CUADERNO DE BITÁCORA.

SESION 1.

Dado que ya tenemos vb instalado, simplemente creamos la maquina de Minix.

La llamamos minix, la damos de tipo de SO otros (64b) 64 mb, usamos el archivo de disco duro que nos ofrece la practica y entramos en su configuración.

Creamos un nuevo controlador de disquete con el archivo proporcionado.

En sistema quitamos la opción óptica y ponemos como preferencia disco duro.

Intentamos realizar la apertura de la segunda maquina porque al copiar los archivos nos daba un error de UUID.

Buscamos una solución y encontramos dos maneras, una de las opciones era mas larga sin embargo podíamos cambiar el uuid como quisiéramos. La segunda opción, que es la que utilizamos era mediante una herramienta de vb que permite clonar la maquina virtual cambiando automáticamente su uuid.

Entramos con vi en la dirección etc/passwd y en la primera linea y ahora cambiamos lo pertinente en el fichero.

Mediante el comando mkdir creamos una carpeta llamada /root/trabajos que va a ser nuestro home.

En el mismo archivo de antes copiamos la linea de tanembaum mediante el comando “yy” y pulsando “p” la pegamos al final del archivo y lo modificamos para darle otro nombre de usuario y establecer la carpeta home así como la shell. También borramos el espacio de la contraseña para modificarlo mas tarde.

Hacemos reboot y entramos en el nuevo usuario, mediante el comando passwd añadimos una contraseña al usuario. Conseguimos entrar desde otro terminal.

Cambiamos el mensaje de inicio en la función tty\_task() con vi.

Después de compilar el nucleo, comprobamos y todos los comandos se han ejecutado y la revision ha cambiado correctamente.

Comprobamos que al cambiar el orden de arranque a disquete primero, el mensaje cambia al original.

Después de dudar un rato, escribimos el programa y funcionaba correctamente. Comprobamos que a partir de 26, daba fallo.

SESION 2.

Entramos en /usr/src/lib/posix y vemos con vi el archivo fork.c, comprobamos que devuelve lo que devuelva syscall(). fork pasa a syscall 3 argumentos, MM, FORK, &m siendo MM el servidor, FORK el tipo de llamada al sistema y &m la dirección de memoria.

A continuación accedemos al código de syscall en la carpeta other/ . llama a sendrec(). Compara el int que devuelve sendrec() con 0 para comprobar que no existan errores. Si hay errores, guarda lo que devuelve sendrec() en el registro que le hemos pasado a syscall().

En la segunda comprobación de error, mira que el registro que hemos pasado exista. Si no existe, syscall devolver,a (-1) y por lo tanto, fork también.

Si el programa funciona con normalidad, devuelve la dirección de la operación de llamada al sistema (FORK en este caso).

taskcall() realiza la misma función que syscall()

Miramos i386/\_sendrec.s que después de guardar lo que recibe como parámetro realiza una interrupción. Suponemos que el argumento 33 hace que la interrupción nos haga pasar a modo privilegiado.

Por lo tanto hasta este punto se ha hecho todo en modo usuario.

En protect.c se llama en la linea 206 a int\_gate con 3 argumentos, SYS386VECTOR que contiene el 33 que antes vimos.Lo que devuelva s\_call convertido en bytes virtuales y después en físicos seguido de una operación que no entendemos.

Vamos a s\_call() y comprobamos lo descrito en la guía de la sesión.

Llegamos a sys\_call y al realizar el trap al kernel continuamos en modo privilegiado.

Volvemos a mpx386.s cuando sys\_call() termina, s\_call deshabilita las interrupciones y llama a \_restart(). Seguidamente, vuelve a cambiar a modo usuario

Para el apartado dos, miramos als funciones del gestor de memoria y decidimos que forkexit.c era la que seguramente realizada la función que necesitábamos (saludar)

Una vez editando su texto, añadimos las lineas necesarias después de la declaración de las variables de do\_fork() y ahora al llamar a fork , hemos comprobado que saluda.

SESION 3.

Entramos en callnr.h, añadimos la función 77, ESOPS y cambiamos el total de llamadas al sistema a 78.

En table.c, añadimos la función do\_esops, al final del archivo, la que seria la posición 77.

Y en proto.h, apartado de “signal.c” escribimos la función do\_esosps dentro de \_PROTOTYPE

Escribimos al final de utility.c la función do\_esops con lo especificado en el fichero de la sesión.

Añadimos la linea de do\_esops en FORWARD \_PROTOTYPE , metemos en el switch de systask()

la función do\_esops() y definimos su código justo antes de la función do\_fork().

Escribimos la función do\_esops, en la que accedemos a los campos de la estructura del mensaje, los imprime, los modifica, y los vuelve a imprimir.

Al intentar compilar el núcleo nos aparece un error en system.c

(Lineas 202,203,204)

Una vez solucionado, creamos el subdirectorio practicaC y el fichero practicaC.c