**T. Conceptos básicos**

**SO tiene dos misiones**: gestionar los recursos de manera que los programas vean una máquina completa a su disposición y abstraer los detalles de los dispositivos hardware.

Un **proceso** es una instancia ejecutable de un programa y requiere código, datos y una serie de metadatos

La parte más importante del SO se llama **kernel** (núcleo). Se ejecuta en un modo privilegiado con acceso a toda la memoria, recursos y registros de la CPU.

Un **proceso** tiene código máquina ejecutable (llamado Texto), datos estáticos y funciones que se llaman usando el stack.

**Arranque sistema operativo: 1)Reset**: Cuando el ordenador se enciende, el microprocesador recibe una señal por el pin de RESET. El efecto de esta señal es dar a todos los registros un valor inicial, incluyendo el contador de programa. **2)Iniciador Hardware:** El equipo saltará a un pequeño programa grabado en ROM o Flash (BIOS/UEFI en un PC). Se conoce como POST (Power-on Self-Test). Al terminar busca un dispositivo de almacenamiento masivo (disco, USB) que tenga un sector de arranque configurado. Se carga en memoria el contenido del MBR (Master Boot Record) de la unidad de arranque y cede el control. **3) Carga del SO:** se carga la parte que está siempre residente. Todavía estamos en modo supervisor y con memoria real. El SO residente crea sus tablas de gestión (BCP, páginas de memoria, ficheros, etc…) y cuando las concluye con éxito habilita las interrupciones y crea un proceso especial llamado init en Unix.

**T. Gestión de procesos**

Un proceso es un programa en ejecución. El proceso consta de una **imagen de memoria**, compuesta por el código ejecutable, sus variables, el *heap*, el *stack* y las librerías que usa si son dinámicas (si son estáticas son parte del texto). También necesita otras informaciones: **1)Registros de la CPU:** contador de programa, puntero de pila… Se conocen como estado del procesador. **2)Entorno**: Datos globales que hereda el proceso al nacer. Son pares clave/valor que se pasan por el stack del proceso al crearlo. **3)Metadatos:** Datos sobre el proceso que produce el Sistema Operativo durante la ejecución. El **Bloque de Control de Proceso (BCP)** guarda la identidad del proceso, su prioridad, su estado, etc. La **tabla de páginas** guarda la correspondencia entre las páginas de memoria virtual y los marcos en los que está cargado el proceso. El **mapa de E/S** almacena información sobre los descriptores de dispositivos de entrada/salida y ficheros que usa el proceso. **4)Disco:** se guarda el código ejecutable del programa.

**Ciclo de vida de un proceso: 1)Creación:** El proceso solo se crea por petición expresa de otro proceso. Las operaciones necesarias para crear un proceso son: Asignar la imagen de memoria, Seleccionar un BCP libre en la tabla correspondiente del kernel y rellenarlo con los datos del nuevo proceso, Cargar el ejecutable y los datos estáticos inicializados y Crear la pila y el heap. Mientras duren estas tareas en el estado del proceso es **Nuevo**. Al completarse se marca como **Listo** y entra en la cola del planificador. **2)Activación:** La activación del proceso consiste en ponerlo en ejecución, de ello se ocupa un componente del *kernel* llamado *dispatcher*. **3)Interrupción:** La activación del proceso consiste en ponerlo en ejecución, de ello se ocupa un componente del *kernel* llamado *dispatcher*. **4)Finalización**: la finalización del código del usuario no significa que el proceso deje de existir de inmediato. Hay recursos del Sistema Operativo que deben liberarse. El proceso no muere de inmediato sino que pasa por un estado terminal llamado zombie.

Se **denomina cambio de contexto** al procedimiento de desalojar a un proceso de la CPU y sustituirlo por otro. Es consecuencia de una decisión del *scheduler.*

Un **proceso servidor** es aquel que está en un bucle infinito pendiente de recibir peticiones de los usuarios y atenderlas de forma atómica.

Un **demonio** es un proceso que se arranca en el inicio del sistema y es inmortal, o con más precisión reencarnante porque el sistema lo arranca de nuevo si termina de forma anormal.

Existen **procesos de sistema** que se ejecutan en modo privilegiado, como el proceso nulo, el demonio de paginación…

Los **threads** de un mismo proceso comparten código, datos (*heap* incluido) y recursos del Sistema Operativo (por ejemplo, ficheros abiertos) y tienen su propio contador de programa, registros y *stack*.

La **planificación del procesador** es una de las misiones más importantes del Sistema Operativo y consiste en decidir a qué proceso se le concede la CPU y cuándo debe expulsarse el que la está ocupando. Hay a largo plazo, a medio y a corto.

**Estados del proceso (Unix): 1)Nuevo:** Estado especial con el que el Sistema Operativo marca un proceso desde que se inicia su carga hasta que está terminado para ejecutarse por primera vez. **2)Listo:** Proceso que está preparado para ejecutarse. **3)En ejecución:** Estado del único proceso que ocupa la CPU. **4)Bloqueado:** El proceso está en una cola con otros procesos durmientes a la espera de que se complete la operación bloqueante. **5)Zombie:** Cuando un proceso termina no se borra de inmediato su BCP, para que el proceso padre pueda completar una llamada a wait(). Es un estado sin retorno y breve.

**Quantum:** tiempo máximo que un proceso puede ejecutarse de forma continuada habiendo otros en cola.

Todos los sistemas tienen un **proceso nulo**, que se ejecuta si no hay ningún proceso en la cola **Listo**.

**Algoritmo FCFS (First Come, First Served):** Es un algoritmo muy simple, el scheduler concede la CPU al proceso que lleva más tiempo en la cola Listo. El proceso se ejecuta de forma continuada hasta terminar. No hay quantum.

**Round Robin:** El algoritmo Round Robin es una mejora sobre el FCFS puro. Un proceso solo puede estar como máximo ejecutándose durante un múltiplo de ticks llamado **quantum**. Si llega a ese límite, el Sistema Operativo lo expulsa de la CPU, lo pone al final de la cola de procesos en estado **Listo** y concede a la CPU al proceso que lleve más tiempo en espera.

**Completely Fair Scheduler** (**CFS**): El CFS se basa en dos ideas brillantes, usar un árbol con todos procesos en estado Listo y un **tiempo de ejecución virtual** que transcurre a velocidades diferentes en función de las prioridades. De esta manera, se simplifica la estructura de control y se eliminan los problemas de **starvation**.

**T. Gestión de Memoria**

La jerarquía de memoria se basa en dos posibilidades estadísticas del código**: proximidad espacial y proximidad temporal.** La **proximidad espacial** se produce porque el programa se ejecuta de forma lineal la mayor parte del tiempo y por tanto las instrucciones de código máquina son consecutivas. La **proximidad temporal** se produce por la repetición de secuencias de código en bucles y llamadas a funciones.

La **memoria caché** es una memoria muy pequeña y muy rápida que reside en el mismo sustrato de silicio que la CPU, aunque no forma parte de ella. **Método de correspondencia directa:** Es el más simple, cada bloque solo puede instanciarse en una entrada determinada de la caché. **Método de correspondencia asociativa:** En la correspondencia asociativa cualquier bloque puede instanciarse en cualquier línea de la caché, con lo que se consigue una ganancia estadística en su uso frente a la correspondencia directa. **Método de correspondencia asociativa por conjuntos:** Los bloques se agrupan en conjuntos de tamaño fijo (potencia de dos) y cada conjunto se instancia siempre en las mismas líneas de caché. La flexibilidad surge porque dentro de las líneas del conjunto un bloque puede instanciarse en cualquiera de ellas.

**Políticas de reemplazo:** si la caché es de ***correspondencia directa*** la política es trivial, porque cada bloque solo puede cargarse en una línea concreta. Si es ***asociativa por conjuntos***, la decisión afectará solo a las líneas del conjunto correspondiente.

**Políticas de escritura: Write Through**: Cada vez que hay una escritura en caché, se reescribe el bloque en la RAM. **Write Back**: La línea solo se escribe en RAM si está sucia en el momento de ser reemplazada.

**Algoritmos de asignación de memoria real: First Fit:** asignado el primer hueco en el que cabía el proceso. **Best Fit:** Se busca el hueco más pequeño en el que cabe el proceso. **Next Fit:** Es igual que el Fist Fit, pero en lugar de recorrer la lista desde el inicio se empieza desde el último espacio asignado. **Worst Fit:** En lugar de buscar el hueco más pequeño disponible, se asigna el más grande. **Quick Fit:** En lugar de una lista se mantienen varias con huecos estandarizados de (2 kB, 10 kB, 40 kB, …). Al terminar un proceso hay que hacer un merge con el espacio de memoria libre más próximo. **El sistema operativo usa estos mismos criterios para gestionar el heap de un único proceso.**

La **TLB** es pequeña y muy rápida (parecida a una caché) y mantiene la relación entre números de página y marcos accedidos más recientemente. La TLB traduce direcciones virtuales a direcciones físicas, la caché solo maneja direcciones físicas.

La TLB se sitúa a la salida de la CPU, la caché entre la TLB y la memoria RAM.

**Algoritmos de reemplazo de página:** Página no usada recientemente (**NRU**): Se expulsa la página que lleva más tiempo sin usar. **FIFO**: Se elimina la página que lleva más tiempo en memoria. **Segunda oportunidad**: Una página referenciada gana una "vida" extra.

**La zona de swap:** es una partición especial del disco para grabar las páginas expulsadas o cargar desde ella las páginas de un nuevo proceso.

**T. Gestión de Ficheros**

La estructura de un fichero consiste en el mapa de agrupaciones que ocupa, es lo que se denomina **mapa del fichero.**

**Metadatos del fichero: Identificador:** clave única que identifica el fichero. **Nombre del fichero:** Cadena de caracteres que debe ser única dentro de un

mismo directorio. **Fechas**: de creación, último acceso y modificación. **Tipo de fichero:** regular, de entrada/salida, directorio, etc… **Mapa de fichero:** lista de agrupaciones ocupadas por el fichero. **Protección:** permisos del fichero. **Tamaño:** Bytes de información.

Un **volumen** coincide con un dispositivo físico, dentro de un volumen pueden definirse **unidades lógicas** (en Windows, C:, D:,…) o **particiones** en Unix.

El **superbloque** contiene información sobre la organización de la partición: Tamaño de la agrupación, tamaño del superbloque, bitmaps y zona de i-nodos, número de agrupaciones, número de i-nodos, número del primer i-nodo.

El **i-nodo** es la estructura más importante para la gestión de ficheros en Unix. Contiene la lista de los bloques ocupados por un fichero, además de algunos metadatos sobre permisos de acceso y fechas.

Hay dos **bitmaps**, uno para i-nodos y otro para agrupaciones.

**File Allocation Table** (FAT) es el sistema de registro de los bloques en MS-DOS. En FAT se usa una lista enlazada de agrupaciones (bloques) para llevar la contabilidad de bloques ocupados y libres.

**Sistema Virtual de Ficheros:** capa superior, del sistema de ficheros, que se añadió en Unix. Es transparente al programador.

El **Servidor de ficheros** es la parte del Sistema Operativo que ofrece al programador la abstracción **FILE**.

**Apertura de un fichero:** Sistema Operativo recorre el directorio hasta localizar el nombre encontrado. Si no existe devuelve el valor NULL. FILE* ADDRfile = fopen("accesos_memoria.txt", "r") //Solo lectura.

El mecanismo de Unix para poder ver todos los ficheros en un solo árbol se denomina **montaje.**

La **caché de ficheros** es una zona de RAM que sí gestiona el Sistema Operativo de forma directa.

**Políticas de sincronización: Write through** (inmediata): Un bloque se escribe en cuanto se modifica, **Write back** (escritura diferida): Los datos solo se escriben desde la caché a disco cuando ésta se ha llenado y hay que reemplazarlos, **Delayed write** (escritura con retardo): La sincronización se produce de forma periódica, **Write on Close** (escritura al cierre): En el cierre de un fichero siempre se produce una sincronización.

Cada vez que se llama a un servicio de disco, los cambios en sus metadatos se registran en el ***journal***. Si todas las operaciones de la transacción se cursan con éxito, los datos de esa transacción se borran del ***journal.***

**T. Entrada / Salida**

Sistema Operativo oculta los detalles más complejos del manejo de estos equipos electrónicos mediante la abstracción **dispositivo** (*device*), que es una ampliación de la abstracción **fichero.** La capa más próxima a los dispositivos son los **manejadores** (*drivers*).

Todos los dispositivos se comunican con la CPU. Si la conexión es compartida por varios dispositivos se denomina **bus**.

**Modos de programación E/S:** La solución más sencilla e ineficaz es la espera activa. Consiste en que el driver lee en un bucle infinito el registro de status hasta que cambia su contenido.

**DMA (Direct Memory Access):** evitar que los datos de estas transferencias masivas pasen por la CPU, sino que fluyan entre los dos dispositivos de una forma transparente. Esto es solo posible con este hardware.

En Unix, la entidad *device* puede ser de tres clases: **Dispositivos orientados a bloque**: Transfieren la información en bloques de información de tamaño fijo y además permiten acceso aleatorio. **Dispositivos orientados a carácter**: eran aquellos que enviaban o recibían secuencias de caracteres hacia o desde la CPU, como un teclado o una impresora. **Dispositivos orientados a red**: Nacieron más tarde, para atender los dispositivos de red, que tienen características mixtas.

Los **elementos internos de un SSD** son: Microcontrolador (Pequeña CPU), SDRAM (Banco de memoria auxiliar del microcontrolador), Interfaz (Realiza el diálogo con el driver), Bancos de memoria Flash y Channels (Buses internos de 8 bits que permiten el intercambio de información)

**T. Sincronización**

Las señales (signals) son un mecanismo de comunicación asíncrona del kernel con los procesos.

Los **pipes** (tuberías) nacieron durante el desarrollo de Unix para poder alimentar la salida estándar de un proceso a la entrada estándar del siguiente. Extensión funcional de esta idea son las **tuberías con nombre** (named pipes), también llamadas **FIFO.**

Hay dos tipos de socket, **datagram** o no orientados a sesión, soportados en UDP, y **stream** u orientados a conexión, que usan TCP.

La diferencia con los *datagrams sockets* es que en los primeros no hay concepto de conexión. El cliente hace una petición y el servidor contesta pero no hay garantía de entrega. TCP es un protocolo orientado a conexión que tiene que establecerse y cerrarse explícitamente.

La **concurrencia** se produce cuando se pide el uso de un recurso limitado por parte de varios peticionarios. Los recursos pueden ser: Físicos, lógicos, Reutilizables, consumibles, Exclusivos, compartidos, Expropiables o no expropiables.

Los procesos son **cooperantes** cuando están diseñados de forma premeditada para colaborar entre ellos o **independientes**, cuando la concurrencia es fortuita.

La cooperación puede ser **implícita**, cuando la orquesta el sistema operativo de forma opaca al programador.

Se dice que una operación es **atómica** cuando para el resto del sistema concurrente ocurre de forma instantánea.

**Sección crítica**: un fragmento de código que se ejecuta de forma ininterrumpida dese el punto de vista de los procesos/threads concurrentes.

La **condición de carrera** se produce cuando varios procesos o threads acceden a un mismo recurso (por ejemplo una memoria compartida) y el orden de llegada es puramente aleatorio.

El **interbloqueo** ocurre cuando un proceso retiene el recurso X y no lo libera a la espera de que otro proceso libere el recurso Y, que retiene pero no puede usar porque está a la espera del recurso X. Para que ocurra un interbloqueo se tienen que dar las cuatro condiciones de Coffman: **1)Exclusión mutua:** Hay un recurso por el que contienden los procesos y solo puede usarlo uno de ellos. **2) Condición de retención y espera:** Los procesos mantienen la posesión de los recursos ya asignados a ellos mientras esperan recursos adicionales. **3) Condición de no expropiación:** Cuando un proceso ocupa el recurso no se le puede quitar. **4) Condición de espera circular:** el tráfico oeste tiene que esperar a que se libere el sentido este y viceversa.

**Mutex**: un mecanismo bloqueante que funciona de forma atómica utilizando instrucciones de ensamblador. Tiene problemas: estampida y inversión de prioridad.

**spin lock:** es un mecanismo de bajísimo nivel del kernel que inhibe el *scheduler* y parte las interrupciones y permite la ejecución atómica casi pura.