Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL5841 Taller de Sistemas Embebidos
II Semestre 2021

TEC | Tecnológico de Costa Rica

Tutorial OpenCV con Yocto project

Manuel Arias Aburto, Susana Astorga Rodríguez, Jose Angulo Durán, Gerardo Araya Fallas

1. Introducción

Yocto Project es un desarrollador de software que tiene por objetivo la producción de herramientas y procesos que permitan la creación de distribuciones de Linux para software embebido y IoT que sean independientes de la arquitectura subyacente del hardware embebido [1]; por lo que en este tutorial se encontrará el proceso de creación de una imagen mínima de Linux a la medida con Yocto Project para correr una aplicación de visión por computador con la biblioteca OpenCV. Además, se mostrará el proceso para ejecutar la imagen de Linux implementada sobre una máquina virtual con arquitectura x86 en VirtualBox. Cabe recalcar que este tutorial fue realizado mediante el sistema operativo Ubuntu en su versión 20.04.

2. Características del host

- OS: Windows 10 Pro 64-bit.
- CPU: AMD Ryzen 7 3700X @ 4.4 GHz.
- RAM: 16 GB.
- HDD de 1 TB y Windows instalado en un SSD de 250 GB.
- Máquina virtual con Ubuntu 20.04.2 LTS de host.
- 200 GB de HDD asignados a la máquina virtual.
- 8 de 16 procesadores lógicos asignados a la máquina virtual.
- 11 GB de RAM asignados a la máquina virtual.

3. Aplicación de OpenCV a mostrar

La aplicación de OpenCV seleccionada consiste en un código sencillo que utiliza la librería OpenCV de python para generar un boceto a lápiz a partir de una imagen a color, esta aplicación es llamada *pencilSketch* y el código se puede observar en [2].

Para la creación de un boceto a lápiz a partir de una imagen a color la aplicación realiza una serie de procesos y transformaciones necesarias a la imagen, por lo que el primer paso a realizar consiste en leer la imagen de entrada como imagen en escala de grises usando el valor 0 en *cv2.imread* para cargar una imagen en escala de grises.

El segundo paso consiste en eliminar cualquier ruido y reducir la cantidad de detalles en la imagen utilizando un filtro gaussiano 3x3.

En el tercer paso se procede a crear una imagen en negativo, por lo que se "invierte" el valor de la escala de grises (que va de un rango [0,255]) de cada píxel, es decir, se hace 255-x donde x para este caso es el valor obtenido una vez aplicado el filtro gaussiano.

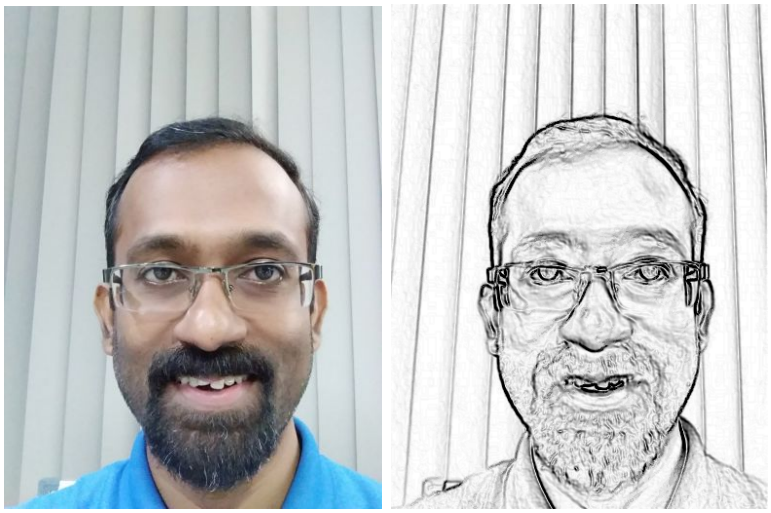
Para el cuarto paso se detecta los bordes de la imagen de entrada y su negativo usando la función sobel, en donde la función sobel detecta bordes usando sobel kernel en la cual detecta los bordes horizontales y los bordes verticales de la imagen.

El quinto paso consiste en fusionar ambas imágenes de bordes horizontales y verticales usando *cv2.bitwise_or*, estas dos imágenes refuerzan algunos bordes y complementan otros y alternativamente estas dos imágenes pueden ser mezcladas con diferentes pesos usando *cv2.addWeighted*.

En el sexto paso se procede a invertir la imagen mezclada para obtener un boceto negro sobre fondo blanco.

Y como último paso se muestra la imagen resultado y se espera a que se pulse una tecla cualquiera para cerrar la ventana.

Para esta aplicación se utilizaron dos imágenes para obtener diferentes resultados, por lo que el resultado de estas imágenes se muestran a continuación en las figuras 1 y 2:



(a) Imagen original (b) Imagen pencilSketch

Figura 1: Resultado 1 de la aplicación pencilSketch

(a) Imagen original (b) Imagen pencilSketch

Figura 2: Resultado 2 de la aplicación pencilSketch

4. Descripción del flujo de trabajo de Yocto project

4.1. Instalación de Yocto project

La instalación de Yocto Proyect se fundamentó en la información recopilada de la página de la Fundación Linux [3]. Al trabajar con Yocto Project hay que tener en cuenta y hacer una distinción entre el "host" de la computadora el cual es donde se generará la imagen y el "target" es cual es donde se correrá la imagen.
Por lo que para que Yocto pueda ejecutarse en cualquier sistema Linux se deben instalar las siguientes dependencias haciendo uso de la terminal:

- Git 2.25.1 o mayor.
- Tar 1.30 o mayor.

- Python 2.7.18 o mayor.
- gcc 9.3.0 o mayor.

Por lo que para descargar la dependencia de Git se debe escribir *sudo apt-get install git* en la terminal y para instalar la dependencia Python se debe escribir *sudo apt-get install python* en la terminal.
Las versiones recientes de Linux Ubuntu ya tiene instalado tar y gcc, pero es recomendable su verificación, para ello se escribe en la terminal *tar –version* y *gcc –version*.

4.2. Preparación del host

Luego de instalar las anteriores dependencias, se debe instalar una lista de paquetes necesarios que necesita el host y que Yocto Proyect usa para trabajar en el entorno Ubuntu de trabajo, por lo que se escribe en la terminal lo siguiente:

```
$sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
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
```

Yocto project tiene su propio repositorio en línea, por lo que el siguiente paso sería clonar el repositorio Poky en el host, para lo cual se ejecuta el siguiente comando en la terminal que nos permitirá descargar una copia del directorio:

```
$git clone git://git.yoctoproject.org/poky
```

4.3. Creación de una imagen

Para crear una imagen en una máquina en particular es necesario especificar el target con el que se va a trabajar, para este caso se va a trabajar con el target *qemux86* debido a requerimientos del proyecto. Por lo que antes de crear la imagen se deberá seleccionar el tipo de target con el cual crear la imagen, para esto se debe ingresar al archivo *local.conf* el cual se encuentra en la carpeta principal *build* dentro de la carpeta *conf*. Una vez dentro de este archivo este mostrará una selección de máquinas disponibles por lo que de esta selección vamos a utilizar la arquitectura qemux86 por lo que descomentamos la línea *MACHINE ?= "qemux86"* la cual le va a indicar al target en que arquitectura necesitamos que genere la imagen.

Ahora bien, para proceder a la creación de la imagen deberemos ingresar varios comandos en la terminal del host que es donde se generará la imagen y se deberá seleccionar el tipo de arquitectura como se mencionó anteriormente. Por lo que primeramente escribiremos en la terminal para ingresar a la carpeta poky el comando:

```
$ cd poky
```

Una vez dentro de la carpeta poky para crear un directorio de compilación escribimos en la terminal:

```
$source oe-init-build-env
```

Para este caso vamos a crear una imagen mínima por lo que escribimos:

```
$bitbake core-image-minimal
```

El comando bitbake procederá a crear la imagen cabe recalcar que ess un proceso tardado y dará como resultado una imagen con una terminal. Una vez que se haya generado la imagen para ver la arquitectura de la imagen creada escribimos:

```
$ls tmp/deploy/images
```

Podemos probar ejecutar la imagen creada con el emulador QEMU por lo que escribimos en la terminal:

```
$runqemu qemux86
```

4.4. Recetas de yocto

Para que Yocto incluya la aplicación que se tiene en la imagen se debe crear una receta personalizada que incluya indicaciones de qué archivos cargar y en donde instalarlos. Lo primero que se debe hacer inicializar el ambiente de trabajo mediante:

```
$ source oe-init-build-env
```

Ahora, indicamos que vamos a crear una capa y debemos especificar donde queremos que se cree de la siguiente manera:

```
$ bitbake-layers create-layer ../meta-pencilSketch
```

De esta manera creamos la capa meta-pencilSketch en la carpeta poky y ahí debemos poner la receta creada. Mencionar la importancia de seguir la convención de nombres de capas, la cual es meta-nombre_de_capa. Una vez creada la capa debemos añadirla al archivo ../poky/build/conf/bblayers.conf, para esto tenemos dos maneras, la primera consiste en agregar la capa a mano e incluir la dirección de la capa en este archivo, la segunda forma de añadir esta capa es mediante el siguiente comando:

```
$ bitbake-layers add-layer ../meta-pencilSketch
```

El comando anterior añade de manera automática la capa para su uso. De cualquier manera que se haya seguido, es bueno corroborar que la capa se agregó de manera correcta, para esto escribimos el siguiente comando:

```
SUMMARY = "PencilSketch recipe"
DESCRIPTION = "Procesa una imagen y da como resultado la imagen dibujada a mano"
LICENSE = "CLOSED"

SRC_URI = "file://pencilSketch.py \
           file://me.jpg \
           file://villageRoad.jpg"

S = "${WORKDIR}"

do_install(){
    install -d ${D}${bindir}
    install -m 0755 pencilSketch.py ${D}${bindir}

    install -d ${D}${bindir}
    install -m 0755 me.jpg ${D}${bindir}

    install -d ${D}${bindir}
    install -m 0755 villageRoad.jpg ${D}${bindir}
}
```

Figura 3: Receta personalizada de la apicación pencilSketch

```
$ bitbake-layers show-layers
```

Una vez agregada la capa de la aplicación, procedemos a revisar los archivos que se encuentran en la carpeta meta-pencilSketch, aquí podemos dejar todo tal y como está e ingresar a la carpeta llamada recipes-example, vamos a encontrar una carpeta llamada example a la que debemos cambiar el nombre y ponerle el de nuestra aplicación. Ingresamos a la carpeta a la que cambiamos el nombre a pencilSketch y ahí encontramos un archivo llamado example_0.1.bb, a este archivo también debemos cambiarle el nombre a pencilSketch_0.1.bb. Para el caso de las recetas también debemos seguir la convención de nombres donde se establece: nombre_versión.bb.

Antes de editar nuestro nuevo archivo llamado pencilSketch_0.1.bb primero vamos a crear una carpeta llamada files aquí mismo y dentro vamos a poner todos los archivos que ocupe nuestra aplicación. En files incluimos los archivos "pencilSketch.py", "me.jpg" y "villageRoad.jpg", luego de incluir estos archivos en files, volvemos a donde teníamos el archivo pencilSketch_0.1.bb.

Dentro del .bb encontramos varias líneas que vamos a ocupar y otras que no nos interesan para nuestros fines. De lo que nos interesa tenemos las líneas SUMMARY y DESCRIPTION que, aunque no son necesarias, nos ayudan a entender de qué trata esa receta. La línea LICENCE es importante ponerla como "CLOSED" para que no nos de problemas más adelante pero en esta línea se puede poner cualquiera de las licencias disponibles como "MIT", "BSD" o "GPL", por mencionar algunas, pero esto llevaría a realizar un paso adicional al cuál no nos vamos a referir en este tutorial.

La línea SRC_URI es de suma importancia ya que indicamos cuales archivos vamos a meter en la imagen, cuando se esté construyendo la imagen, Yocto va a buscar estos archivos en la carpeta files que creamos anteriormente y por esto es que debemos crear la carpeta con ese nombre especifico, S representa el directorio de fuente de la aplicación, siempre debemos dejarlo como "${WORKDIR}". Finalmente, el do_install tiene la función de instalar los archivos en el directorio bin de la imagen creada para su uso. El archivo pencilSketch_0.1.bb debería verse como la figura 3.

Finalmente, con nuestra nueva receta creada y ubicada en su respectiva capa debemos verificar que la aplicación compile de manera correcta, para esto escribimos en la terminal la siguiente instrucción:

```
$ bitbake pencilSketch
```

Durante el proceso es normal que salga una advertencia que nos indique que como el nombre pencilSketch tiene una letra mayúscula, podría provocar algún comportamiento no deseado, sin embargo, podemos ignorar esta advertencia sin ningún problema. Una vez haya finalizado el proceso sin errores entonces sabemos que la receta está bien creada y que la capa en la que está se agregó correctamente. Ahora lo que falta es modificar el archivo local.conf, en donde debemos agregar la siguiente línea:

```
$ IMAGE_INSTALL_append = " pencilSketch"
```

Es importante dejar un espacio al abrir las primeras comillas. Con esto creamos la receta para que al crear la imagen se incluya la aplicación que queremos implementar.

4.5. Mapeo de dependencias

Para asociar los paquetes de software requeridos en el Yocto project del *meta-oe* y *meta-pencilSketch* debemos ubicarnos en el archivo *bblayers.conf* ubicado en *\poky \build \config* y deberemos de escribir las dependencias necesarias de cada una de las meta-layers, por lo que para realizar esto debemos dentro del archivo *bblayers.conf* escribir lo siguiente:

```
CORE_IMAGE_EXTRA_INSTALL += " python3"
CORE_IMAGE_EXTRA_INSTALL += " opencv"
CORE_IMAGE_EXTRA_INSTALL += " pencilSketch"
```

5. Descripción del flujo de trabajo de OpenCV

5.1. Características del toolkit

OpenCV la cual significa Open Computer Vision (Visión Artificial Abierta), es la enorme biblioteca de código abierto desarrollada por Intel en el mes de enero de 1999, para la visión por computadora, el aprendizaje automático y el procesamiento de imágenes. OpenCV está totalmente desarrollado en C++, orientado a objetos y con alta eficiencia computacional. Su API es C++ pero incluye conectores para otros lenguajes como Python, Java, Matlab, Octave y Javascript [4].

La biblioteca de OpenCV ofrece soporte para varios sistemas operativos y varias arquitecturas de hardware, pero también ofrece el código fuente para que cualquier desarrollador lo compile en cualquier sistema operativo y arquitectura particular. Además, brinda instrucciones de instalación

y opcionalmente binarios para los sistemas operativos de GNU/Linux, Windows, MacOs, Android y iOS [5].

Esta biblioteca se utiliza prácticamente en cualquier plataforma con capacidad suficiente, compilando su código fuente. De este modo se la utiliza en diversas arquitecturas de hardware como x86, x64 (PC), ARM (celulares y Raspberry Pi) [5].

En este tutorial trabajaremos con OpenCV-Python que es la API de Python para OpenCV, la cual combina las mejores cualidades de la API de C++ de OpenCV y el lenguaje Python.

5.2. Instalación de OpenCV en Yocto project

Para la inclusión de OpenCV en Yocto Project fue necesario incluir la capa meta-oe incluida en meta-openembedded, para esto primeramente debemos posicionarnos en la carpeta de Poky utilizando la terminal con el comando *cd*. Una vez ahí debemos comprobar que nos encontramos en el branch master para esto utilizamos el comando:

```
$ git checkout master
```

Y procedemos a clonar el repositorio git de meta-openembedded:

```
$ git clone git://git.openembedded.org/meta-openembedded
```

Ya teniendo esto listo procedemos a iniciar el ambiente de trabajo de bitbake con el comando:

```
$ source oe-init-build-env
```

Ahora debemos entrar a la carpeta Build/Conf donde se encuentra el archivo BBLayers.conf abriendo este archivo debemos agregar la dirección del meta-oe (Es importante mencionar que esta dirección varía de acuerdo a la ubicación del Poky y nombre de usuario), en nuestro caso quedando de esta forma:

```
BBLAYERS += " /home/linux/Documents/Git-Repositorys/TestYocto/poky
/meta-openembedded/meta-oe \"
```

Ya con la capa añadida también debemos indicar en el archivo local.conf que instale OpenCV para lo cual al final añadimos el siguiente código:

```
CORE_EXTRA_IMAGE_INSTALL = " opencv"
CORE_EXTRA_IMAGE_INSTALL = " opencv-examples"
```

Ya con esto volviendo a la terminal con el ambiente de trabajo de bitbake inicializado podemos revisar si la integración funcionó corriendo el siguiente comando:

```
$ bitbake opencv
```

5.3. Síntesis de la imagen

Para realizar la síntesis de la imagen basta con utilizar el comando:

```
$ bitbake build-appliance-image
```

Para generar un archivo .vmdk correspondiente a un disco duro virtual de Virtual Box. Este se va a crear en la carpeta poky/build/tmp/deploys/images en nuestro caso aparece con el nombre *build-appliance-image-qemux86-20210905202507.rootfs.wic.vmdk*, la última secuencia de números corresponde a la fecha por lo tanto va a variar.

6. Proceso de instalación de la imagen sintetizada en VirtualBox

Para realizar la instalación en el Virtual Box procedemos a crear una nueva máquina virtual, le asignamos un nombre y es importante seleccionar el kernel de "Other Linux 32-bits" tal y como se muestra en la siguiente imagen:

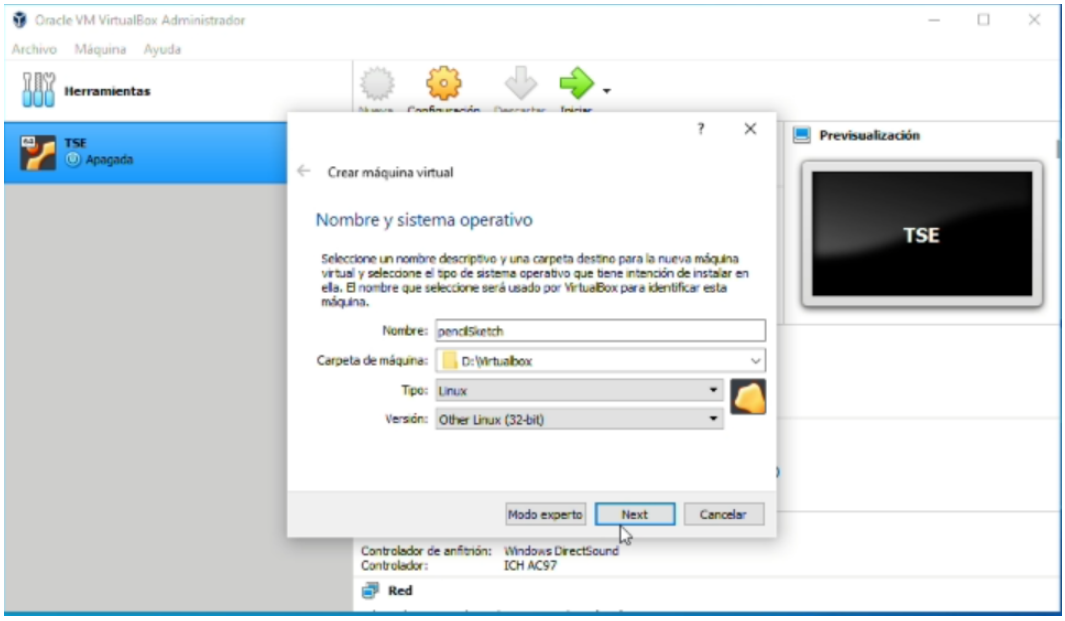


Figura 4: Configuración de la máquina virtual

Seguidamente procedemos a asignarle memoria RAM, es recomendable no pasar del indicador Verde en la linea, tal y como se muestra en la imagen:

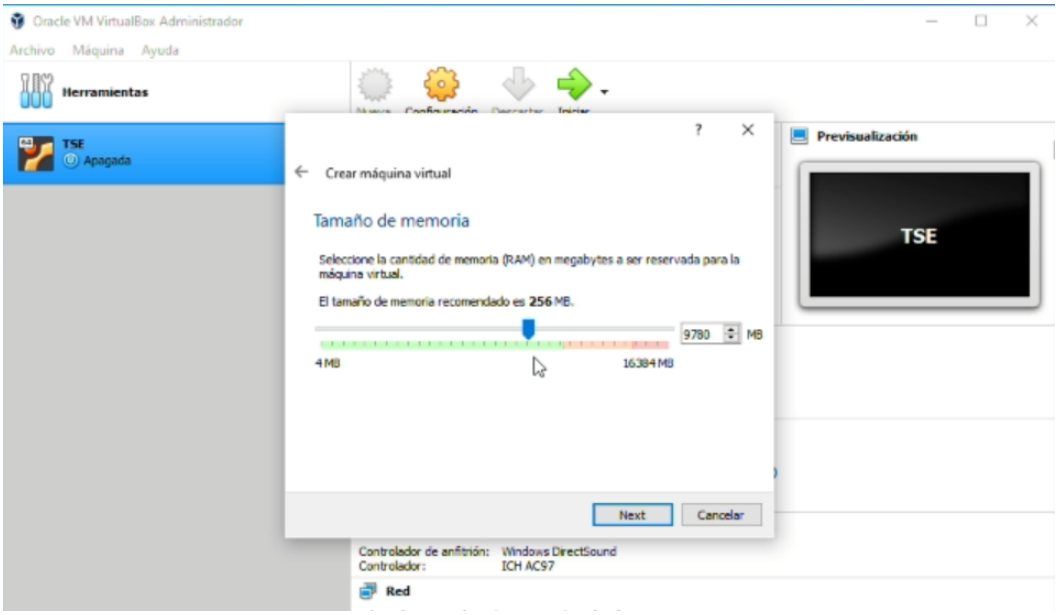


Figura 5: Asignar memoria RAM a la máquina Virtual

Luego debemos seleccionar "Usar un archivo de disco duro virtual existente" como se muestra a continuación:

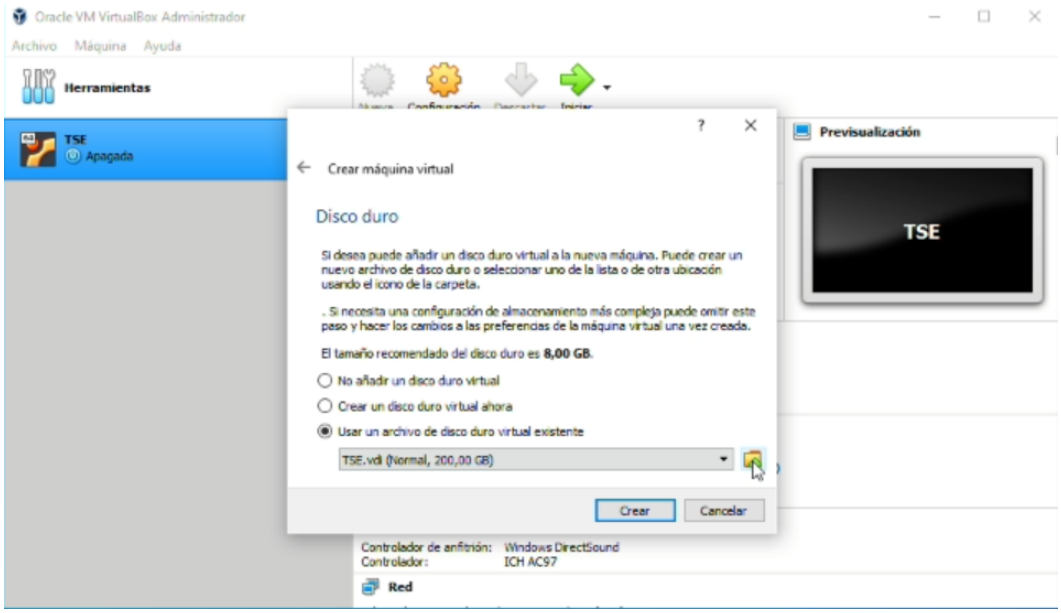


Figura 6: Seleccion de Disco Duro Virtual

Y se selecciona la imagen de disco duro virtual que creamos anteriormente:

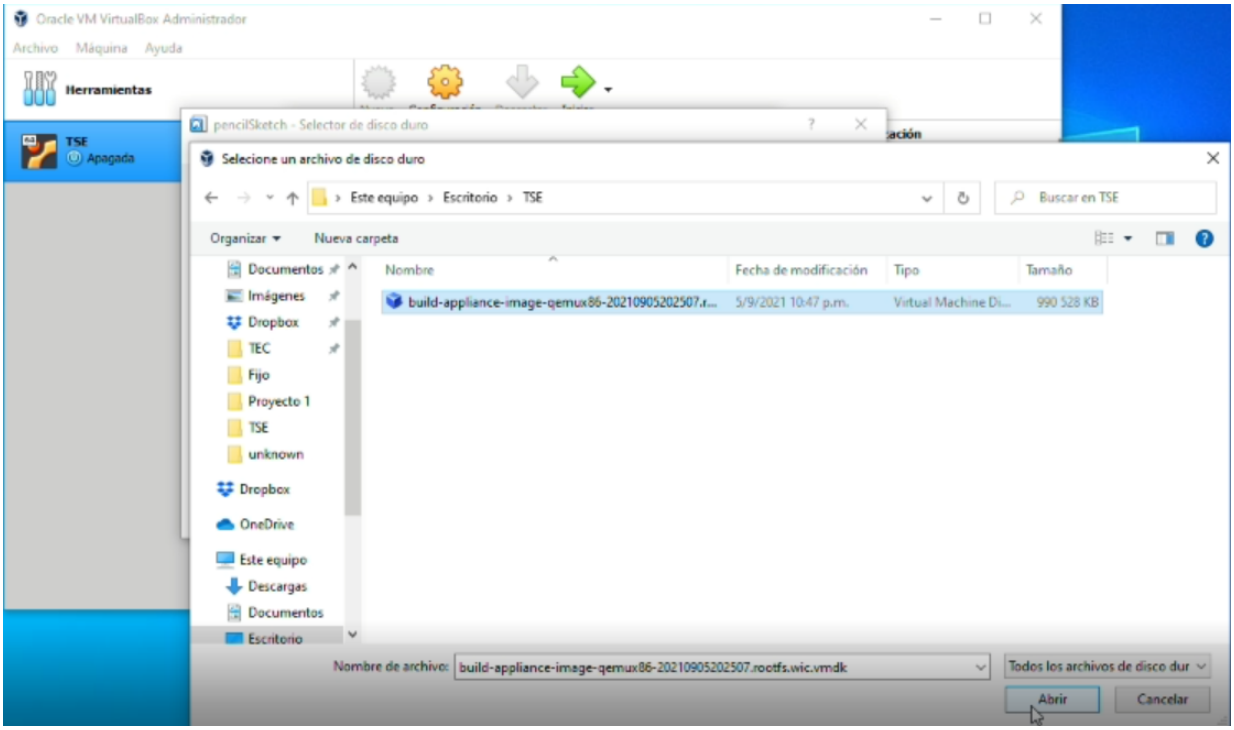


Figura 7: Selección de nuestra imagen de disco duro virtual

Por último ya creada la máquina virtual debemos ir a la configuración y asignarle suficientes cores del procesador, al igual que con la ram es preferible no pasar del indicador verde:

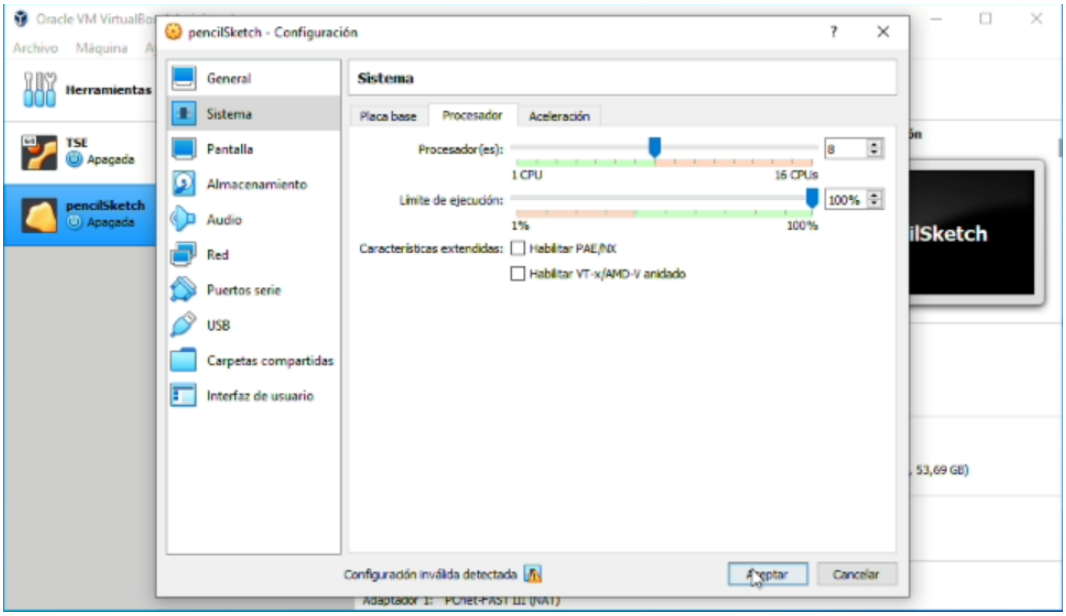


Figura 8: Asignación de Cores del procesador

Con esto ya tenemos la máquina virtual lista para iniciar y utilizar nuestra aplicación.

Referencias

[1] Colaboradores de Wikipedia. (2021). *Yocto Project.* Recuperado de: https://es.wikipedia.org/wiki/Yocto_Project

[2] Muthuganesan, T. (2018). *pencilSketch.* Recuperado de: https://github.com/TMuthuganesan/pencilSketch

[3] Linux Foundation. (2020). *Yocto Project Quick Build*. Recuperado de: https://www.yoctoproject.org/docs/current/brief-yoctoprojectqs/brief-yoctoprojectqs.html

[4] Ramswarup, K. (2021). *OpenCV - Overview.* Recuperado de: https://www.geeksforgeeks.org/opencv-overview/

[5] Colaboradores de Wikipedia. (2021). *OpenCV.* Recuperado de: https://es.wikipedia.org/wiki/OpenCV