sampleapp.bb

#

# This file is the sampleapp recipe. #

SUMMARY = "Simple sampleapp application" SECTION = "sample apps"

LICENSE = "CLOSED"

SRC\_URI = "file://helloworld.c" S = "${WORKDIR}"

DEPENDS = "libsample" do\_compile() {

${CC} ${CFLAGS} ${LDFLAGS} -o hello helloworld.c -lsample

}

do\_install() {

install -d ${D}${bindir}

install -m 0755 hello ${D}${bindir}

}

FILES\_${PN} += "helloworld"

* 1. Once we have created the recipe for the sample application we need to add the source files required for that we will create a directory with name files and we will add source files into it.

conatiner@~opt/yocto/poky/meta-example/recipes-example/sampleapp$mkdir files conatiner@~opt/yocto/poky/meta-example/recipes-example/sampleapp$cd files

* + 1. Create a helloworld.c file and add the below content into it after that save and exit conatiner@~opt/yocto/poky/meta-example/recipes-example/sampleapp/files$vi helloworld.c

helloworld.c

#include <stdio.h>

#include <SAMPLE/libsample.h> int main(void)

{

function(); samplelib(); return 0;

}

1. Once we have added the content for the recipes-example we need to append our application’sto core minimal image for that we will create recipes-image folder and add the configuration’s to it.

conatiner@~opt/yocto/poky/meta-example$mkdir recipes-image conatiner@~opt/yocto/poky/meta-example$cd recipes-image

* 1. Create the file name with core-image-minimal.bbappend and the below configuration into it. conatiner@~opt/yocto/poky/meta-example/recipes-image$ vi core-image-minimal.bbappend

core-image-minimal.bbappend

IMAGE\_INSTALL \_append = "libsample \

sampleapp"

1. Once the above steps are done we need to source this will take us into build folder. conatiner@~opt/yocto/poky$source oe-init-build-env
2. When we are in the build folder we need to add our custom layer to the bblayers.conf for that need to add path of our new layer in conf/bblayers.conf

conatiner@~opt/yocto/poky/build$cd conf conatiner@~opt/yocto/poky/build/conf$vi bblayers.conf

bblayers.conf

POKY\_BBLAYERS\_CONF\_VERSION is increased each time build/conf/bblayers.conf # changes incompatibly

POKY\_BBLAYERS\_CONF\_VERSION = "2"

BBPATH = "${TOPDIR}" BBFILES ?= ""

BBLAYERS ?= " \

/opt/yocto/poky/meta \

/opt/yocto/poky/meta-poky \

/opt/yocto/poky/meta-yocto-bsp \

/opt/yocto/poky/meta-example \ >(added this line)

"

1. Move back to the build folder and build the image using the bitbake command conatiner@~opt/yocto/poky/build$bitbake core-image-minimal
2. Once the minimal image is built for the qemu run the command and login as root and check whether our application is added or not

conatiner@~opt/yocto/poky/build$runqemu qemux86 nographic

1. Once the qemu is up and running run our application by typing hello in cmd.

# 5. Procedure to change Kernel version in Yocto build system

1. **STEPS TO CHANGE KERNEL VERSION IN YOCTO FRAMEWORK**
2. The configuration file poky/meta/conf/machine/include/qemu.inc has the following lines # Use a common kernel recipe for all QEMU machines

PREFERRED\_PROVIDER\_virtual/kernel ?= "linux-yocto"

1. The above line directs bitbake to look for kernel recipes under path

**poky/meta/recipes-kernel/linux/**

In the current Dockerfile we have cloned yocto-poky “rocko” branch provifing the following kernel recepies

linux-yocto-tiny\_4.4.bb linux-yocto-tiny\_4.9.bb linux-yocto-tiny\_4.10.bb linux-yocto-tiny\_4.12.bb

1. By design bitbake will use linux-yocto\_4.12.bb, i.e., 4.12 kernel will be built for machine qemux86. That is because, bitbake is designed to prefer the highest recipe version.
2. In the configuration file(**/opt/yocto/poky/meta-poky/conf/distro/poky.conf**) we modified version from 4.12 to 4.10.

**PREFERRED\_VERSION\_linux-yocto = "4.10%"**

Modifying this line will make bitbake use **poky/meta/recipes-kernel/linux/linux-yocto\_4.10.bb recipe**, i.e., 4.10 kernel for qemux86.

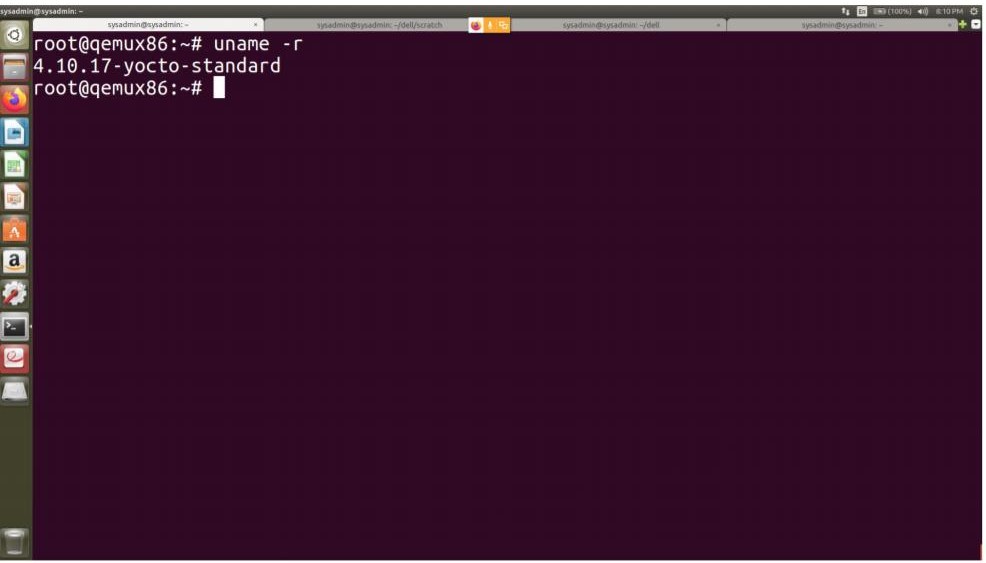
1. Once the above steps are done we need to generate the core-minimal image for that execute the below commands

conatiner@~$source /opt/yocto/poky/oe-init-build-env conatiner@~$MACHINE=qemux86 bitbake core-image-minimal

1. Once core minimal image is generated for the qemux86 run the image with the below command and login as root

container@~$runqemux86 runqemu nographic

1. Once you logged in as root u can check the kernel version by below command. root@qemu~#uname -r



# Procedure to change kernel configuration with menuconfig

1. **STEPS FOR KERNEL CONFIGURATION IN YOCTO**

\* create the dockerfile with all the necessory dependency for yocto and also write a command to install the screen package.

Add the below line in Dockerfile,

# RUN apt-get install screen

Where,

**screen** – helps in launching a new devshell for opening the menuconfig window.

* Once the Dockerfile is created, Build the docker image by using that docker file.
* After building the docker image run it in a container.
* Once you are in container source the oe-init-build- env script and run the bitbake menuconfig. To achieve the above follow the below commands in the container,

**container@~** source /opt/yocto/poky/oe-init-build-env

**container@~** bitbake linux-yocto -c menuconfig

Where,

**linux-yocto**: is the name of kernel recipes(.bb) files which you can find them in

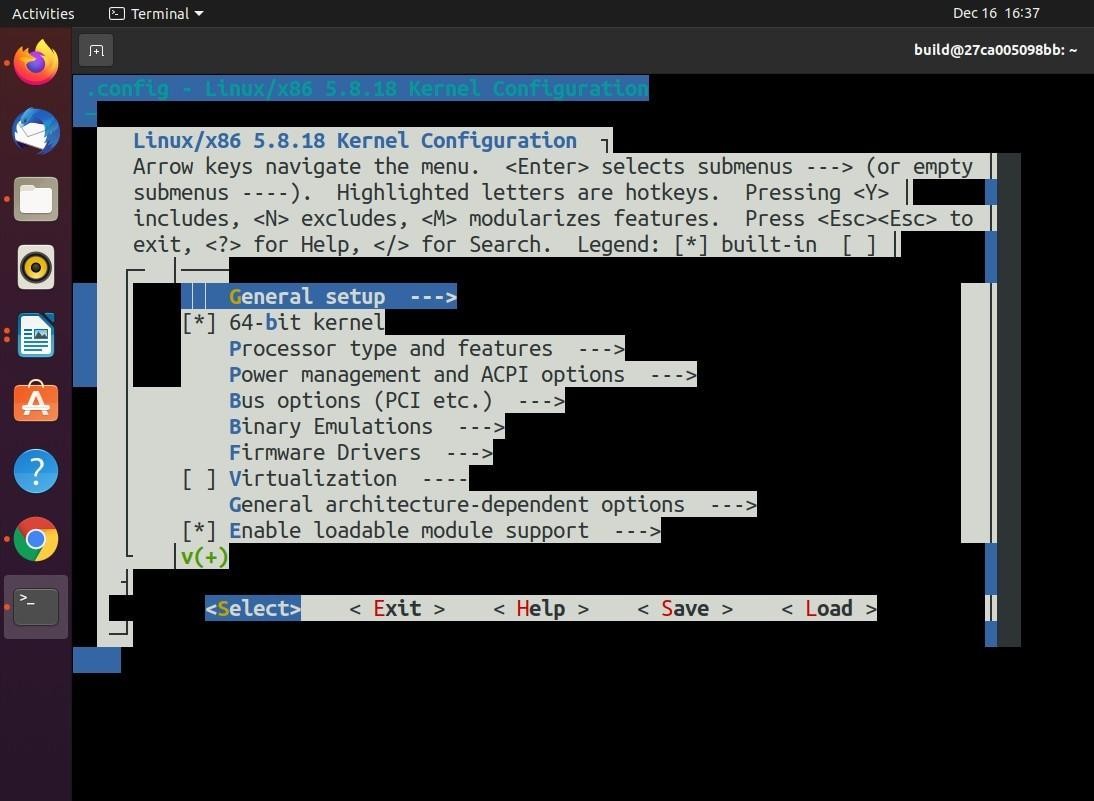
**/opt/yocto/poky/meta/recipes-kernel/linux** directory. In my case it is **linux-yocto.**

To find the name of kernel execute the below command,

**container@~** bitbake -e virtual/kernel | grep “^PN”

Where it dispays as PN=”name\_of\_kernel” which you can use in bitbake menuconfig.

* Once you execute the above commands, bitbake fetches the kernel source for the first time, If it is not the first time it will simply fetches the kernel source which has been fetched before. And then opens a devshell with menuconfig window as in the below image



* After this window you can configure the kernel with the configuration you want.
* After configuring the kernel you can execute the below command to build the images for a particular machine.

After this move to ~/build directory in the container and execute below command,

**container@~** MACHINE=qemux86-64 bitbake core-image-minimal Where, **MACHINE** is the actual hardware architecture you are building the images for. In my case it is **qemux86-64**

**core-image-minimal** is the minimal image I am building for the specific machine, But we can also use core-image-sato, core- image-base etc based on requirment.

* After building the images we can boot the system as below,

**conatiner@~** runqemu qemux86-64 nographic

1. **Procedure to apply the kernel patch to existing kernel**

In order to change the existing kernel recipes, first we need to create a new layer as given below,

1. go to /opt /yocto /poky directory in the yocto project and create the following directories and subdirectories,

**poky$** mkdir meta-mylayer

**poky$** mkdir meta-mylayer/conf

**poky$** mkdir meta-mylayer/recipes-kernel

**poky$** mkdir meta-mylayer/recipes-kernel/linux

**poky$** mkdir meta-mylayer/recipes-kernel/linux/linux-yocto

Where,

meta-mylayer -> The new layer created.

Just after creating the new layer (i.e. meta-mylayer in my case), add the new layer in **bblayer.conf**

file located in **~/build/conf** directory as given below,

# POKY\_BBLAYERS\_CONF\_VERSION is increased eachtime build/conf/bblayers.conf # changes incompatibly

POKY\_BBLAYERS\_CONF\_VERSION = "2" BBPATH = "${TOPDIR}"

BBFILES ?= ""

BBLAYERS ?= " \

/opt/yocto/poky/meta \

/opt/yocto/poky/meta-poky \

/opt/yocto/poky/meta-yocto-bsp \

/opt/yocto/poky/meta-mylayer" >**Added layer**

1. Create **layer.conf** in the **meta-mylayer/conf** directory with the following content,

# We have a conf and classes directory, add to BBPATH BBPATH .= ":${LAYERDIR}"

# We have recipes-\* directories, add to BBFILES

BBFILES += "${LAYERDIR}/recipes-\*/\*/\*.bb \

${LAYERDIR}/recipes-\*/\*/\*.bbappend"

BBFILE\_COLLECTIONS += "mylayer"

BBFILE\_PATTERN\_mylayer = "^${LAYERDIR}/" BBFILE\_PRIORITY\_mylayer = "5"

In order to modify the existing kernel we have to follow the below steps:

1. Create the append file
2. Apply patches
3. Changing the configuration
4. **Create the append file:**

\*We have to create a file with .bbappend extension in the **meta-mylayer/recipes-kernel/linux/**

directory with the name of the recipe you are using from **meta/recipes-kernel/linux** direcory.

Eg. In my case the recipe I am using is linux-yocto\_5.8.bb, So I have created a file with a name of linux-yocto\_5.8.bbappend in **meta-mylayer/recipes-kernel/linux/** directory with the following content,

FILESEXTRAPATHS\_prepend:="${THISDIR}/${PN}:"

SRC\_URI\_append += "0001-block-patches.patch"

# Where,

The append file should initially extend the FILESPATH search path by prepending the directory that contains your files to the FILESEXTRAPATHS.

For FILESPATH see base.bbclass in meta/classes directory.

${THISDIR}/${PN} - > expands to linux-yocto in the working directory.

0001-block-patches.patch is a patch file which I have created in **meta-mylayer/recipes- kernel/linux/linux-yocto** directory.

1. **Apply patches:**

As explained above 0001-block-patches.patch is a patch file to be applied(In my case) to the existing kernel. You can apply any number of patches by creating patch files in **meta-mylayer/recipes- kernel/linux/linux-yocto** directory and adding the file in .bbappend file as,

SRC\_URI\_append += “file://created\_file.patch”

(Mentioned in above step)

Where, created\_file specifies the name of the patch file.

1. **Changing the configuration:**

execute **find . -name .config** command from ~/build directory to find the kernel configuration file.

Copy the file,

# ~/build/tmp/work/qemux86\_64-poky-linux/linux-yocto/ 5.8.18+gitAUTOINC+7883b60d32\_3c5d210805-r0/linux-qemux86\_64- standard-build/.config to defconfig file created in the meta-mylayer/recipes-kernel/linux/linux-yocto directory.

**Build the Kernel:**

After doing all the above steps rebuild the images you want i.e. core- image-minimal, core-image- sato, core-image-base etc.

**container@~** MACHINE=qemux86-64 bitbake core-image-minimal

Now you can go to the ~/build/tmp/work-shared/qemux86-64/kernel- source/ directory and check wether the patches are applied or not based on to what source code of the kernel you applied the patches.

Where, qemux86-64 is the machine I built the images for, It should be the machine you are building the images for.

# Procedure to add custom kernel module to the root file system

**VIII. STEPS FOR ADDING THE CUSTOM KERNEL MODULE TO YOCTO FRAMEWORK**

1. Here we are adding the a simple helloworld.ko module to the qemu core-image-minimal by creating the meta layer in the yocto framework.
2. For creating the meta layer first move to the poky directory inside the yocto source code and create a directory with **meta-layername**.

Execute the below commands for creating the meta-directory container@~cd /opt/yocto/poky/

container@~mkdir meta-example container@~cd meta-example

1. Once we have created the meta-example directory then we need to add the configurations required to the meta layer we have created.
   1. Create the conf directory and create a file layer.conf insideit. This will define the paths to the recipes.

Execute the below commands for creating the layer.conf and copy the below content into it and save it.

conatiner@~/opt/yocto/poky/meta-example$mkdir conf conatiner@~/opt/yocto/poky/meta-example/conf $vi layer.conf

layer.conf

# We have a conf and classes directory, add to BBPATH BBPATH .= “:${LAYERDIR}”

# We have recipes-\* directories, add to BBFILES BBFILES += “${LAYERDIR}/recipes-\*/\*/\*.bb \

${LAYERDIR}/recipes-\*/\*/\*.bbappend \

${LAYERDIR}/recipes-\*/\*.bb \

${LAYERDIR}/recipes-\*/\*.bbappend ”

BBFILE\_COLLECTIONS += “example”

BBFILE\_PATTERN\_example = “^${LAYERDIR}/” BBFILE\_PRIORITY\_example = “1”

LAYERVERSION\_example = “1”

1. Now we need to create the recipe folder and we will add the kernel module recipe to that follow the below steps for creating the recipe.

conatiner@~/opt/yocto/poky/meta-example$mkdir recipes-kernel conatiner@~/opt/yocto/poky/meta-example$cd recipes-kernel

* 1. Create a folder with the hello-mod and create a hello-mod.bb file inside it copy the below content into it and save it.

conatiner@~opt/yocto/poky/meta-example/recipes-example$mkdir hello-mod conatiner@~opt/yocto/poky/meta-example/recipes-example$cd hello-mod conatiner@~opt/yocto/poky/meta-example/recipes-example/hello-mod$vi hello-mod.bb

hello-mod.bb

SUMMARY = "Example of how to build an external Linux kernel module" LICENSE = "GPLv2"

LIC\_FILES\_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e" inherit module

SRC\_URI = "file://Makefile \ file://hello.c \ file://COPYING \

"

S = "${WORKDIR}"

* 1. Once the above commands are executed get back to the hello-mod directory and create the files(source) directory and add the hello-mod.c and Makefile and COPYING license file into it and copy below content into those files.

conatiner@~opt/yocto/poky/meta-example/recipes-example/hello-mod$mkdir files conatiner@~opt/yocto/poky/meta-example/recipes-example/hello-mod/files$vi hello-mod.c

hello-mod.c

#include <linux/module.h> int init\_module(void)

{

printk("Hello World!\n"); return 0;

}

void cleanup\_module(void)

{

printk("Goodbye Cruel World!\n");

}

MODULE\_LICENSE("GPL");

* + 1. Create the Makefile and by executing the below command and copy the below content into it save and exit.

conatiner@~opt/yocto/poky/meta-example/recipes-example/hello-mod/files$vi Makefile Makefile

obj-m := hello-mod.o SRC := $(shell pwd)

all:

$(MAKE) -C $(KERNEL\_SRC) M=$(SRC)

modules\_install:

$(MAKE) -C $(KERNEL\_SRC) M=$(SRC) modules\_install

clean:

rm -f \*.o \*~ core .depend .\*.cmd \*.ko \*.mod.c

rm -f Module.markers Module.symvers modules.order rm -rf .tmp\_versions Modules.symvers

* + 1. Create a license file by executing the below command and copy the content into it and save it. conatiner@~opt/yocto/poky/meta-example/recipes-example/hello-mod/files$vi COPYING.MIT COPYING.MIT

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

1. Once we have added the content for the recipes-example we

need to append our application to core minimal image for that we will create recipes-image folder and add the configuration to it.

conatiner@~opt/yocto/poky/meta-example$mkdir recipes-image conatiner@~opt/yocto/poky/meta-example$cd recipes-image

5.1 Create the file name with core-image-minimal.bbappend and the below configuration into it. conatiner@~opt/yocto/poky/meta-example/recipes-image$ vi core-image-minimal.bbappend

core-image-minimal.bbappend IMAGE\_INSTALL \_append = "hello-mod"

1. Once the above steps are done we need to source this will take us into build folder. conatiner@~opt/yocto/poky$source oe-init-build-env
2. Now we need to make some changes to the local.conf file open the local.conf file and add the below line at end of the file save and exit.

conatiner@~opt/yocto/poky/build/conf$vi local.conf local.conf

MACHINE\_ESSENTIAL\_EXTRA\_RRECOMMENDS += "hello-mod"

1. Now we need to add our custom layer to the bblayers.conf for that need to add path of our new layer in conf/bblayers.conf

conatiner@~opt/yocto/poky/build$cd conf conatiner@~opt/yocto/poky/build/conf$vi bblayers.conf

bblayers.conf

POKY\_BBLAYERS\_CONF\_VERSION is increased each time build/conf/bblayers.conf # changes incompatibly

POKY\_BBLAYERS\_CONF\_VERSION = "2"

BBPATH = "${TOPDIR}" BBFILES ?= ""

BBLAYERS ?= " \

/opt/yocto/poky/meta \

/opt/yocto/poky/meta-poky \

/opt/yocto/poky/meta-yocto-bsp \

/opt/yocto/poky/meta-example \ >(added this line)

"

1. Move back to the build folder and build the image using the bitbake command conatiner@~opt/yocto/poky/build$bitbake core-image-minimal

Once the minimal image is built for the qemu run the below command and login as root and check whether our module is added or not.

conatiner@~opt/yocto/poky/build$runqemu qemux86 nographic

1. Once the qemu is up and running check whether our module is added to the image or not by running below command.

root@qemux86:~#cd /lib/modules/4.12.28-yocto-standard/extra/ root@qemux86:/lib/modules/4.12.28-yocto-standard/extra# ls

# 9. SETTING UP THE YOCTO ON UBUNTU MACHINE

1. First intsall all the dependencies required for the yocto setup by running the below command. sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential

chrpath socat libsdl1.2-dev xterm

1. The clone the required branch of the yocto project using the below command here we have cloned the dunfell branch if you havent mentioned any branch the master brach will be cloned.

git clone -b dunfell <http://git.yoctoproject.org/git/poky>

1. Now move to the poky directory and execute the below commads it will create the core-image- minimal

user@~$source /opt/yocto/poky/oe-init-build-env (For 32 bit)

user@~$MACHINE=qemux86 bitbake core-image-minimal

(For 64 bit)

user@~$MACHINE=qemux86-64 bitbake core-image-minimal