



Universidad
Carlos III de Madrid

Grado en Ingeniería Informática

Curso 2019-2020

Diseño de Sistemas Operativos

Trabajo Final

Autor:

Alejandro Valverde Mahou 100383383

Índice

| | |
|--|-----------|
| 1. Introducción | 4 |
| 1.1. Introducción | 4 |
| 1.1.1. Interfaz del Usuario | 4 |
| 1.1.2. Gestor de Recursos | 4 |
| 1.1.3. Máquina extendida | 5 |
| 1.1.4. Estructura del Sistema Operativo | 6 |
| 1.2. Ejecución Asíncrona y Modular | 7 |
| 1.2.1. Ejecución Asíncrona | 7 |
| 1.2.2. Interrupciones <i>hardware</i> | 8 |
| 1.2.3. Módulos del kernel | 8 |
| 1.2.4. Bibliotecas | 8 |
| 2. Funcionamiento del Sistema Operativo | 9 |
| 2.0.1. Tratamiento de eventos | 9 |
| 2.0.2. Proceso de arranque del sistema | 9 |
| 2.1. Eventos | 9 |
| 2.1.1. Llamadas al Sistema | 10 |
| 2.1.2. Excepciones | 10 |
| 2.1.3. Interrupciones <i>software</i> | 10 |
| 2.1.4. Interrupciones <i>hardware</i> | 10 |
| 2.2. Procesos del núcleo | 10 |
| 3. Introducción a la Gestión de Procesos y Drivers | 10 |
| 3.1. Introducción a Procesos y Periféricos | 10 |
| 3.1.1. Procesos | 10 |
| 3.1.2. Periféricos | 11 |
| 3.2. C.C.V., C.C.I y Planificación | 12 |
| 3.3. Drivers y Servicios Ampliados | 14 |
| 4. Introducción al Diseño de un Sistema de Ficheros | 15 |
| 4.0.1. Marco de Trabajo | 16 |
| 4.1. Sistema de Ficheros | 16 |
| 4.1.1. Representación tipo Unix | 17 |
| 4.1.2. Representación tipo FAT | 17 |
| 5. Sistema de Memoria | 18 |
| 5.1. Introducción al Sistema de Memoria | 18 |
| 5.2. Gestión de Memoria | 20 |
| 6. Aspectos avanzados | 21 |
| 6.1. Seguridad en sistemas operativos | 21 |
| 6.2. Tolerancia a fallos en sistemas operativos | 22 |
| 6.3. Sistemas distribuidos | 23 |
| 6.4. Sistemas de tiempo real | 23 |

Índice de figuras

| | | |
|-----|---|----|
| 1. | Definición de Sistema Operativo [1] | 4 |
| 2. | Hipervisor [2] | 5 |
| 3. | Tipos de Máquinas Virtuales | 5 |
| 4. | Estructura del Sistema Operativo MS-DOS [3] | 6 |
| 5. | Estructura de un Manejador de Eventos | 7 |
| 6. | Programación de un Manejador de Ejemplos Básico | 7 |
| 7. | Interrupciones <i>Hardware</i> | 8 |
| 8. | Arranque Simplificado del Sistema Operativo | 9 |
| 9. | Esquema Simplificado de un Periférico | 11 |
| 10. | Diagrama de Estados de los Procesos con Cambio de Contexto Voluntario | 12 |
| 11. | Pseudocódigo del Cambio de Contexto Voluntario | 13 |
| 12. | Diagrama de Estados de los Procesos con Cambio de Contexto Involuntario | 13 |
| 13. | Arquitectura del Sistema de Entrada/Salida [6] | 14 |
| 14. | Estructura de un Módulo Driver | 15 |
| 15. | Sistema de Ficheros tipo Unix | 17 |
| 16. | Sistema de Ficheros tipo FAT | 18 |
| 17. | Mapa de Memoria de un Proceso | 19 |
| 18. | Esquema de la Arquitectura Moderna de Gestión de Memoria | 20 |
| 19. | Esquema de Comunicación Básico | 21 |
| 20. | Esquema de la Progresión de los Errores | 22 |
| 21. | Esquema de un Sistema Distribuido[10] | 23 |

1. Introducción

1.1. Introducción

El **Sistema Operativo** es el *software* principal de un sistema informático que gestiona el *hardware* y sus recursos, y provee servicios *software* a los programas de aplicación. Su función principal es permitir la comunicación entre usuario y ordenador, y gestionar sus recursos de manera eficiente.



Figura 1: Definición de Sistema Operativo [1]

El sistema operativo actúa como interfaz de usuario, gestor de recursos, y máquina extendida.

1.1.1. Interfaz del Usuario

Se divide en la interfaz del programador, que se realiza a través de llamadas al sistema y la interfaz del usuario, que se realiza a través de la interfaz gráfica o la línea de comandos.

1.1.2. Gestor de Recursos

Las áreas de gestión son:

- **Gestión de Procesamiento**

Se encarga de controlar el planificador de procesos, incluyendo prioridades y capacidades de multiusuario.

- **Gestión de Memoria**

Se encarga de repartir la memoria entre procesos, con protección y compartición.

- **Gestión de Almacenamiento**

Se encarga de la persistencia y etiquetación, independiente del medio físico, con una visión unificada para programas y usuarios.

- **Gestión de Dispositivos**

Se encarga de cubrir y abordar las dependencias de *hardware*, y de la gestión de accesos concurrentes.

1.1.3. Máquina extendida

Las abstracciones fundamentales usadas por los sistemas son los **Procesos** y los **Archivos**.

Los procesos se organizan en una jerarquía a través de los árboles de procesos, y son controlados a través del planificador.

Los archivos contienen las rutas para cada directorio y fichero.

Las **Máquinas Virtuales** virtualizan ciertas partes del *hardware*. El hipervisor realiza una virtualización de toda la máquina, de forma que varios sistemas operativos pueden ser ejecutados simultáneamente.

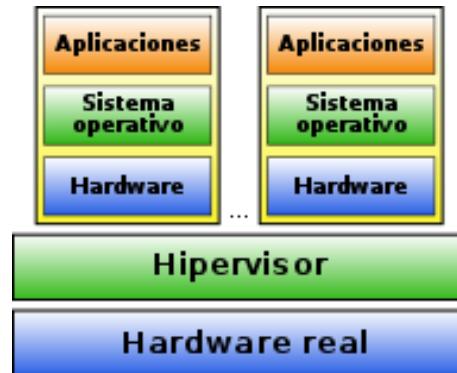


Figura 2: Hipervisor [2]

El hipervisor expande las capacidades *hardware*, de forma que cada sistema operativo puede trabajar simultáneamente con otros.

Existen diferentes tipos de máquinas virtuales, según sus capacidades.

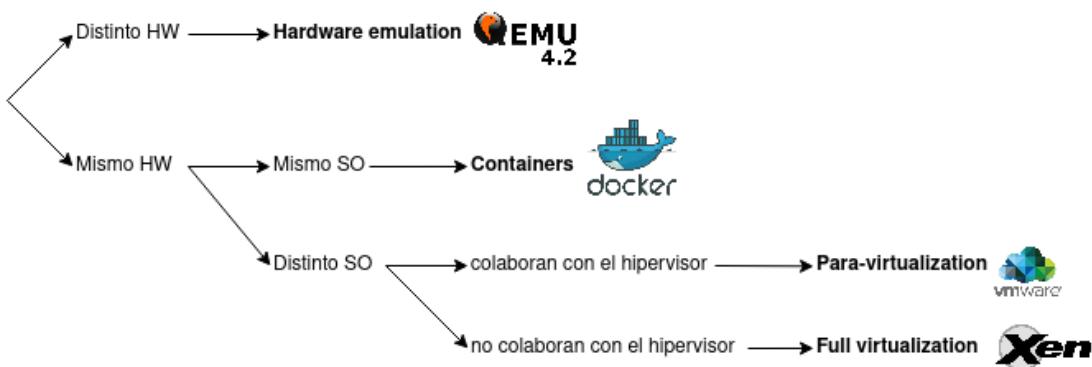


Figura 3: Tipos de Máquinas Virtuales

Las principales características de un sistema operativo son:

- **Versatilidad y portabilidad:** Puede ejecutarse en diferentes máquinas.
- **Adaptatividad:** Permite añadir y modificar módulos, en función de la demanda de los usuarios, evolución del *hardware* o aparición de distintos entornos.

- **Multidisciplinariad**: Integra trabajos de distintas áreas.
- **Complejidad**: Dado que realiza numerosas tareas diferentes, es un programa muy complejo y con muchas líneas de código.
- **Delicado**: Un fallo en cualquier parte del sistema puede provocar un colapso completo del sistema operativo.

Los objetivos en el diseño de un sistema operativo son:

- **Rendimiento**: Eficiencia y velocidad.
- **Estabilidad**: Robustez y resistencia.
- **Capacidad**: Prestaciones, flexibilidad y compatibilidad.
- **Seguridad y Protección**.
- **Portabilidad**.
- **Claridad**.
- **Extensibilidad**.

Estas características son incompatibles entre sí, por lo que en el diseño es necesario priorizar una característica sobre otras, buscando el equilibrio entre todas.

1.1.4. Estructura del Sistema Operativo

1. Monolítico

Es no estructurado. Desde cualquier punto del código se puede acceder a cualquier variable o función. Es difícil de mantener y muy sensible a los errores.

2. Subsistemas

Es un sistema monolítico compuesto de subsistemas lógicos. Se agrupan procedimientos y estructuras de datos relacionadas.

3. Por Capas

Sistema monolítico, pero estructurado por capas, de forma lógica. Cada capa proporciona acceso únicamente a la interfaz de niveles inferiores, de forma que no hay conexión entre capas no conectadas.

4. Microkernel

Una pequeña parte de las acciones se hacen en el kernel. El resto se hacen fuera, a nivel usuario.

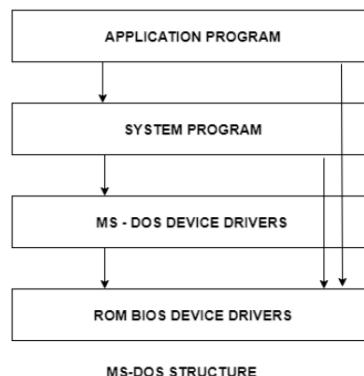


Figura 4: Estructura del Sistema Operativo MS-DOS [3]

1.2. Ejecución Asíncrona y Modular

Los sistemas operativos son sistemas orientados a eventos.

Los eventos interrumpen la ejecución normal del código, dejándolo en segundo plano y sin actuar, hasta que el manejador del evento termina su ejecución.

1.2.1. Ejecución Asíncrona

Las señales consta de un evento y un manejador.

```

1   int global1; // La comunicacion de funciones se usa a traves de variables globales
2
3   void handler( ... ){ // La funcion manejadora se encarga de tratar el evento
4     ...
5   }
6
7   int main( ... ){
8     ...
9     On(event1, handler1); // Se asocia el manejador al evento
10    ...
11  }
12

```

Figura 5: Estructura de un Manejador de Eventos

Ejemplo completo de señales:

El comando `signal()` añade una señal a un manejador.

El comando `sigaction()` asocia una señal a un manejador usando una máscara.

```

1   #include <stdio.h>
2   #include <signal.h>
3   #include <unistd.h>
4
5   int s1;
6
7   void sig_handler (int signo){
8     printf(" S ");
9   }
10
11  int main(void){
12    if (signal(s1, sig_handler) == SIG_ERR){
13      printf("\nNo se puede capturar SIGINT\n");
14    }
15
16    for(int i = 0; i < 5; i++){
17      if(i == 2 || i == 4){
18        s1();
19      }
20      printf(i);
21    }
22    return 0;
23  }
24

```

Figura 6: Programación de un Manejador de Ejemplos Básico

En el código anterior, el resultado obtenido sería: 0, 1, S, 2, 3, S, 4.

1.2.2. Interrupciones hardware

Cada periférico puede generar una interrupción y dispone de una línea **IRQ**(*Interrupt ReQuest*), conectada con el **PIC**(*Programmable Interrupt Controller*).

El **PIC** controla las líneas de **IRQ** para recibir las señales que envíen. Si llega una señal:

1. Asigna al **IRQ** del periférico un valor que guarda en **PIC**.
2. Avisa a la **CPU** a través de la línea de Interrupciones Pendientes(**INT**).
3. La **CPU** procesa el registro del **PIC**.
4. La **CPU** informa de que ha leído correctamente la interrupción.
5. El **PIC** anula la línea de interrupción procesada.

Por último, la **CPU** almacena la interrupción en el **IDT**(*Interrupt Description Table*), y guarda el estado actual de la pila, de forma que puede ejecutar la interrupción y recupera al finalizarla el contenido del proceso de la pila.

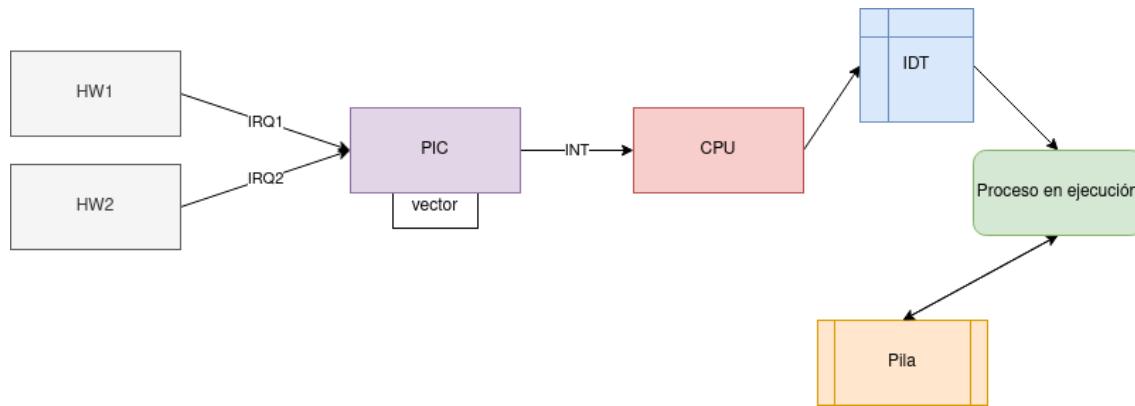


Figura 7: Interrupciones Hardware

1.2.3. Módulos del kernel

Los primeros *kernel* necesitaban incluir código para todo tipo de dispositivos, algo que actualmente es inconcebible debido al gran número de dispositivos que existen. Además, era necesario recomilar el kernel cada vez que se quería añadir un nuevo dispositivo.

Los módulos se crearon para poder superar estas dificultades, permitiendo al *kernel* añadir la inclusión condicional de controladores de dispositivos, los **drivers**.

Los módulos permiten añadir código de forma dinámica de un driver precompilado. Para poder ser distribuidos, se usan **bibliotecas dinámicas**(*.so/*.ko, *.dll).

1.2.4. Bibliotecas

Las bibliotecas son conjuntos de módulos objeto. Existen dos tipos de bibliotecas:

Biblioteca Estática: Es añadida por el compilador, al igual que los módulos objeto.

Biblioteca Dinámica: Es añadida en tiempo de ejecución, y por tanto debe informarse de su uso en la ejecución. Su diferencia principal reside en el uso de memoria. Si dos procesos usan la misma biblioteca estática, debe cargarse dos veces, mientras que si usan la misma biblioteca dinámica solo hace falta cargarla una vez. Por tanto, en caso de que varios procesos usan la misma biblioteca, las bibliotecas dinámicas ahoran memoria.

2. Funcionamiento del Sistema Operativo

El sistema operativo se ejecutará como programa ejecutable en el arranque del sistema.

El sistema operativo necesita que existan al menos dos modos de ejecución:

Modo privilegiado, o modo kernel: Tiene acceso a todo el espacio de memoria y a todas las instrucciones.

Modo ordinario, o modo usuario: Solo tienen acceso a parte del espacio de memoria e instrucciones.

2.0.1. Tratamiento de eventos

El sistema operativo espera de forma asíncrona y pasiva a la ejecución de los distintos eventos posibles.

Existen una serie de procesos, denominados **Procesos del Núcleo** que pertenecen al sistema operativo, y no a los usuarios, como el *firewall* o el antivirus. Estos procesos realizan tareas 'especiales' y son ocultos en funcionamiento al usuario.

2.0.2. Proceso de arranque del sistema

1. **Reset** pone todo a 0 y se cargan los valores iniciales en la **ROM**.
2. Se ejecuta el cargador del sistema, leyendo el trozo del **disco duro** en **memoria** y hace el *Power-On Self Test(POST)*, que comprueba que no haya ningún cambio en el *hardware* y se carga el **MBR**(*Master Boot Loader*).
3. El **MBR** va al **disco duro** y al **BL**(*Boot Loader*) que inicializa el kernel del sistema operativo.
4. Una vez el kernel se ha iniciado, se carga el sistema y todos los *daemons* necesarios.



Figura 8: Arranque Simplificado del Sistema Operativo

El **UEFI** es un programa de arranque que se aloja en una partición especial. Ni puede ser modificado y es muy flexible. Al combinar el **UEFI** con el **MBR** aumenta la complejidad y disminuye la seguridad.

Para acelerar el proceso de arranque se inicia en paralelo el *software* y el *hardware*.

2.1. Eventos

Existen cuatro tipos de eventos, que, ordenados de más cercanos a más lejanos al usuario son: Llamadas al sistema, excepciones, interrupciones *software* e interrupciones *hardware*.

Independientemente del *hardware* del sistema, se utiliza el mismo mecanismo: Antes de la microinstrucción *fetch*, se hace una comprobación de que no haya ningún evento pendiente. Si hay alguno pendiente, se entra en modo privilegiado y se realiza el evento. Cuando no queda ningún evento activo, se reinicia el estado de la pila y se vuelve al modo anterior.

2.1.1. Llamadas al Sistema

Son solicitudes de servicios al sistema operativo. Todas las llamadas suelen llamar a la misma función, y, a través de una tabla secundaria se redirige al comportamiento deseado. Este tratamiento se hace así para que la tabla auxiliar sea lo más compacta posible.

Desde el punto de vista de los usuarios, las llamadas al sistema son funciones, y para la CPU es lo mismo que si un periférico hiciera una interrupción, porque se simula una interrupción por *software*.

Se hacen siempre desde modo usuario, y son generadas por las aplicaciones. Son eventos síncronos de *software*

2.1.2. Excepciones

Son interrupciones de la CPU a sí misma para el tratamiento de rutinas especiales. Son los errores en la programación. Cuando se detecta una excepción, se salta a la subrutina asociada, y se comprueba si se ha generado desde modo usuario o modo kernel.

Si es desde el modo de usuario, y no se está en modo depuración, se aborta el proceso.

Si es desde modo kernel, se llama a la subrutina `panic()`, que para el sistema completamente (En *Windows* es el pantallazo azul). De esta forma se evita la sobreescritura indeseada de datos.

2.1.3. Interrupciones *software*

Se encargan de ejecutar las partes críticas llamándose desde el sistema operativo. Se resuelven todas las interrupciones por orden de urgencia, y no se vuelve al modo usuario hasta que no queda ningún evento por tratar.

2.1.4. Interrupciones *hardware*

Son eventos asíncronos *hardware*. Son formas de avisar al sistema operativo cuando se requiere de algún servicio *hardware*. Son requeridas varias subrutinas para cada evento.

Cuando se inicializa el sistema operativo, se establecen los manejadores de todas las posibles interrupciones, y, cuando hay una interrupción, se guarda el estado actual, se realiza la interrupción y si es necesario, se programa una tarea pendiente. Por último, se reinicia el estado del sistema.

2.2. Procesos del núcleo

Pertenecen al *kernel* y se ejecuta siempre en modo privilegiado. Se utilizan cuando se tiene que usar el sistema operativo en una operación concreta que funciona mejor siendo un proceso independiente. Suelen tener mayor prioridad que el resto de procesos. Un ejemplo de proceso de núcleo son aquellos procesos encargados de realizar las operaciones de bloqueo.

3. Introducción a la Gestión de Procesos y Drivers

3.1. Introducción a Procesos y Periféricos

3.1.1. Procesos

Un proceso es una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados, o, coloquialmente, un proceso es un programa en ejecución [4].

Los requisitos principales de un sistema operativo son:

- **Control de recursos** → En que lugares se encuentran los distintos recursos (código datos y pila), que archivos se encuentran abiertos...
- **Multiprogramación** → Poder tener varias aplicaciones cargadas en memoria, realizar cambios de contexto voluntarios para mejorar la eficiencia de uso del procesador...
- **Protección y compartición** → Impedir que un programa o proceso acceda a los datos de otro proceso, pero permitiendo la comunicación en ciertos casos.
- **Jerarquía de procesos** → Permitir operaciones que afecten a conjuntos o familias de procesos.
- **Multitarea** → Limitar el tiempo de actividad de los procesos no bloqueados, para permitir varios procesos actuando a la vez.
- **Multiproceso** → Poder ejecutar varios procesos al mismo tiempo, uno en cada core.

Los sistemas operativos cuentan con una serie de tablas, que actúan como pequeñas bases de datos, y son: **Tabla de Procesos**, que almacena el **BCP** de los procesos del sistema; **Tabla de Memoria**; **Tabal de Entrada/Salida** y **Tabla de Ficheros**.

El **BCP(Bloque de Control de Procesos)** es una estructura de datos que almacena la información necesaria para identificar y gestionar un proceso.

Los procesos pueden ser creados o en el arranque del sistema, o cuando un proceso existente hace una llamada al sistema para crear un proceso nuevo.

Un proceso puede terminar de forma voluntaria, o de forma involuntaria, cuando es finalizado por el usuario, el sistema u otro proceso.

3.1.2. Periféricos

Se consideran periféricos a las unidades o dispositivos de *hardware* a través de los cuales la computadora se comunica con el exterior, y también a los sistemas que almacenan o archivan la información, sirviendo de memoria auxiliar de la memoria principal [5].

Un periférico está compuesto de un dispositivo *hardware*, encargado de realizar la interacción con el entorno, y un módulo de entrada/salida, que actúa de interfaz entre el dispositivo y la CPU.

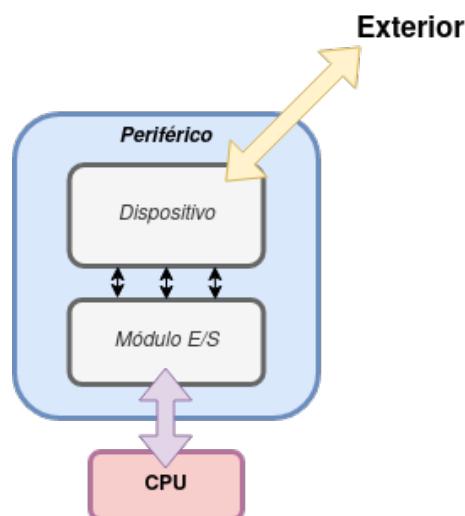


Figura 9: Esquema Simplificado de un Periférico

Es necesario incluir el modulo de entrada/salida porque:

1. Existe una gran variedad de periféricos, y una CPU no almacena la información requerida para cada uno de ellos.
2. Los periféricos son lentos, y su velocidad de transferencia es menor que la de la memoria o el procesador. Necesitan de algo que iguale esas velocidades.
3. Los formatos y tamaños no siempre coinciden entre los distintos periféricos

Los periféricos se pueden dividir por:

1. Direccionamiento de E/S
 - **Espacio de memoria conjunto**
Se puede acceder a las direcciones de memoria del controlador desde la memoria principal.
 - **Espacio de memoria separado (puertos)**
Se accede a los registros del controlador a través de instrucciones en ensamblador especiales.
2. Unidad de transferencia
 - **Dispositivos de bloque**
Dispositivos como cintas o discos, se acceden por bloques de *bytes*.
 - **Dispositivos de carácter**
Dispositivos como teclados o impresoras, se acceden por caracteres.
3. Interacción con computador
 - **E/S programada o directa**
Está constantemente comprobando si es necesario transferir información. Es lento y muy ineficaz.
 - **E/S por interrupciones**
Solo se actua cuando es necesario enviar la información (no hace comprobaciones constantemente). Mejora la eficiencia.
 - **E/S por DMA (*Direct Memory Access*)**
Solo se avisa cuando se han terminado todas las tareas que tenía encargadas. Se suele usar en dispositivos que se comunican por bloque. Requiere un mayor coste de *hardware*, pero mejora la eficiencia y velocidad.

3.2. C.C.V., C.C.I y Planificación

Para poder tener varias aplicaciones en memoria, es necesario dotar a los procesos de un estado. De esta forma, se tiene una lista de procesos listos, cuyo primer proceso pasa al estado ejecutando. El siguiente proceso no puede comenzar a ejecutar hasta que el proceso que está ejecutando se bloquee o termine.

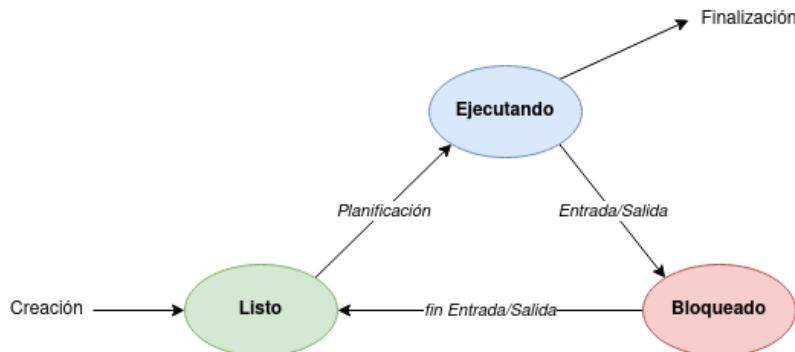


Figura 10: Diagrama de Estados de los Procesos con Cambio de Contexto Voluntario

Para implementar esta estructura y poder realizar **CCV**(*Cambio de Contexto Voluntario*), es necesario el uso de dos colas diferentes: **Cola de Listos** y **Cola de Bloqueados**. Cada una almacena los procesos listos y los procesos bloqueados correspondientemente.

```

1 void CCV () {
2     procesoActual->estado = BLOQUEADO;      // Primero se pone el proceso actual a bloqueado
3     insert (L_BLOQUEADOS, procesoActual);    // Y se inserta en la lista de bloqueados
4
5     proceso = procesoActual;                // Se almacena el proceso actual en una variable auxiliar
6
7     procesoActual = planificador ();        // Y se actualiza el proceso actual
8     procesoActual->estado = EJECUTANDO;    // Poniendose como ejecutando
9
10    // Por ultimo se realiza el cambio de contexto
11    cambioContexto (&(proceso), &(procesoActual))
12
13 }
```

Figura 11: Pseudocódigo del Cambio de Contexto Voluntario

El **planificador** es el encargado de decidir cual es el siguiente proceso que debe ser ejecutado, y la función **cambioContexto** actua como *activador*, y da control al proceso seleccionando anteriormente.

El **CCI**(*Cambio de Contexto Involuntario*) permite dividir el tiempo de los procesos, y alternar entre los procesos en ejecución, no exclusivamente cuando el proceso que se está ejecutando termina o se bloquea. De esta forma, según la planificación, se puede alternar entre ejecutando y listo.

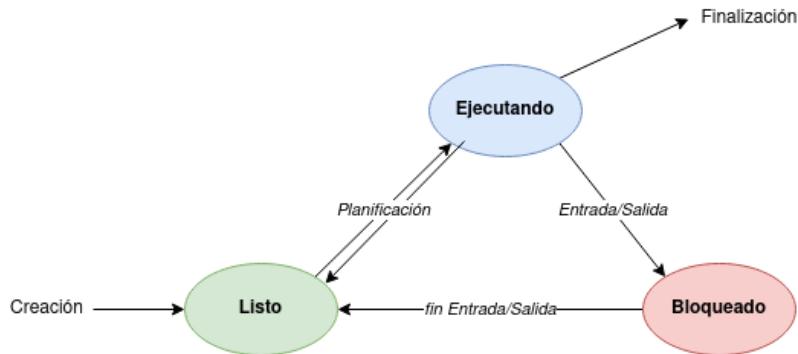


Figura 12: Diagrama de Estados de los Procesos con Cambio de Contexto Involuntario

Dentro de la planificación, existen 3 niveles:

- **Planificación a largo plazo:** Decide que procesos deben ponerse a ejecutar. Dado que es lento, se invoca con una frecuencia baja.
- **Planificación a medio plazo:** Es la que decide que procesos deben añadirse a la RAM.
- **Planificación a corto plazo:** Es el proceso más rápido, y que se ejecuta más frecuentemente. Decide qué procesos tiene la CPU.

Características de los planificadores:

- **Multitarea Apropiativa:** Puede ser con o sin expulsión. Si tiene expulsión, requiere de un reloj que limite el tiempo en ejecución de los procesos. Sin expulsión es CCV y con expulsión es CCI.

- **Clasificación de elementos en las colas:** Esta clasificación puede ser por prioridades, que tienen que ser añadida en cada proceso, o puede ser por tipo, que puede ser, por ejemplo, por más 'rachas' de tiempo usando la CPU.
- **Conocimiento de la CPU:** Aquí se puede tener en cuenta la *Afinidad*, o simpatía de un proceso con una CPU concreta, y la *Simetría*, o capacidades especiales de la CPU para ejecutar un proceso concreto.

Principales algoritmos de planificación:

- **Round Robin:** Asignación rotatoria de procesos. Se le asigna un tiempo máximo en forma de rodaja a los procesos en ejecución.
- **Por prioridad:** Se pone en ejecución el proceso con mayor prioridad en todo momento. Se suele combinar con otros algoritmos tras relajizar una división en prioridades.
- **SJF:** Primero el trabajo más corto. Se ordenan los procesos de menor a mayor.
- **FIFO:** El primero en llegar es el primero en ejecutar. Se ejecuta estrictamente en el orden de llegada.

3.3. Drivers y Servicios Ampliados

Los drivers son la parte del sistema operativo que se encarga de interactuar con todos los controladores posibles. Es la parte más cercana al *hardware*, y es la que permite el flujo de información con el mismo.

No permite la reutilización porque es muy dependiente del sistema operativo.

Como se está creando y añadiendo *hardware* nuevo continuamente, aparecen nuevos drivers continuamente. Debido a esto, y a su continua necesidad de actualizaciones, se utiliza una forma modular de implementación.

El objetivo de los drivers es optimizar lo máximo posible el uso del *hardware*, y permitir añadir y quitar dispositivos *hardware* dinámicamente.

La mayor parte del sistema operativo está ocupada por los drivers. Este código tiene acceso completo al sistema, de forma similar al kernel.

En Linux se pueden usar llamadas genéricas, porque todo se trata como si fuera un fichero. Esto permite sustituir el *hardware* sin necesidad de recomilar ni modificar el código.

Cada dispositivo *hardware* tiene un *ID mayor* que identifica el driver que usa, y un *ID minor* que identifica al componente *hardware* en concreto.

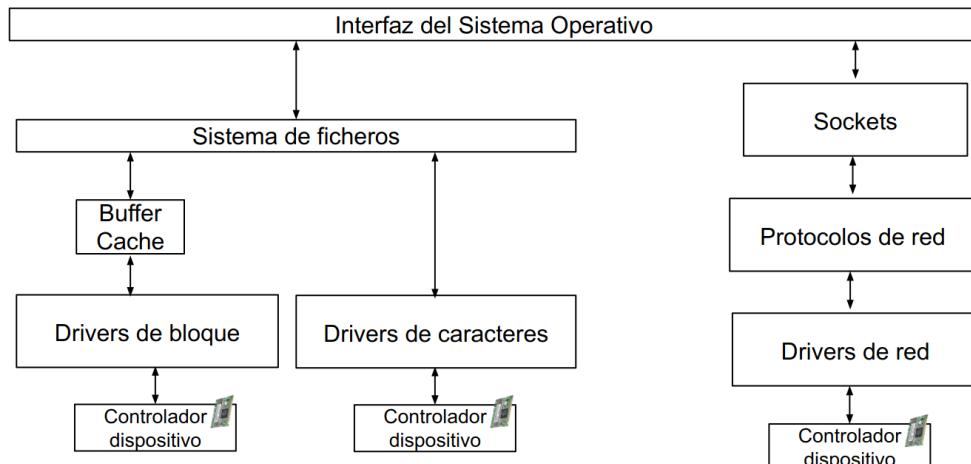


Figura 13: Arquitectura del Sistema de Entrada/Salida [6]

La clasificación clásica de los drivers es en 3 tipos diferentes:

- **Dispositivos de caracteres**
- **Dispositivos de bloques**
- **Dispositivos de red**

Los servicios ampliados son módulos utilizados para añadir y/o ampliar las funcionalidades de los drivers. Es un driver que comunica con otro driver, en lugar de comunicar con el *hardware*.

Los módulos driver tienen varios componentes diferenciados:

1. **Interfaz**: Comunica el módulo con los procesos.
2. **Específico**: Comunica el módulo con cada uno de los componentes *hardware*.
3. **Carga**: Comunica el módulo con el sistema operativo.
4. **Planificación de la Entrada/Salida**: Realiza los computos internos relacionados con la planificación.
5. **Inicialización y finalización**: Encargado de controlar si el driver se encuentra inicializado o terminado.

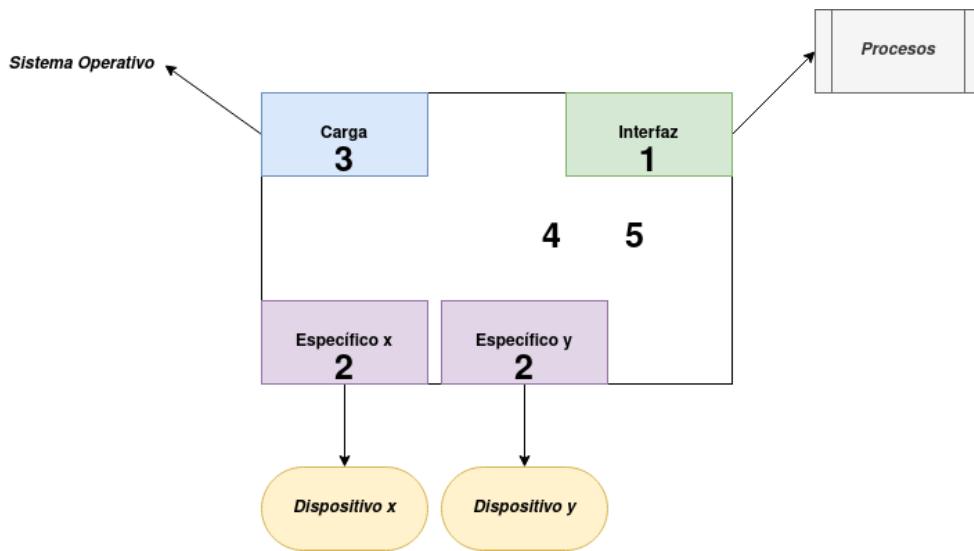


Figura 14: Estructura de un Módulo Driver

4. Introducción al Diseño de un Sistema de Ficheros

La computación y los datos están muy relacionados el uno con el otro, ya que la computación requiere de datos sobre los que trabajar, y los datos necesitan de computación para obtener conclusiones útiles.

Los datos deben estar almacenados en algún lugar dentro del ordenador. Este lugar puede ser la memoria principal o la memoria secundaria.

La **memoria principal** no guarda la información de forma persistente, tiene un poco tamaño y es muy rápida, mientras que la **memoria secundaria** si almacena los datos de forma persistente, tiene un mayor tamaño, pero es más lenta.

Hace falta un mecanismo que facilita la labor de comunicación entre las dos memorias. Y para evitar, en la medida de lo posible, que los programadores tengan que realizar asignaciones en memoria manuales, se utiliza una abstracción intermedia, en forma de sistema de ficheros y carpetas, controlada por el kernel.

Esta abstracción ofrece una visión lógica unificada, siendo lo suficientemente simple como para que su uso sea posible y útil, pero completa, para funcionar en la gran mayoría de máquinas actuales. Todos los distintos componentes del sistema operativo se almacenan de esta forma.

Este sistema es completamente transversal, es decir, tiene comunicación con todas las partes del sistema operativo.

4.0.1. Marco de Trabajo

Para acceder al sistema de ficheros que trabaja a nivel de bloque, es necesario usar la caché de bloques. Para ello, es necesario usar los siguientes comandos:

- `getblk`: Sirve para obtener un bloque.
- `brelse`: Sirve para liberar un bloque.
- `bwrite`: Sirve para escribir en un bloque.
- `bread`: Sirve para leer de un bloque
- `breada`: Sirve para leer de un bloque y del siguiente.

La caché, al recibir estas funciones, se encargará de llamar al driver manejador del dispositivo.

4.1. Sistema de Ficheros

Dentro de la memoria existen dos regiones diferentes: **Metadatos**, que permiten la localización de los datos de los usuarios, y los **Datos** de los usuarios.

Los metadatos tienen que estar almacenados tanto en memoria secundaria como en memoria principal.

El número de errores críticos dentro del sistema de ficheros debe ser mínimo, ya que la confianza de los usuarios en este sistema es muy grande.

Las estructuras que forman el sistema de ficheros son:

- **Metadatos**

- *Bloque de Arranque*
- *Superbloque*
- *Mapas de Bits*
- *Lista de Recursos Libres*
- *Indexación*

- **Datos**

- *Ficheros*
- *Directorios*
- *Enlaces*

4.1.1. Representación tipo Unix

El bloque de **i-nodos** almacena los atributos de entrada y la referencia a los bloques de índice para cada uno de los ficheros. Actúa como índice de los ficheros.

La **asignación de recursos** almacena el mapa de bits de los bloques, e indica si están ocupados o no.

El **superbloque** almacena el número de bloques usados para los i-nodos, para asignación de recursos, la dirección al i-nodo raíz, etc.

El **bloque de arranque** almacena el código de arranque y la tabla de particiones.

Los **ficheros** son referenciados desde los inodos. Se almacena en el i-nodo una referencia directa al principio del fichero, y una referencia indirecta al final del mismo.

Los **directorios** son referenciados también a través de los i-nodos, y almacenan información de los ficheros que contienen. Es tratado como un fichero especial.

Los **enlaces** se dividen en duros y blandos. Los duros son tratados como otro fichero, y tiene asignando su propio i-nodo, mientras que los blandos son tratados como accesos directos, y son accedidos desde el directorio al que pertenecen.

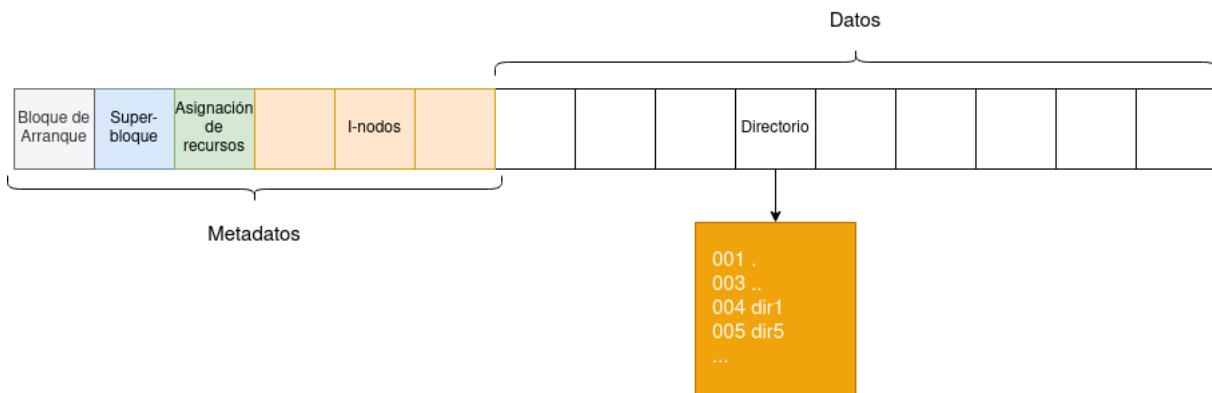


Figura 15: Sistema de Ficheros tipo Unix

4.1.2. Representación tipo FAT

Los bloques **FAT** almacenan la dirección a los distintos ficheros. El número que acompaña a FAT (FAT 12, FAT 16, FAT 32) indica el número de bloques de datos que almacena (2^{12} bloques, 2^{16} bloques, 2^{32} bloques). Además, este bloque FAT se encuentra duplicado, como medio de recuperación de datos perdidos.

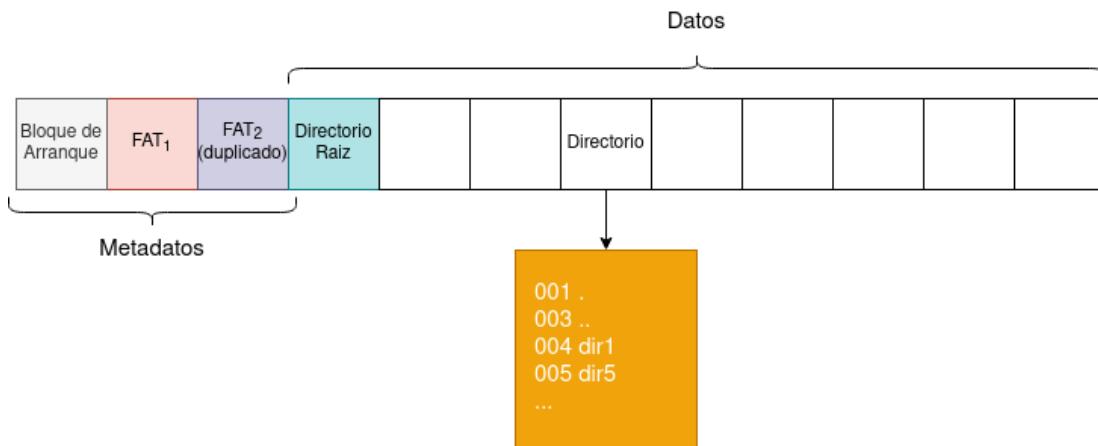


Figura 16: Sistema de Ficheros tipo FAT

En un mundo ideal, todos los metadatos se almacenarían en memoria, pero dado que esta memoria es limitada, tan solo se guarda la información estrictamente necesaria del superbloque , procesos e i-nodos de la partición en uso.

5. Sistema de Memoria

5.1. Introducción al Sistema de Memoria

Un programa es una secuencia de instrucciones escritas para realizar una tarea específica en una computadora [7]. Un proceso es un programa en ejecución, y su imagen en memoria es el espacio de memoria que ocupa ese proceso.

Los **Sistemas Monoprogramados** son aquellos que solo permiten la ejecución de un solo proceso simultáneamente. La memoria es compartida entre el sistema operativo y el proceso.

Los **Sistemas Multiprogramados** son aquellos que permiten la ejecución de varios procesos al mismo tiempo, gestionados por el sistema operativo. De esta forma se mejora la ocupación de la CPU.

Una **región de memoria** es una zona contigua de la imagen de un proceso, que poseen las mismas características o propiedades. Estas propiedades pueden ser:

- **Permisos:** Que pueden ser de lectura, escritura, ejecución o cualquier combinación de los anteriores.
- **Compartición entre hilos:** Que puede ser privada o compartida.
- **Tamaño:** Que puede ser fijo o variable.
- **Valor inicial:** Que puede ser con o sin soporte.
- **Creación:** Que puede ser estática o dinámica.
- **Sentido de Crecimiento:** Que puede ser ascendente o descendente.

Los procesos están formados por varias regiones.

Regiones principales:

- **Código:** Es donde se almacena el código fijo del proceso. Se encuentra en la parte inferior de la imagen.

- **Datos:** Es el sitio en memoria donde se almacenan las variables globales. Está dividido en dos subsecciones: región para variables con valor inicial, y región para variables sin valor inicial. Se encuentra a continuación del código.
- **Pila:** Es el lugar donde se almacenan las variables locales. Se encuentra en la parte superior de la imagen. Es de tamaño variable.
- **Región de Heap:** Actua de soporte para la memoria dinámica. Crece de forma ascendente.
- **Ficheros proyectados:** Región específica para los ficheros usados en el proceso. La región del proceso está asociada a un fichero. Se direcciona a memoria para poder acceder a la información necesaria.
- **Bibliotecas dinámicas:** Región que almacena las bibliotecas con código y datos.
- **Memoria compartida:** Región que almacena la información común de varios procesos.



Figura 17: Mapa de Memoria de un Proceso

El soporte para memoria principal del sistema operativo permite usar más memoria de la posible en el disco. Puede hacerse por medio de **paginación, segmentación o segmentación paginada**.

- **Paginación:** Dividir la dirección virtual en páginas, para trabajar a nivel de página en lugar de byte. Se divide la memoria en bloques de tamaño fijo, y se enlaza la memoria virtual y la memoria física.
- **Segmentación:** Dividir la memoria en secciones de tamaño variable denominados segmentos, en lugar de tamaño fijo. Se utiliza una tabla de segmentos para la correspondencia entre la memoria física y la memoria virtual.
- **Segmentación paginada:** Utiliza segmentos divididos en páginas. La tabla de segmentos lleva a una tabla de paginación, y después accede a memoria principal o a disco según corresponda. Dado que estas tablas son muy grandes, se utilizan tablas multinivel.

5.2. Gestión de Memoria

La gestión de memoria es el ámbito encargado de gestionar la memoria entre los distintos procesos y el sistema operativo.

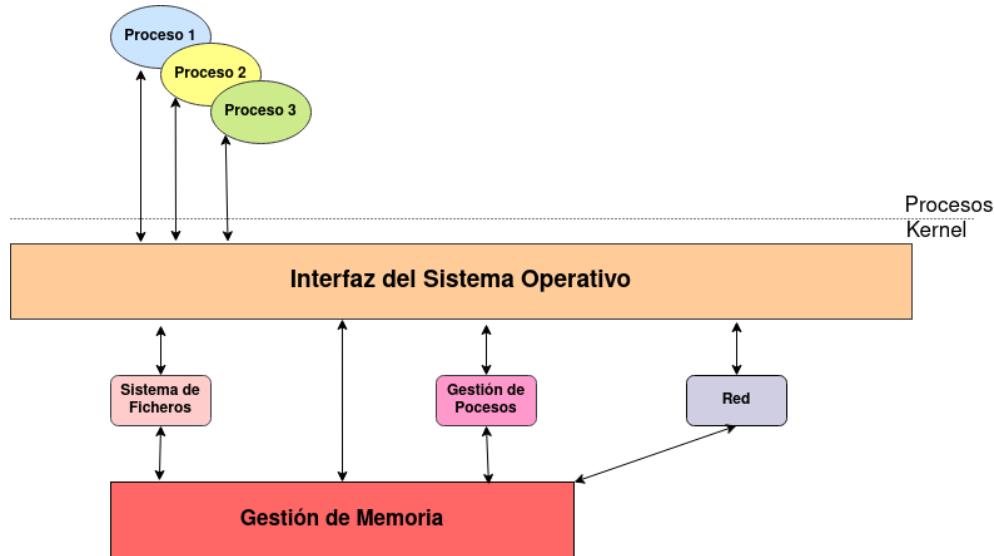


Figura 18: Esquema de la Arquitectura Moderna de Gestión de Memoria

Objetivos principales de la memoria:

1. Localizar las referencias a memoria, traduciéndolas como direcciones físicas.

Esto es necesario porque así se evita que el programador tenga que conocer dónde se va a colocar el programa dentro de la memoria al ejecutar.

2. Proteger los espacios de memoria, para evitar accesos a información de otros procesos sin los permisos necesarios.

Para conseguir esto, las posiciones de memoria deben ser comprobadas en tiempo de ejecución, y tienen que comprobarse por *hardware*.

3. Compartir los espacios de memoria comunes entre distintos procesos.

Esta función debe ser tanto solicitada como aceptada explícitamente, para evitar accesos a la información de los procesos de forma indeseada.

4. Realizar la organización lógica de los programas, dividiéndolos en módulos independientes.

Esto significa gestionar correctamente cada una de las regiones de un proceso, ya que todos los procesos no tienen las mismas regiones,

5. Realizar la organización física de la memoria, organizándola de forma óptima para alojar los diferentes programas y módulos.

Permite ejecutar procesos que requieren más memoria de la que hay disponible en un momento concreto, y evita la pérdida de memoria por fragmentación del disco.

La memoria a la que tiene acceso cada proceso consiste en un espacio de direcciones plano y lineal. Además, el espacio del *kernel* es compartido por todo el resto de procesos.

6. Aspectos avanzados

6.1. Seguridad en sistemas operativos

Los requisitos de seguridad de un sistema operativo utilizan los conceptos y bases de la **Teoría de la Comunicación**, y sigue un proceso similar al mostrado en el siguiente esquema:

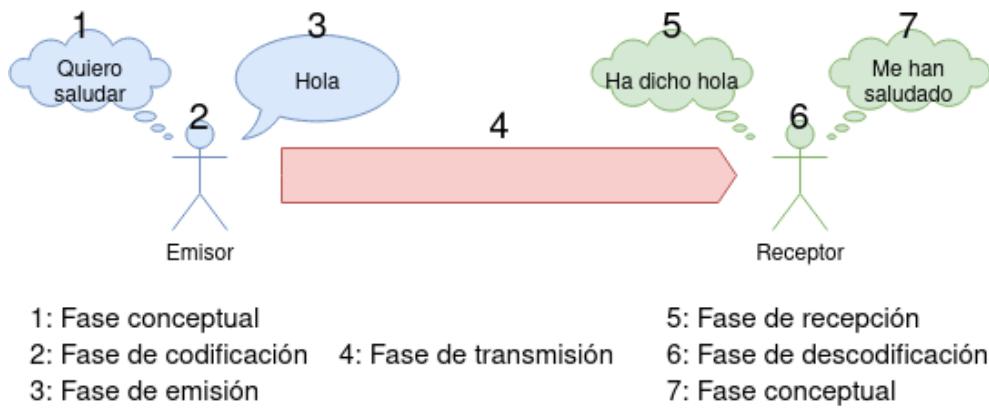


Figura 19: Esquema de Comunicación Básico

Los requisitos de seguridad de un sistema operativo son:

- **Confidencialidad**

Consiste en restringir el acceso a usuarios no autorizados a la información del sistema informático. Para ello, es necesario que el sistema sea capaz de autorizar o autenticar a los usuarios a través de su identidad.

- **Integridad**

Consiste en evitar que usuarios no autorizados puedan modificar cualquier elemento del sistema informático.

- **Disponibilidad**

Consiste en habilitar los elementos del sistema informático a los usuarios habilitados.

Existen dos formas de aplicar la seguridad informática: **Seguridad Física**, que utiliza medios físicos para aplicar la seguridad (Cerrojos, puertas...) y **Seguridad Lógica**, que utiliza mecanismos lógicos, como las contraseñas o las claves.

Las amenazas a la seguridad que sufren los sistemas operativos se pueden categorizar en 4 tipos:

- **Interrupción:**

Consiste en atacar la disponibilidad del sistema operativo, eliminando cualquier elemento que forma parte del proceso de comunicación.

- **Intercepción:**

Consiste en el acceso de un usuario no autorizado a un elemento del sistema operativo. Es un ataque que afecta la confidencialidad.

- **Modificación:**

Es un ataque que afecta a la integridad, y consiste en que un usuario no autorizado accede y modifica un elemento del sistema.

■ Invención:

También es un ataque que vulnera la integridad, y consiste en insertar archivos nuevos al sistema sin tener autorización.

A la hora de diseñar el sistema de seguridad de un sistema operativo es necesario tener en cuenta los principios de seguridad:

1. El principio de necesidad de saber

Es aquel que dice que solo debe mostrarse información a aquél que la solicita, y en el grado que la solicita, y de su autorización.

2. El principio de diseño intrínseco de la seguridad

Debe ser algo básico de todos los sistemas tener algún método de protección.

3. El principio del mínimo privilegio

Cada elemento del sistema debe tener el menor privilegio posible que le permita realizar sus tareas correctamente.

4. El principio del ahorro de mecanismos

Los sistemas usados para facilitar la seguridad deben ser tan pequeños y simples como sea posible.

5. El principio de aceptación

Los sistemas de seguridad deben evitar entrometerse en la realización de las tareas del sistema informático siempre que sea posible.

6. El principio de mediación total

El acceso al control de los sistemas de seguridad debe hacerse con la información de control necesaria.

7. El principio de diseño abierto

La seguridad de un dispositivo no debe depender de cuánto secreto es un algoritmo, si no de su funcionamiento correcto.

6.2. Tolerancia a fallos en sistemas operativos

La tolerancia a fallos es la propiedad que le permite a un sistema seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. Un diseño tolerante a fallos es un sistema que está capacitado para continuar su funcionamiento cuando algún componente del sistema falla[8].

El objetivo no es construir un sistema perfecto que no cometa fallos, ya que esto es algo imposible, si no un sistema que pueda seguir funcionando correctamente cuando se encuentra con un fallo. Estos sistemas suelen estar diseñados para tener una vida útil larga, que son difíciles de mantener, que tienen una disponibilidad alta, e implementan aplicaciones críticas.



Figura 20: Esquema de la Progresión de los Errores

6.3. Sistemas distribuidos

Un sistema distribuido es un conjunto de ordenadores que trabajan juntos de forma coordinada, a través del intercambio de mensajes, para conseguir un objetivo. En dicho sistema, el estado y los programas se guardan en múltiples ordenadores. A pesar de que los procesos que tienen lugar están separados entre los diferentes participantes, para el usuario parece que está trabajando con un único ordenador[9].

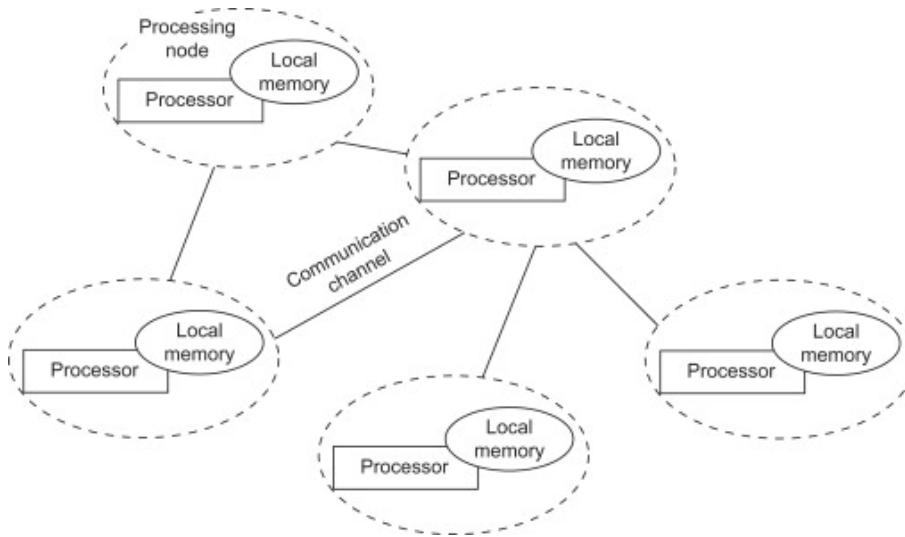


Figura 21: Esquema de un Sistema Distribuido[10]

Las principales ventajas y desventajas de los sistemas distribuidos son:

| Ventajas | Desventajas |
|--|--|
| Se comparten los recursos de <i>hardware</i> , <i>software</i> y de datos | Tiene una complejidad muy alta |
| La escalabilidad se hace añadiendo nuevo <i>hardware</i> | Depende en gran medida de la interconexión |
| Tiene un rendimiento muy alto gracias al gran número de procesadores | Posee un menor grado de seguridad |
| Tolerancia a errores, gracias al gran número de procesadores e información replicada | El <i>software</i> del sistema operativo es más complejo |
| Relación muy buena entre rendimiento y coste | Necesidad de repartir la carga de trabajo eficientemente |

6.4. Sistemas de tiempo real

Un sistema operativo de tiempo real es aquel que ha sido desarrollado para aplicaciones de tiempo real. Como tal, se le exige corrección en sus respuestas bajo ciertas restricciones de tiempo. Si no las respeta, se dirá que el sistema ha fallado. Para garantizar el comportamiento correcto en el tiempo requerido se necesita que el sistema sea predecible[11].

El procesamiento a tiempo real es aquel en el que la corrección y exactitud del sistema se mide en función del resultado obtenido y el instante en el que el resultado se produce. Para ello, necesita que los procesos se ejecuten a tiempo real, y tienen que reaccionar a sucesos externos.

Clasificación de los sistemas operativos de tiempo real:

■ ***Hard-real time***

El periodo de tiempo es muy importante y tiene que cumplirse estrictamente. Las tareas suelen ser críticas, y durante períodos de tiempo muy pequeños.

■ ***Soft-real time***

El periodo de tiempo es importante pero no estricto. Suelen ser tareas no críticas, durante un tiempo más largo.

■ ***Firm-real time***

El periodo de tiempo es muy importante, y una tarea fuera de plazo es inútil, pero no produce errores.

Requisitos de los sistemas operativos de tiempo real:

1. **Determinismo**

Los sistemas deterministas son aquellos que realizan sus operaciones siempre de la misma forma, ocupando el mismo tiempo. Es decir, no se ven involucrados factores aleatorios.

2. **Sensibilidad**

Hace referencia al tiempo que tarda el sistema en dar servicio a una interrupción, una vez ha sido reconocida.

3. **Control del usuario**

Esta característica es mayor en los sistemas operativos de tiempo real que en los sistemas operativos convencionales. En los de tiempo real, es necesario permitir a los usuarios decidir la prioridad de las tareas.

4. **Fiabilidad**

Es extremadamente necesaria, dado que responde a los sucesos que recibe al momento.

5. **Tolerancia a fallos**

Al igual que la fiabilidad, es extremadamente necesaria, y por el mismo motivo.

Referencias

- [1] Sistema operativo. En Wikipedia. Recuperado el 16 de mayo de 2020, de https://es.wikipedia.org/wiki/Sistema_operativo
- [2] Hipervisor. En Wikipedia. Recuperado el 16 de mayo de 2020, de <https://es.wikipedia.org/wiki/Hipervisor>
- [3] tutorialspoint, Operating System Structure. Kristi Castro. Recuperado el 16 de mayo de 2020, de <https://www.tutorialspoint.com/Operating-System-Structure>
- [4] Proceso (informática). En Wikipedia. Recuperado el 23 de mayo de 2020, de [https://es.wikipedia.org/wiki/Proceso_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Proceso_(inform%C3%A1tica))
- [5] Periférico (informática). En Wikipedia. Recuperado el 24 de mayo de 2020, de [https://es.wikipedia.org/wiki/Perif%C3%A9rico_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Perif%C3%A9rico_(inform%C3%A1tica))
- [6] Arquitectura del sistema de E/S. Lección 3c procesos, periféricos, drivers y servicios ampliados. Grupo AR-COS, Departamento de Informática, Universidad Carlos III de Madrid. Diapositiva 22. Recuperado el 27 de mayo de 2020, de https://aulaglobal.uc3m.es/pluginfile.php/3449191/mod_resource/content/18/dso-clases-3c-ppdsa_drv_sa-v5e.pdf (Puede que no sea visible a no miembros de la Universidad Carlos III de Madrid)
- [7] Programa informático. En Wikipedia. Recuperado el 30 de mayo de 2020, de https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico
- [8] Diseño de tolerancia a fallos. En Wikipedia. Recuperado el 31 de mayo de 2020, de https://es.wikipedia.org/wiki/Dise%C3%B1o_de_tolerancia_a_fallos
- [9] Medium, ¿Qué es un sistema distribuido?. Adrian Rodriguez. Recuperado el 31 de mayo de 2020, de <https://medium.com/@adr.rod87/qu%C3%A9-es-un-sistema-distribuido-442870a40bc4>
- [10] ScienceDirect, Distributed Computing. Dinkar Sitaram, Geetha Manjunath. Recuperado el 31 de mayo de 2020, de <https://www.sciencedirect.com/topics/computer-science/distributed-computing>
- [11] Sistema operativo de tiempo real. En Wikipedia. Recuperado el 31 de mayo de 2020, de https://es.wikipedia.org/wiki/Sistema_operativo_de_tiempo_real