Introduccion a los Sistemas Operativos

Teoria 1 – 20/8/24:

Sistema Operativo:

* Es un software que esta pensado para administrar otro software.
* Requiere un procesador y memoria
* El SO oculta el HW y presenta a los programas abstracciones mas simples de manejar.
* Usuario 🡪 SO 🡨 Hardware
* Administra los recursos de HW de uno o mas procesos.
* Ejecución simultanea de procesos.

Interrupciones:

* Por Hardware: Son asincrónicas
* Por Software: Se causan por algo que se esta haciendo mal. Por ejemlo: Acceso a memoria indebido, división por 0, buffer overflow. Se causa por el ciclo de execute.

Teoría - 27/8/24:

System Calls:

* Es la forma en que los programas de usuario acceden a los servicios del SO
* Se ejecutan en modo kernel o supervisor
* Los parámetros asociados a las llamadas pueden pasarse de carias maneras: por registros, bloques o tablas en memoria o la pila.
* Ej: count=read(file, buffer, nbytes);

Los programas no funcionan en los dos con el mismo código ya que las System Calls en cada sistema operativo es diferente.

Los compiladores no son los mismos entre sistemas operativos. Entre 32 y 64 bits es diferente, RM e INTEL también son diferentes, etc.

* El codigo: read(file, buffer, nbytes); se traduce en:
  + Push file
  + Push buffer
  + Push nbytes
  + Call

Se usa como una “api” para llegar del mundo usuario al mundo kernel, que hace:

Se busca forzar una interrupción

Para buscar que llamada al sistema busco de todas las que hay

En la llamada a sistema hay un vector de direcciones de funciones

Una vez que estoy en el vector de llamadas al sistema, el sys call handler, es el que sabe a donde avanzar y ejecutando la llamada a sistema.

SYSCALL:

Para programar el clásico “hello world” se necesitan mínimo 2 llamadas al sistema:

* SYSCALL WRITE
* SYSCALL EXIT

El comando man es para ayuda, “man write” en consola te dice que hace el write.

En arquitectura de 32 bits, Linux tiene 3 archivos standard:

* Entrada standard pos 0
* Salida standard pos 1
* Error standard pos 2

En arquitectura de 64 bits

Proceso:

* Es un programa en ejecución
* Para nosotros serán sinónimos: tarea, job y proceso.

Diferencia entre programa y proceso:

Programa:

* Es estático
* No tiene program counter
* Existe desde que se edita hasta que se borra

Proceso:

* Es dinámico
* Tiene program counter
* Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina.

El programa es el “.exe”, cuando vos ejecutas el programa, se crea un proceso que ejecuta el programa. Lo que se ejecuta es el proceso que ejecuta el programa.

El programa nunca cambia, lo que cambia es el proceso ya que cada vez que se abre se crea un proceso que ejecuta este programa, y cuando se cierra se termina la ejecución del proceso.

Los procesos pueden coexistir en un SO, pero si solo tenemos una CPU, se puede ejecutar 1 por vez

Atributos de un proceso:

* Identificación del proceso y del proceso padre.
* Identificación del susuario que lo “disparo”
* Si hay estructura de gruspo, grupo que lo disparo.
* En ambientes multiusuario, desde que terminal y quien lo ejecuta.

Process control block (PCB):

* Estructura de datos asociada al proceso (abstracción)
* Existe 1 por proceso
* Es lo primero que se crea cuando se crea un proceso y lo ultimo que se borra cuando termina.
* Contiene la información asociada con cada proceso:
  + PID, PPID, etc.
  + Valores de los registros de la CPU (PC, AC, etc.)
  + Planificación (estado, prioridad, tiempo consumido, etc.)
  + Ubicación (representación) en memoria.
  + Accouting
  + Entrada salida (estado, pendientes, etc.)

Practica – 29/8/24:

En /dev hay archivos que identifican dispositivos.

En /etc hay archivos de configuración.

UEFI:

hace como un escribano

cuando el so bootea, va a la parte del uefi y checkea si el software esta certificado y quien lo certifico. Lo mismo ocurre con el navegador.

Practica 1 fin hasta UEFI.

Practica 2 empieza:

Las particiones se identifican con letras desde la C en Windows.

Las particiones en Linux están basadas en los buses:

* /dev/hda: configurado como Master en el 1º bus IDE
* /dev/hdb: configurado como Slave en el 1º bus IDE
* /dev/hdc: configurado como Master en el 2º bus IDE
* /dev/hdd: configurado como Slave en el 2º bus IDE

Después surgieron otros nuevos:

* /dev/sda
* /dev/sdb
* /dev/sdc
* /dev/sdd

Si se cambian los discos de lugar en los puertos de la mother, no va a bootear ya que se tiene guardado donde esta el /boot y no lo encontrara.

Usar VIM

Los archivos mas importantes de la parte de USUARIO son:

* /etc/passwd
* /etc/group
* /etc/shadow

Comandos para manejo de usuarios:

* Useradd
* Passwd
* Usermod
* Userdel
* Groupdel

Permisos:

RWX

R: READ

W: WRITE

X: EXECUTION

Sobre los usuarios se aplican:

UGO:

1. Permisos del dueño
2. Permisos del grupo
3. Permisos de otros usuarios

Para cambiar los permisos se usa CHMOD:

1 1 1  
R W X  
  
Esto en binario es 7, para que tenga todos los permisos se debe poner 7.

1 0 1  
R W X

Esto en binario es 5, que te da permisos de lectura y ejecución, el mas normal.

Para tener todos los permisos de dueño y el grpo y otros usuarios tener solo lectura y ejecución, se debe poner:

CHMOD 755 /directorio/

7 5 5  
U G O

El proceso de arranque:

1. Se empieza a ejecutar el codigo del BIOS
2. El BIOS ejecuta el POST
3. El BIOS lee el sector de arranque (MBR)
4. Se carga el gestor de arranque (MBC)
5. El bootloader carga el kernel y el initrd
6. Se monta el initrd como sistema de archivos raız y se inicializan componentes esenciales (ej.: scheduler)
7. El Kernel ejecuta el proceso init y se desmonta el initrd
8. Se lee el /etc/inittab
9. Se ejecutan los scripts apuntados por el runlevel 1
10. El final del runlevel 1 le indica que vaya al runlevel por defecto
11. Se ejecutan los scripts apuntados por el runlevel por defecto
12. El sistema esta listo para usarse

Comandos:

El pipe “|” concatena comandos

Wc: cuenta líneas

“algo” > miarchivo modifica el archivo entero.

“algo mas” >> miarchivo agrega al final del archivo.

Teoría – 30/8/24:

Que es el espacio de direcciones en un proceso?

* Es el conjunto de direcciones de memoria que ocupa el proceso:
  + Stack
  + Text
  + Datos
* No incluye su PCB o tablas asociadas.
* faltan

El bus de direcciones le da el espacio de direcciones del proceso

El contexto de un proceso:

* Incluye toda la información que SO necesita para adminstrar el proceso, y la CPU para ejecutarlo correctamente.
* Son parte del contetto, los registros de cpu, inclusive el contador de program, prioridad del proceso si tiene E/S pendientes, etc.

Cambio de contexto (Content Switch):

* Se produce cuando la CPU cambia de un proceso a otro.
* Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornara después a la CPU.
* Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto.
* Es tiempo no productivo de CPU.
* El tiempo que consume depende del soporte de HW.

Cada interrupción (que es la q despierta al kernel) va a involucrar un Content Switch

Este modelo plantea meter el kernel dentro de los procesos.

El kernel “dentro” del proceso:

* El “código” del kernel se encuentra dentro del espacio de direcciones de cada proceso
* El kernel se ejecuta en el mismo contexto que algún proceso de usuario
* El kernel se puede ver como una colección de rutinas que el proceso utiliza

Las PCB son estructuras de datos que tienen que estar en la ram.

El stack/pila tiene:

Stack modo kernel

* Si el proceso pisa alguna dirección de retorno del kernel, cuando vuelva al modo usuario se arma quilombo, por eso se tiene que administrar bien los stack pointer.

Stack modo usuario

Teoría – 3/9/24:

Estados de un proceso:

Siempre tiene que haber un proceso corriendo para que se mantenga vivo, si no hay un proceso el sistema tiene que crear uno

P1 se viene ejecutando, sucede una interrupción y P1 no para su ejecución, ya que esta corriendo en modo usuario. Por la interrupción pasa a modo kernel. Tiene tiempo? (preguntar esto)

P2 venia en estado READY

Sucede el cambio de contexto y empieza a ejecutarse P2 en modo kernel porque veníamos en modo kernel y carga el contexto.

Cuando termina de cargar el contexto cambia de modo kernel a modo usuario y corre normalmente.

Módulos de planificación:

* Scheduler de long term: se encarga de las transiciones de nuevo a ready to run. Determina de todos los nuevos, cual de los procesos pasa a ready to run.
* Scheduler de medium term: hace años, determinaba que había que bajar la cantidad de procesos listos en memoria para ejecutar ya que se requeria liberar memoria principal.
* Scheduler de short term: se encarga de la transición de ready to run a running. Es el planificador.

Su nombre proviene de la frecuencia de ejecución.

Practica - 6/9/24:

Shell Scripting:

* Practico para manejar archivos
* Simple para crear procesos y manipular salidas
* Funciona en cualquier sistema operativo
* Independiente de la plataforma

Instrucciones: comandos

* Internos o built in
* Externos

Redirecciones y pipes

Comentarios que empiezan con #

Estructuras de control:

* If
* While
* For
* Case

Variables

* Strings: si son números solos también se consideran strings
* Arreglos ()

Funciones

Chmod:

chmod +x (archivo) a todos

chmod +w (archivo)

chmod +r (archivo)

chmod u+x (archivo) solo a usuario

chmod g+x (archivo)

chmod o+x (archivo)

Remove:

rm -rf directorio1 (r es recursivo y rf si lo encontras borra sino pasa de largo)

cat para mostrar

less se puede buscar dentro de un archivo y recorrerlo

more se puede ver el archivo entero

wc -l cant palabras

wc -c cant de caracteres

grep para buscar una palabra en un archivo

grep hola texto.txt (busca todos los “hola” dentro de texto.txt)

para buscar un archivo se usa find

find tmp \*

el asteriosco es un comodin, por ejemplo: \*.sh busca todo lo que termine en .sh, hola\* busca todo lo que empiece con hola

Teoría – 10/9/24:

Comportamientos de los procesos:

* CPU-Bound
  + Mayor parte del tiempo utilizando la CPU
* I/O Bound
  + Mayor parte del tiempo esperando por E/S

La velocidad de la cpu es mucho mas rápida que la de los dispositivos de E/S

Algoritmos apropiativos y no apropiativos:

* Apropiativos: existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU. (va de RUNNING a READY TO RUN. Es una decisión que toma el kernel)
* No Apropiativo:

Procesos batch:

* FCFS: First come first served
* SJF: Shortest job first

Se puede tener en cuenta el uso de FCFS o SJF en cual es mas rápido.

A:8 B:4 C:4 D:4

FCFS: 9 promedio. 8 + 4 + 4 + 4 = 0 + 8 + 12 + 16 = 36/4 = 9

SJF: 6 promedio. 4 + 4 + 4 + 8 = 0 + 4 + 8 + 12 = 24/4 = 6

Procesos iterativos:

* Round Robin
* Prioridad: se ordena por una prioridad que se asigna
* Colas multinivel
* SRTF: Shortest remaining time first.

Creacion de procesos:

* Todo proceso es creado por un proceso. (El Shell es el PROCESO que usa el usuario para crear procesos) (los procesos se crean con system calls)
* Un proceso padre tiene uno o mas procesos hijos
* Se forma un árbol de procesos.

Como se crea:

* En UNIX (2 System calls):
  + System call fork() crea nuevo proceso igual al llamador. (literalmente identico)
  + System call execve(), generalmente usada después del fork, carga un nuevo programa en el espacio de direcciones.
* En Windows (1 System call):
* System call CreateProcess(crea un nuevo proceso y carga el programa par ejecucion)

Fork() devuelve dos valores, un valor para el padre y un valor para el hijo.

Practica - 12/9/24:

* Wc cuenta palabras pero si le pones -l cuenta líneas.
* Grep filtra por palabras
* ls | grep variables
* cut -d: f1 archivo
  + cut es como el split. -d y luego el separador. F1 es la primera parte del cut
* cat \* mira el contenido de todos los archivos donde estoy parado
* Para buscar una palabra en un archivo se usa: “cat archivo | grep palabra”
  + Si quiero contar las líneas encontradas que tienen la palabra buscada se le agrega un “ | wc -l”
* Tar se usa para empaquetar y desempaquetar
  + Tar -cvf archivo1 archivo2 archivo3
  + Tar -xvf archivo.tar
    - Sirve para desempaquetar. (x)
  + Tar -cvzf archivo.tar.gz f1 f2 f3
    - Se comprime y se le pone “.gz” para saber que esta comprimido
* Gzip se usa para descomprimir
  + Gzip archivo.tar

Variables:

* Son case sensitive
* Se crea con ‘ NOMBRE=”pepe” ‘
* Se imprime con
  + Echo “hola $NOMBRE”
  + Echo ${NOMBRE}asdadsadasdas

Arreglos:

* Arreglo1=()
* Arreglo2=(1,2,3,4,5)
* Para imprimir: echo $arreglo1
  + imprime la primera posición del arreglo.
* Echo $(arreglo2[2])
  + imprime la posición 2 del arreglo.
* Echo $(arreglo2[@])
  + Imprime todo el contenido.
* Echo $(#arreglo2[@])
  + Imprime la cantidad de valores que hay

Case:

Case $variable in  
 “valor 1”:   
 block  
 ;;  
esac

If:

If [condicion]

Variable de entorno:

* Export A=”Algo”
* Con UNSET borro la variable
  + Unset A

Operadores con números:

* Igual = $edad -eq 20 (equals)
* Desigual = $edad -ne 20 (not equals)
* Mayor que = 5 -gt (greater than)
* Mayor o igual que = 5 -ge (greater equals)
* Menor que = 5 -lt (less than)
* Menor o igual que = 5 -le (less equals)

For:

Puede ser for each o for normal

For normal:

For (condicion como java)  
do  
 block  
done  
--------------  
For each:

For v in value1  
do  
 block  
done

Teoria – 17/9/24:

Administracion de memoria:

Todo proceso tiene un espacio de 0 a n donde n es la arquitectura.

La idea es programar de forma abstracta, si hacemos referencia a direcciones no nos preocupamos por si otro proceso las quiere usar.

El SO brinda la posibilidad de acceso compartido a memoria por ejemplo a dos procesos Word que físicamente seria erroneo que tengan 2 espacios en memoria.

El rango de direcciones lógicas que puede tener un sistema es depende de la arquitectura del procesador:

* 32 bits: 0 a 2^32 – 1
* 64 bits: 0 a 2^64 – 1

Direcciones Lógicas y Físicas.

El hardware que mapea direcciones virtuales a físicas:

* Es parte del procesador
* Programarla se debe hacer en kernel mode.

Técnicas

Partamos la memoria principal en particiones

Se puede particionar fijamente pero esto puede ceder a muchos problemas como por ejemplo que venga algo mas grande que mi partición.

La memoria RAM arranca pelada, aparece el primer proceso y genero una partición, lo meto ahi y así sucesivamente.

Si tengo que agregar un proceso se pueden aplicar diferentes condiciones, ingresarlo a la primera partición que se encuentre, la mas grande, la mas chica que entre, etc.

El .exe tiene un compilador que sabe a quien llamar y en que parte de la memoria meterse para ejecutar el archivo.

Practica - 19/9/24:

IMPORTANTE BAJA PUNTOS EN PARCIAL:

Validar los parámetros y terminar con exit <0 para una salida sin errores.

Teoría - 20/9/24:

Mecanismos de asignación de memoria:

* Particiones fijas
* Particiones dinámicas

Todo lo que se requiere como estructura se encuentra en la RAM.

Ejemplo: Mov 1234a

Primero tiene que acceder a memoria para buscar la tabla de segmentos y traducir la dirección ingresada “1234a”

Paginacion:

Se agarra el proceso y se divide en paginas (ejemplo del profe en el pizarrón: tajear el cuadradito del proceso)

Cualquier pagina del proceso entra en cualquiera de los marcos, ya que todos son del mismo tamaño

Adentro de la entrada de cada pagina esta el ancho y largo de esta.

La diferencia entre paginas y segmentación es que al ejecutar 2 veces el mismo proceso, se van a guardar en 2 sectores diferentes en memoria principal en vez de tenerlo solo en 1.

2 direcciones lógicas que al traducirlas terminan físicamente en el mismo lugar de memoria.

HASTA TRANSPARENCIA INTEL X386 ENTRA EL PRIMER PARCIAL DE PROMOCION.