Tarea 4
Sistemas operativos
Brayan Poloche- Karen Garcia

- **Artículo Linux Interrupts: The Basic Concepts.**

1. **Resumen.**

   Este documento presenta los conceptos y mecanismos básicos relacionados con el manejo de interrupciones en el kernel de Linux (versión 2.4.18-10). Se describe la diferencia entre interrupciones y excepciones, su procesamiento tanto en hardware como en software, y se detallan estructuras clave como la tabla de descriptores de interrupciones (IDT), softirqs, tasklets y bottom halves. Además, aborda el modelo de interrupciones en sistemas SMP (multiprocesadores) y las diferencias entre interrupciones de hardware, software y excepciones.

2. **¿Qué es una interrupción?.**

   Una interrupción es una señal que detiene temporalmente la ejecución actual de un programa en la CPU para que el procesador atienda una solicitud urgente, la cual puede ser generada por un dispositivo externo, un error del sistema o una instrucción del software.

3. **¿Por qué son necesarias las interrupciones?**

   Porque permiten al sistema operativo reaccionar de forma eficiente a eventos externos sin depender de técnicas eficientes como el "polling".

4. **¿Qué tipos de interrupciones existen y cuáles son sus características?.**

   - **Interrupciones de hardware.**

     - **Características:**

       - Son generadas por dispositivos a través de IRQs.

       - Pueden ser enmascarables o no enmascarables.

        - Estas interrupciones son gestionadas por el controlador de interrupciones PIC, el cual puede manejar hasta 8 interrupciones hardware.

        - Las PIC se pueden conectar en cascada.

- **Interrupciones generadas por software**

    - **Características:**

        - Se activan mediante la instrucción INT n, donde *n* especifica el número de la interrupción.

        - No pueden ser enmascaradas por el flag IF (Interrupt Flag) de la CPU, a diferencia de las interrupciones de hardware.

        - Son sincrónicas, ya que ocurren al ejecutarse una instrucción específica en el programa.

- **Excepciones**

    - **Características:**

        - Son generadas por el procesador cuando ocurre un error o evento especial durante la ejecución de una instrucción.

        - Las síncronas ocurren justo después de ejecutar una instrucción.

        - Tiene una clasificación interna que contiene: Faults las cuales permiten reintentar (ej. page fault). Traps son usados para depuración (ej. breakpoint). Aborts son errores, no recuperables (ej. corrupción de tablas).

5. **¿Qué son los "excepciton handler"?.**

    Son controladores que gestionan excepciones generadas por el procesador durante la ejecución de instrucciones. Guardan registros, analizan la situación y pueden terminar el proceso o aplicar mecanismos de recuperación.

6. Interrupciones generadas por software

- **¿Qué son?**

  Las interrupciones generadas por software son provocadas por instrucciones en el código del programa, y no por dispositivos de hardware ni por errores del procesador.

- **¿Para qué sirven?**

  Sirven principalmente para:

  - Invocar llamadas al sistema desde el modo usuario al modo kernel.
  - Ejecutar rutinas del sistema operativo de forma controlada y sincronizada.
  - Generar interrupciones programadas para depuración o pruebas (como int3 para breakpoints).

- **¿Qué algoritmos/diagramas de flujo son usados para el manejo de interrupciones?**

  Detección del evento por parte del procesador (INT o IRQ).

  - Identificación del vector asociado (usando la tabla IDT).
  - Verificación de privilegios (DPL vs. CPL).
  - Cambio de contexto:
    - Guardar registros: EIP, CS, EFLAGS, y opcionalmente SS, ESP.
    - Usar nueva pila si cambia de modo usuario a kernel.
  - Ejecución del handler (interrupción o excepción).

- Retorno al contexto original usando la instrucción iret:

    - Restaura registros y pila.

    - Si hay señales pendientes o necesidad de planificar (schedule()), se hace antes de volver al proceso.

7. **¿Qué son las IRQ y las estructuras de datos en relación con las interrupciones?.**

    - **¿Qué es?**

      Las IRQ son líneas de señalización mediante las cuales los dispositivos de hardware solicitan atención del procesador. Pueden ser compartidas entre varios dispositivos si el controlador y el software lo permiten.

    - **Estructuras :**

        - irq_desc_t: Estructura que describe una IRQ, contiene su estado, controlador, lista de acciones, etc.

        - hw_interrupt_type: Describe funciones específicas del controlador de interrupciones.

        - irqaction: Representa manejadores asociados a una IRQ, puede haber varios por línea si es compartida.

- **Articulo BOOTKITS: PAST, PRESENT & FUTURE**

1. **Resumen.**

   Este artículo analiza la evolución de los bootkits, una clase de malware que compromete el proceso de arranque para ejecutar código malicioso antes de que se inicie el sistema operativo. Se examinan tanto pruebas de concepto como amenazas reales (por ejemplo, Mebroot, TDL4, Rovnix, Gapz), con un enfoque en las técnicas de infección del MBR/VBR y en las amenazas modernas dirigidas al firmware UEFI.

2. **Principales puntos.**

- Orígenes históricos: Desde virus del sector de arranque (como Brain) hasta Apple II (Elk Cloner).

- Evolución moderna: Bootkits como Mebroot y Stoned resurgieron con técnicas más sofisticadas tras políticas de firma digital en Windows 64-bit.

- TDL4 y Rovnix: TDL4 sobreescribe el MBR; Rovnix modifica el VBR y emplea técnicas como manipulación de la IDT y hooking del int1h.

- Gapz: Infecta de forma muy sigilosa el VBR modificando solo unos pocos bytes y emplea almacenamiento oculto cifrado con AES y una pila TCP/IP propia en modo kernel para evadir firewalls.

- UEFI y Dreamboot: Explora la transición hacia plataformas con UEFI. Dreamboot reemplaza el bootloader original, manipula la carga del kernel y desactiva mecanismos como PatchGuard.

- Futuro: A pesar de tecnologías como Secure Boot, se anticipan ataques más dirigidos contra firmware UEFI vulnerable, especialmente por la falta de actualizaciones regulares y herramientas forenses especializadas.

- Herramientas:

    - CHIPSEC: Marco de Intel para analizar seguridad de firmware.

    - Hidden File System Reader: Para recuperar información en áreas ocultas del disco.

- **Brazo robótico**

1. Se instala el paquete python3-venv, el cual permite crear entornos virtuales en Python 3 para mantener los proyectos aislados.

```
brayan@brayan-laptop:~$ sudo apt install python3-venv
[sudo] contraseña para brayan:
Lo siento, pruebe otra vez.
[sudo] contraseña para brayan:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
python3-venv ya está en su versión más reciente (3.12.3-0ubuntu2).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 103 no actualizados.
brayan@brayan-laptop:~$ s
```

- sudo → Pide permisos de administrador.

- apt install python3-venv → Instala el módulo que permite crear entornos virtuales en Python 3.

2. Se crea un entorno virtual en una carpeta llamada venv/, que contiene su propia instalación de Python y sus propias librerías independientes del sistema.

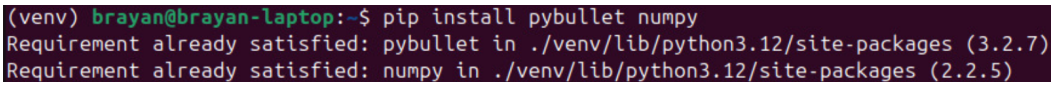
- python3 → Llama al intérprete de Python 3.
- -m venv → Usa el módulo venv para crear un entorno virtual.
- venv → Es el nombre de la carpeta donde se va a crear el entorno virtual.

3. Se activa el entorno virtual de Python 3 para trabajar dentro de un ambiente aislado.

- source → Ejecuta un archivo de configuración en la terminal.

- venv/bin/activate → Es el script que activa el entorno virtual.

4. Se instalan los paquetes necesarios dentro del entorno virtual: pybullet, para la simulación física del brazo robótico, y numpy, para realizar cálculos numéricos de manera eficiente.



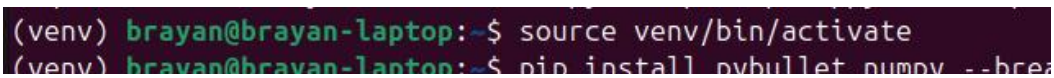
- pip → El programa para instalar paquetes de Python.

- install pybullet numpy → Instala dos paquetes:

    - pybullet → para simulaciones físicas

    - numpy→ para cálculo numérico

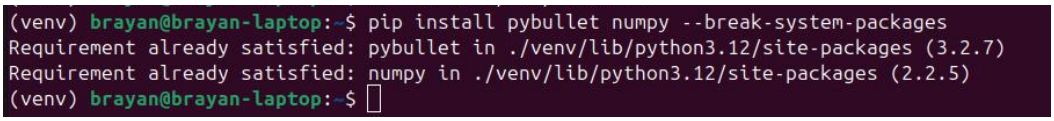
5. Se garantiza que el entorno virtual está activado, asegurando que todas las operaciones y ejecuciones utilicen las librerías locales del proyecto.



- source → Ejecuta un script en el contexto del terminal actual.

- venv/bin/activate → Es el script que activa el entorno virtual que se creó en la carpeta venv.

6. Se permite la instalación de paquetes que podrían romper las restricciones de librerías del sistema (--break-system-packages), asegurando compatibilidad en el entorno virtual.

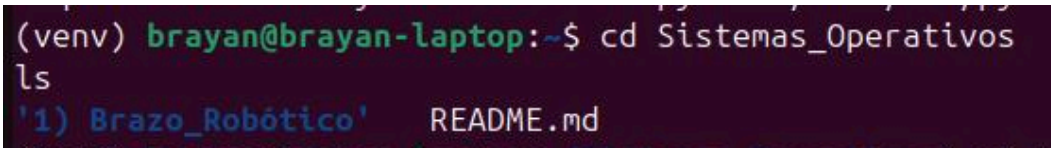


- pip install → Indica que se quiere instalar paquetes de Python.

- pybullet numpy → Son los nombres de los paquetes que se estan instalando (pybullet y numpy

- --break-system-packages → Permite que pip ignore restricciones del sistema en algunos entornos protegidos

7. Se ingresa al directorio Sistemas_Operativos y se listan sus contenidos para ubicar el proyecto.



- cd Sistemas_Operativos → Cambia al directorio llamado Sistemas_Operativos.

- ls → Listas del contenido de esa carpeta (muestra archivos y subcarpetas).

8. Se selecciona la carpeta 1) Brazo_Robótico, donde se encuentra el código del proyecto.

- cd → Entra a una carpeta

9. Se ingresa al interior de la carpeta 1) Brazo_Robótico y se listan los archivos disponibles.

```
(venv) brayan@brayan-laptop:~/Sistemas_Operativos/1) Brazo_Robótico$ ls
Dockerfile  main.py  two_joint_robot_custom.urdf
```

- ls → Listas del contenido de esa carpeta (muestra archivos y subcarpetas).

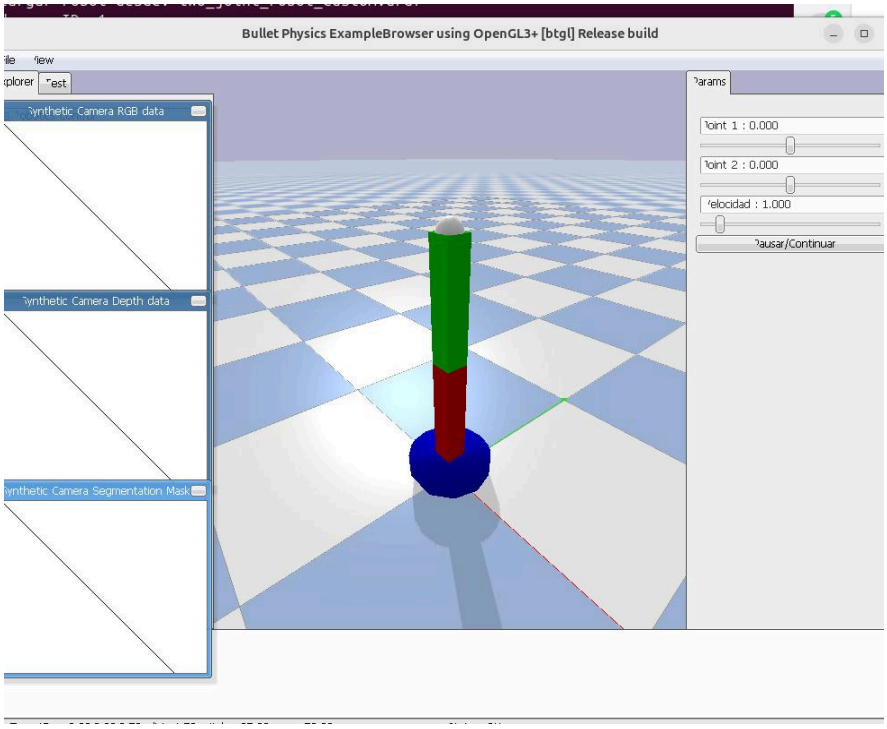
10. Se identifica y ejecuta el archivo main.py, que contiene la simulación del brazo robótico.

```
(venv) brayan@brayan-laptop:~/Sistemas_Operativos/1) Brazo_Robótico$ python main.py
pybullet build time: Apr 24 2025 13:19:33
startThreads creating 1 threads.
starting thread 0
started thread 0
argc=2
argv[0] = --unused
argv[1] = --start_demo_name=Physics Server
ExampleBrowserThreadFunc started
X11 functions dynamically loaded using dlopen/dlsym OK!
X11 functions dynamically loaded using dlopen/dlsym OK!
Creating context
Created GL 3.3 context
Direct GLX rendering context obtained
Making context current
GL_VENDOR=AMD
GL_RENDERER=AMD Radeon Graphics (radeonsi, renoir, LLVM 19.1.1, DRM 3.59, 6.11.0-21-generi
)
GL_VERSION=4.6 (Core Profile) Mesa 24.2.8-1ubuntu1~24.04.1
GL_SHADING_LANGUAGE_VERSION=4.60
pthread_getconcurrency()=0
Version = 4.6 (Core Profile) Mesa 24.2.8-1ubuntu1~24.04.1
Vendor = AMD
Renderer = AMD Radeon Graphics (radeonsi, renoir, LLVM 19.1.1, DRM 3.59, 6.11.0-21-generic
b3Printf: Selected demo: Physics Server
startThreads creating 1 threads.
starting thread 0
started thread 0
MotionThreadFunc thread started
ven = AMD
ven = AMD
Intentando cargar robot desde: two_joint_robot_custom.urdf
Robot cargado con ID: 1
Simulación iniciada. Use los deslizadores para controlar el robot.
Para mantener la ventana abierta, NO cierre este terminal.
X connection to :0 broken (explicit kill or server shutdown).
(venv) brayan@brayan-laptop:~/Sistemas_Operativos/1) Brazo_Robótico$
```

- python → Es el programa que interpreta y corre archivos .py (archivos de Python).

- main.py → Es el archivo que se quiere ejecutar dentro de esa carpeta.

11. Se visualiza la simulación del brazo robótico en la consola, utilizando la librería PyBullet.



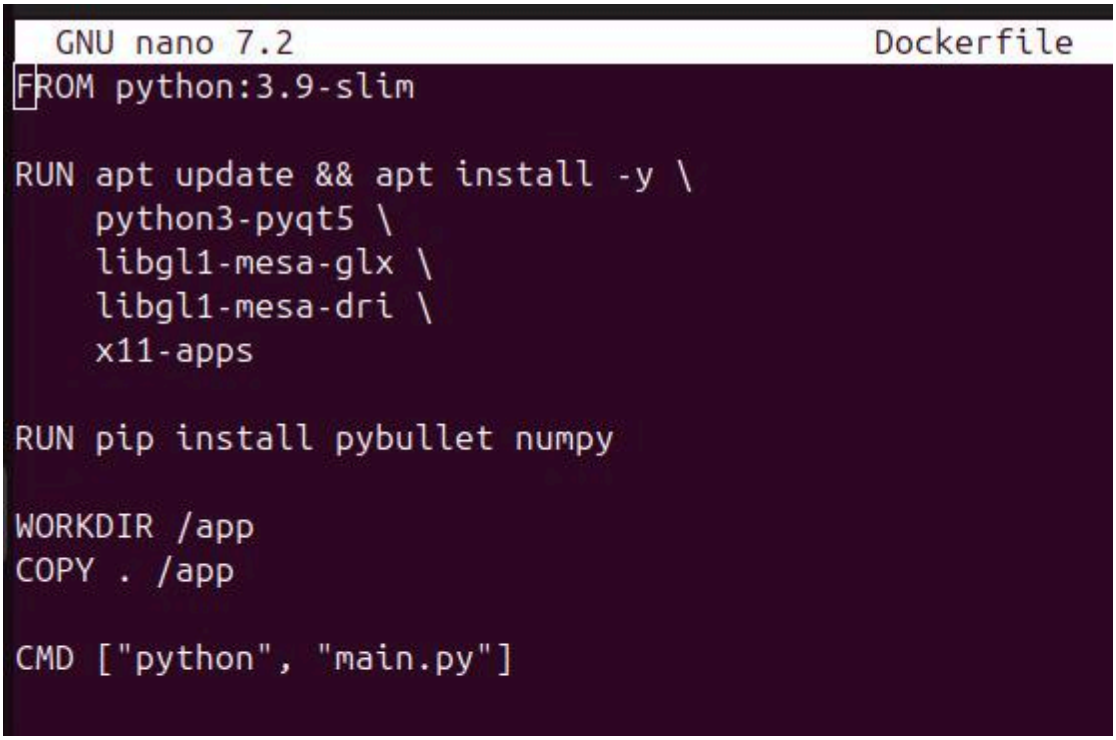
12. Se creó un archivo llamado Dockerfile con el editor nano para definir instrucciones de construcción de una imagen Docker.

```
(venv) brayan@brayan-laptop:~/Sistemas_Operativos/1) Brazo_Robótico$ nano Dockerfile
```

- nano → Es un editor de texto en la terminal..

- Dockerfile → Contiene instrucciones para construir una imagen de Docker.

13. Dentro del Dockerfile, se escribieron las instrucciones necesarias para preparar el entorno de ejecución.



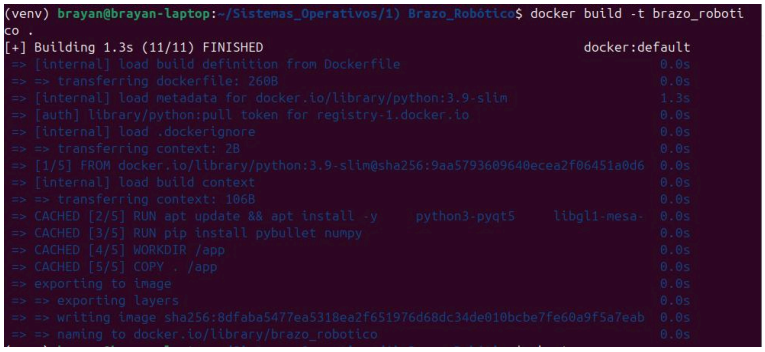
- FROM→ Sirve para indicar desde qué imagen base empieza el contenedor.
- python:3.9-slim → Imagen de Python versión 3.9, versión ligera (ocupa poco espacio).
- RUN → Ejecuta comandos dentro de la imagen para instalar programas o configurar.
- apt update → Actualiza la lista de paquetes del sistema operativo.
- apt install -y → Instala paquetes automáticamente (sin pedir confirmación).
- python3-pyqt5 → Se instala para crear interfaces gráficas (ventanas, botones).
- libgl1-mesa-glx → Se instala para manejar gráficos 3D (OpenGL).
- libgl1-mesa-dri → Se instala para acelerar gráficos (drivers de renderizado)..
- x11-apps → Se instalan herramientas gráficas básicas (para mostrar ventanas).

- RUN pip install pybullet numpy → Instala con `pip` las librerías de Python:

    - pybullet → Para simulación física.
    - numpy → Para operaciones matemáticas y matrices.

- WORKDIR /app → Cambia al directorio de trabajo /app dentro del contenedor.
- COPY ./app→ Copia el contenido de la carpeta local en el /app del contenedor.
- CMD ["python", "main.py"]→ Ejecuta el archivo main.py usando Python cuando arranca el contenedor.

14. Se construye la imagen llamada brazo_robotico a partir del Dockerfile.



- docker build → Le dice a Docker que construya (cree) una imagen nueva a partir de un Dockerfile
- -t brazo_robotico → Asigna el nombre (tag) brazo_robotico a la nueva imagen.
- . → Indica que el contexto de construcción (los archivos necesarios) están en el directorio actual.

15. Para permitir que la simulación gráfica se despliegue correctamente en pantalla, se prepara el entorno gráfico X11 y luego se corre el contenedor.

Primero, se habilita la conexión al servidor gráfico.

- xhost +→ Permite que cualquier cliente pueda conectarse al servidor gráfico (X11). Se necesita para mostrar interfaces gráficas desde Docker.

Después, se ejecuta la imagen.

- docker run -it --rm \ → Ejecuta un contenedor:

    - `-i` → modo interactivo.

- `-t` → asigna terminal.
- `--rm` → borra el contenedor al salir.

-

- -e DISPLAY=$DISPLAY \→ Le pasa la variable DISPLAY al contenedor para que pueda abrir ventanas gráficas.
- -v /tmp/.X11-unix:/tmp/.X11-unix \ → Comparte el socket de X11 entre el sistema y el contenedor.
- (Permite que el contenedor use la pantalla.)
- brazo_robotico → Nombre de la imagen Docker que se quiere correr.

16. Una vez ejecutado el contenedor, se abre una ventana en la que se visualiza la simulación del brazo robótico controlado mediante la librería PyBullet.