Sistemas Operativos - Apuntes para final

Gianfranco Zamboni

27 de marzo de 2020

Índice

1. Introducción

Un sistema operativo es un intermedario entre los elementos de hardware y los programas de un usuario. Tiene que manejar la contención y la concurrencia tratando de lograr un buen rendimiento.

Sus elementos básicos son:

- **Drivers:** Programas manejan los detalles de bajo nivel relacionados con la operación de los distintos dispositivos.
- **Núcleo**: (o Kernel) Es el sistema operativo, propiamente dicho. Se encarga de las tareas fundamentales y contiene los diversos subsitemas que iremos viendo a lo largo de la materia.
- **Interprete de comandos:** (o Shell) Un programa que le permite al usuario interactuar con el sistema operativo.
- **Procesos:** Son todos los programas en ejecución junto con su espacio de memoria asociado y otros atributos.
- **Sistema de archivos**: Forma de organizar los datos en el disco para gestionar su acceso, permisos, etc.
  - **Archivo:** Secuencia de bits con un nombre y una serie de atributos que indican permisos.
    - **Binario del sistema**: Son archivos que no forman parte del kernel pero suelen llevar a cabo tareas muy importantes o prooveer las utilidades básicas del sistema.
    - **Archivos de configuración:** Son archivos especiales con información que el sistema operativo necesita para funcionar.
  - **Directorio**: Colección de archivos y directorios que contiene un nombre y se organiza jerárquicamente.
    - **Directorios del sistema**: Son directorios donde el propio SO guarda archivos que necesita para su funcionamiento.
  - **Dispositivo virtual**: Abstracción de un dispositivo físico bajo la forma, en general, de un archivo de manera tal que se pueda abrir, leer, escribir, etc.
  - **Usuario**: La representación, dentro del propio Sistema Operativo, de las personas qo entidades que pueden usarlo. Sirve principalmente como una forma de aislar infromación entre distintos usuarios reales y de establecer limitaciones.
  - **Grupo**: Una colección de usuarios

Manejo básico de un shell Unix

2. Procesos y API

En este modelo, todo software que puede ser ejecutado (incluso el mismo SO) es organizado en procesos. Los procesos son instancias de un programa en ejecución, incluyen program counter (PC), registros, variables, archivos abiertos por el proceso y cualquier dato necesario para que el SO pueda ejecutarlo.

A cada proceso se le asigna un **process Id** (o pid)

Cuando el CPU tiene un solo núcleo y se necesitan ejecutar varios procesos al mismo tiempo, se simula el multitasking haciendo correr de manera consecutiva cada proceso por un intervalo corto de tiempo. Esto es, dado un intervalo de tiempo (quantum), un proceso es ejecutado durante 1 quantum, luego se realiza un cambio de de contexto y se pasa al siguiente proceso.

2.1. Cambio de contexto (Cotext Swith)

1. Se guarda el program counter (PC), registros, stack pointer (SP), recursos en uso y estado del proceso [etc] actual en la tabla de procesos.
2. Se carga el program counter del vector de interrupciones que fue asignado al scheduler.
3. Se carga los datos del scheduler (los mismo que en el primer paso).
4. El scheduler decide a que proceso le toca y devuelve el process id de este.
5. Se cargan los datos del próximo proceso a ejecutar.

El Process Control Block (PCB) Para implementar el modelo de procesos, el sistema operativo mantiene una tabla de procesos (PT) con una entrada por proceso, cada una de estas entradas es un PCB y contiene información sobre el estado del proceso, el PC, el SP, el estado de los archivos que tiene abiertos, la información de scheduling y cualquier otra información que sea considerada necesaria por el SO para que, si el proceso es interrumpido, cuando vuelva a ser su turno pueda seguir corriendo como si nada hubiese pasado.

System calls (syscalls) La forma más frecuente, para un proceso, de comunicarse con el SO son las sycalls. Los syscalls son rutinas escritas en lenguajes de bajo nivel que permiten hacer uso de los servicios habilitados por el sistema. Por lo general, para hacer un syscall, se pasa el número correspondiente a la syscall que se desea hacer al registro indicado por el sistema operativo (en Linux el registro ‘rax‘). Después se ejecuta una interrupción del tipo **TRAP** que cambia de modo usuario a modo kernel y luego se ejecuta el handler correspondiente. Por último, se vuelve a modo usuario y, si es posible, se retorna el control al proceso con el PC aumentado en uno.

Las syscalls se diferencian de las llamadas a funciones de librería, en que ejecutan código provisto por el kernel en modo kernel cuando las otras se ejecutan con los privilegios del proceso que la llamó .

Las syscalls pueden ser agrupadas en seis grandes categorias: * **De control de procesos**

Crean, borran o modifican el estado y/o los atributos de un proceso. * **De administración de archivos**

Crean, borran, abren, cierran, leen o modifican archivos. * **De administración de dispositivos**

Permiten, a los procesos, usar los recursos del sistema (los recursos pueden ser pensados como dispositivos físicos o virtuales) * **De mantenimiento de información**

Permiten acceder a información del sistema (fecha, hora, nombre del SO, archivos y otros atributos) * **De Comunicación**

Permiten que dos o mas procesos y/o dispositivos intercambien información (pasándose mensajes o a través de memoria compartida).

Estados de un proceso New

Cuando el proceso es recién creado. En Linux, la única syscall que permite crear un nuevo proceso es 'fork'. Esta, crea un clon exacto del proceso que llamó a la syscall.

Terminated

Cuando el proceso terminó de ejecutarse. Esto se puede deber a varios motivos: * Porque terminó lo que debía hacer. * Porque hubo algún error durante la ejecución. * Otro proceso lo obligó a terminar.

Ready

Cuando el proceso está listo para ser ejecutado pero no hay recursos disponibles para que esto ocurra. Running

Cuando el proceso está siendo ejecutado Waiting/Blocked

El proceso no se puede ejecutar porque esta esperando algún evento externo a éste ocurra. (Por ejemplo esperando una operación de I/O).

Transiciones entre estados ![State$_{G}raph$](*https* : *//github.com/Gian*150/*talleres*−*so/blob/master/wiki*$_{d}$*ocument*

Jerarquía de procesos En algunos sistemas, cuando un proceso crea otro proceso, el proceso padre y el proceso hijo son asociados de alguna forma. El proceso hijo puede, a su vez, crear mas procesos. En Linux, un proceso, todos sus hijos y cualquier proceso que desciendan de estos forman un grupo de procesos.

Cuando el sistema empieza (en Linux), se lanza un proceso 'init' que lee en un archivo cuantas terminales debe haber y luego se "$_{f}orkea$" *enunnuevoprocesoporcadaterminal.Estosprocesosesperanaquealgúnusua*

Para ver como funciona el 'fork' en linux, ver [Ejercicio 1](https://github.com/Gian150/talleres-so/wiki/Ejercicio-1-(Taller-Ptrace)) del taller de Ptrace.

Parte 2: Comunicación entre procesos Hay dos tipos de procesos: * independientes * cooperativos

Un proceso $_{i}ndependiente_{n}oafectaniesafectadoporotroproceso, uno_{c}ooperativo_{p}uedeafectaroserafectadoporotr$

Los procesos cooperativos requieren un mecanismo de comunicación de interprocesos (IPC) que les permita intercambiar información. Algunas razones para permitir que procesos se comuniquen son: * Compartir información * Aumentar velocidad de computo de ciertas tareas que puedan ser divididas en subtareas que se puedan ejecutar en paralelo * Modularizar un proceso y separar sus funcionalidades Hay dos tipos fundamentales de modelos de IPC: **memoria compartida** y **transmisión de mensajes**.

Memoria compartida Cuando se usa memoria compartida, es necesario establecer la región de memoria que debe ser compartida y para que procesos este área será habilitada. El tipo de

información y donde esta información se guarda es definida por los procesos y no por el sistema operativo. Y, además, los mismos procesos son los que se encargan de asegurar que no están escribiendo la misma dirección de memoria simultáneamente.

Transmisión de mensajes La transmisión de mensajes provee una forma para que dos o mas procesos se comuniquen entre ellos sin la necesidad de compartir memoria. Si dos sistemas $_{P y Q q}uierencomunicarsedebeexistirunlinkdecomunicaciónentreambosprocesos.Estelinkpuedeserlogradodedistinta$

La transmisión de mensajes es particularmente útil en sistemas distribuidos donde los procesos que necesitan comunicarse pueden estar siendo ejecutados en distintas computadoras conectadas a una red.

Comunicación directa o indirecta * **Comunicación Directa Simétrica**: El proceso que quiere enviar el mensaje debe especificar al proceso destinatario y el proceso destinatario debe indicar cuál es el proceso que le debe mandar el mensaje * **Comunicación Directa Asimétrica**: El proceso que quiere enviar el mensaje debe espicificar el destinatario pero el destinatario puede aceptar mensajes de cualquier proceso. * **Comunicación indirecta** Los mensajes son enviados a un $_{m}ailbox_{.Un_{m}ailbox_{e}suncontenedordemensajesalqueciertosprocesospuedenenviarmensajesydelqueciertosprocesospuederecibirmensajes.}$

Sincronización El intercambio de mensajes puede ser sincronizado (bloqueante) o asíncrono(no bloqueante). * **Envío bloqueante**: El proceso que envía es bloqueado hasta que el mensaje halla sido recibido. * **Envío no bloqueante**: El proceso envía el mensaje y sigue con su ejecución. * **Recepción bloqueante**: El proceso se bloquea hasta que le llegue un mensaje. * **Recepción no bloqueante**: El proceso recibe un mensaje válido o no recibe nada pero sigue ejecutándose.

Parte 3: Anexo Taller IPC (Inter-Process Comunication) (Ver [Taller IPC](https://github.com/Gian150/talleres-so/wiki/Taller-IPC))

Estados de un proceso Introducción al scheduler E/S bloqueante / no bloqueante IPC

3. Scheduling

Objetivos de la política de scheduling Scheduling con y sin desalojo Políticas de scheduling Scheduling para tiempo real y para SMP

4. Sincronización entre procesos

Contención Condiciones de carrera Secciones críticas TestAndSet Busy waiting / sleep Productor - Consumidor Semáforos Introducción a deadlock Monitores Variables de condición

5. Programación Concurrente

Algoritmos wait-free Algoritmos lock-free CAS ABA Programación de multicores Invalidación de caché Reorden de instrucciones

6. Adminitración de memoria

Segmentación Paginación Swapping MMU Memoria virtual Copy-on-write Algoritmos de reemplazo de páginas

7. Adminitración de entrada/salida

Polling, interrupciones, DMA Almacenamiento secundario Drivers Políticas de scheduling de E/S a disco Gestión del disco (formateo, booteo, bloques dañados) RAID Copias de seguridad Spooling Clocks

8. Sistema de archivos

Responsabilidades del FS Punto de montaje Representación de archivos Manejo del espacio libre FAT, inodos Atributos Directorios Caché Consistencia, journaling Características avanzadas NFS, VFS

9. Protección y seguridad

Conceptos de protección y seguridad Matrices de permisos MAC vs. DAC Autenticación, autorización y auditoría Funciones de hash de una vía Encriptación simétrica RSA Privilegios de procesos Buffer overflows Inyección de parámetros Condiciones de carrera Sandboxes Principios generale de seguridad

10. Sistemas distribuidos

Taxonomía de Flynn Arquitecturas de HW y SW para sistemas distribuidos RPC Threads Pasaje de mensajes Orden parcial entre eventos Livelock Acuerdo bizantino Intuición de safety, liveness, fairness Algoritmo del banquero Panadería de Lamport Modelos de fallas y métricas de complejidad Exclusión mutua y locks distribuidos Elección de líder Instantánea global consistente 2PC

11. Conceptosavanzados

Virtualización Contenedores Cloud computing