GDB para el kernel

En esta cadena les paso los pasos necesarios para poder utilizar GDB con el kernel que están desarrollando.

QEMU

Es necesario indicarle a qemu que en cuanto arranque congele el CPU para que puedan avanzar ustedes manualmente. Opción -S

También es necesario indicarle a qemu que espere una conexión de gdb. Opción -s

GDB

GDB necesita la tabla de símbolos, de lo contrario solo puede mostrar el assembler. Sin embargo, por más que compilen todo el TP con la opción -g, toda la información de debugging se pierde ya que el formato de salida que le piden al linker (en los scripts .ld) es binary. Para esto pueden linkear 2 veces, una con output binary y otra con output elf64-x86-64 que preserva la información de debugging. Otra opción es linkear con todos los símbolos y luego extraer el binario con objdump. Una vez que tienen los elf64-x86-64 de kernel y userland, le indican a GDB que cargue los símbolos desde estos archivos con la opción add-symbol-file. Esta opción necesita la dirección de memoria donde será cargado en memoria cada módulo.

GDB necesita la dirección IP y el puerto donde está escuchando qemu (opción -s de qemu). Por defecto el puerto es 1234 (en realidad la poción -s de qemu es lo mismo que -gdb tcp::1234), luego le indican a GDB estos datos con la opción target remote IP:PORT. La dirección IP debe ser la dirección local (ifconfig).

Con esto es suficiente , sin embargo les voy a pasar algunos pasos/datos extra.

Sería interesante que el run.sh tome como parámetro si tiene que ejecutar qemu para GDB o no así no lo están modificando a cada rato:

```
#!/bin/bash
if [[ "$1" = "gdb" ]]; then
  qemu-system-x86_64 -s -S -hda Image/x64BareBonesImage.qcow2 -m 512
else
  qemu-system-x86_64 -hda Image/x64BareBonesImage.qcow2 -m 512
fi
```

GDB soporta interfaces gráficas en python, lo que permite entre otras cosas poder ver diferentes layouts de GDB simultáneamente. En los archivos del TP2 adjunté un archivo llamado .gdbinit con una interfaz. Este archivo debe estar en la ruta en la que van a ejecutar GDB o en el home. A este archivo le pueden agregar los comandos para GDB que vimos anteriormente, por ejemplo, el head de .gdbinit quedaría así: (ojo con las mayúsculas y minúsculas de kernel y Userland, deben coincidir con las de su TP).

```
target remote 192.168.0.11:1234
add-symbol-file kernel/kernel.elf 0x100000
add-symbol-file Userland/0000-sampleCodeModule.elf 0x400000
python

# GDB dashboard - Modular visual interface for GDB in Python.
…
```

Para el linkeo en un formato diferente que el explicitado en los scripts .ld, es posible ignorar la opción del script agregando la opción al comando ld. Si usan gcc también es posible pasar flags directo al linker, por ejemplo:

Si linkean de esta manera

```
$(LD) $(LDFLAGS) -T kernel.ld -o $(KERNEL) $(LOADEROBJECT) $(OBJECTS) $(OBJECTS_ASM) $(STATICLIBS)
```

Pueden cambiar el formato con

```
$(LD) $(LDFLAGS) -T kernel.ld --oformat=elf64-x86-64 -o kernel.elf $(LOADEROBJECT) $(OBJECTS) $(OBJECTS_ASM) $(STATICLIBS)
```

Si compilan y linkean en un solo paso de esta manera

```
$(GCC) $(GCCFLAGS) -I./include -T sampleCodeModule.ld _loader.c $(SOURCES) $(OBJECTS_ASM) -o ../$(MODULE)
```

Pueden cambiar el formato con la opción -Wl que envia los flags especificados directamente al linker.

```
$(GCC) $(GCCFLAGS) -I./include -T sampleCodeModule.ld -Wl,--oformat=elf64-x86-64 _loader.c $(SOURCES) $(OBJECTS_ASM) -o ../0000-sampleCodeModule.elf
```

Recuerden que no deben reemplazar los comandos de su make por estos, sino agregar los presentados aquí, ya que el comando original genera el binario que se va a utilizar como siempre y el comando nuevo genera un binario que preserva los símbolos y será usado por GDB unicamente.

Finalmente, para obtener la IP local de su pc pueden probar lo siguiente

```
ifconfig | grep "status: active" -B4 | grep "inet " | xargs | cut -d " " -f2
```

El Docker que tienen ya tiene gdb instalado, y pueden conectar gdb directamente con qemu corriendo en el host.

Qemu también permite imprimir todas las interrupciones que ocurren con la opción -d int. Si bien el output es ilegible dado que imprime muchísima información, se puede filtrar la salida con grep, además, frente a un problema podemos estar interesados en la última interrupción, excepción o fault, así podremos averiguar exactamente dónde estaba el RIP al momento del problema, por ejemplo una NMI o GP.