Un **proceso** es un programa en ejecución, a cada proceso se le asigna un espacio de direcciones virtuales para albergar al proceso y, cada tanto, al proceso se le puede asignar memoria virtual y otros recursos, como canales de E/S, archivos, etc. Un proceso es un camino de ejecución a través de uno o más programas. Esta ejecución se puede intercalar con la de otros procesos. Está compuesto por varios threads (eventualmente uno)

Un **thread** es una unidad de procesamiento más pequeña, que puede ser planificada por un SO. Posee variables propias y acceso a la memoria y recursos del proceso. Tiene 3 estados: Listo – Ejecución – Bloqueado. Las ventajas son:

* Mejores tiempos de respuesta
* Compartir recursos
* Economía
* Sincronización de procesos

Como los hilos de un proceso comparten memoria la comunicación es más rápida. La estructura de un hilo está constituida por: PC, un conjunto de registros y un espacio de stack

**Hyper threading**: Permite al software programado para ejecutar múltiples hilos procesar los hilos en paralelo dentro de un único procesador. Simular dos procesadores lógicos dentro de un único procesador físico

**Tipos de hilos**:

* **ULT**: El trabajo de gestión de hilos lo realiza la aplicación a través de librerías. El kernel no se entera. Las ventajas son: Intercambio entre hilos, planificación independiente, portabilidad, mientras que las desventajas son: bloqueo del proceso durante un system call, no se puede multiplexar en distintos procesadores, no hay protección entre hilos.
* **KLT**: El kernel se encarga de la gestión y planificación. Las ventajas son: Se puede multiplexar el proceso en distintos procesadores, independencia de bloqueos entre threads de un mismo proceso. Las desventajas son: Cambios de modo de ejecución en el switch entre hilos del mismo proceso, la creación y administración de los KLTs es más lenta que los ULTs

Es posible combinar ULT y KLT, este enfoque aprovecha las ventajas de ambos tipos (LWP)

**Modelos de multithreading**:

* **Uno a uno**: Cada ULT mapea con un KLT, cuando se necesita un ULT se debe crear un KLT, ej: Windows
* **Muchos a uno**: Muchos ULT mapean a un único KLT, usado en sistemas que no soportan KLT, si se bloquea un ULT se bloquea el proceso, ej: JAVA
* **Muchos a muchos**: Muchos ULT mapean a muchos KLT, ej: Solaris

En Linux no existe el concepto de thread, es posible clonar un proceso para compartir recursos

Solaris maneja un esquema combinado de ULT y KLT

En Windows el modelo es uno a uno pero es posible el modelo muchos a uno utilizando fibras. Una fibra es una estructura similar a un thread, ejecuta instrucciones, no es mantenida por el kernel, es creada por el usuario y mantenida y administrada por código de usuario. Corren en el contexto de un thread, el usuario decide cuando cambiar entre fibras, el thread desconoce la existencia de las fibras y cuando este es eliminado también todas sus fibras. Desventaja de no poder ejecutarse en paralelo y ventaja de ser planificada por el usuario.

**Comunicación entre threads**: Entre threads es fácil compartir el espacio de direcciones, pero los problemas de superposición y dependencias siguen ocurriendo. Los procesos cooperativos sirven para compartir información, acelerar el cómputo y planificar tareas para que se puedan ejecutar en paralelo. Sin embargo, el acceso a los recursos globales es crítico, puede haber interbloqueo y los errores son difíciles de localizar.

**Condicion de carrera**: El resultado final depende del orden de ejecución. La existencia de mas de un procesador no garantiza que esto no suceda.

**Exclusión mutua**: Prohibir que más de un proceso lea y escriba datos compartidos al mismo tiempo. La sección critica es una parte del código que accede a recursos compartidos y que no se puede ejecutar mientras otro proceso este allí.

Respecto a la exclusión mutua no existen suposiciones respecto a las velocidades de los CPUs, en algún momento un procesador va a poder entrar (espera limitada). Un proceso que no está en su SC no puede bloquear a otra.

La solución a SC debe satisfacer exclusión mutua, continuidad y espera limitada.

**Técnicas por software**:

* **Busy waiting**: Se ejecutan continuamente las instrucciones de entrada a la SC hasta que se concreta. Se usa cuando hay expectativa razonable de que la espera será corta.
* **Variable candado**: Es una variable compartida que permite saber si un proceso puede estar en la SC. Posible condición de carrera
* **Solución de Peterson**: Arreglo de flags.

**Técnicas por hardware**:

* **Deshabilitar interrupciones**: El proceso se ejecuta hasta que se invoca una system call. Al no poder interrumpir el proceso se garantiza la exclusión muta. No sirve para multiprocesadores y peligroso su uso por parte de procesos de usuario. También conocido como elevar el nivel de procesador.
* **Test and Set**: Se ejecuta en su … , no puede ser interrumpido
* **Swap**
* **Spinlock**: Mecanismo que asocia una primitiva de locking a la estructura de datos que se quiere proteger. Se bloquea el uso de bus multiprocesador para proteger la ubicación de memoria que se está accediendo.

**IPC**: Mecanismo para sincronizar y comunicar procesos. Consta de mensajes, memoria compartida y semáforos.

**Mensajes**: Send y Receive. Hay un link entre los procesos que se quiere comunicar. Ese link puede ser unidireccional o bidireccional, simétrico o asimétrico.

Cuando la comunicación es directa el proceso debe especificar explícitamente el nombre de quien recibe o manda la comunicación, si es indirecta usa un mailbox o port, cada mailbox tiene una identificación única. El SO debe proveer el mecanismo para crear, compartir, enviar y recibir mensajes y destruir un mailbox. La capacidad del mailbox puede ser cero (Rendezvous: El emisor debe esperar a que el receptor reciba el mensaje para poder mandar otro), limitado (la longitud es finita) o ilimitado (tiene una longitud infinita y el emisor nunca espera).

**Naming asimétrico**: Send envía un mensaje a P y receive recibe un mensaje desde cualquier proceso. Propiedad del proceso: Está definido como parte de él o asignado directamente a él. Dueño es el que recibe los mensajes a través del mailbox. Usuario es quien envía los mensajes a ese mailbox.

El SO tiene un mailbox independiente.

**IPC**: Emisor y receptor pueden ser bloqueantes o no bloqueantes, si el mensaje ya se mandó el receptor lo recibe, si aún no se mandó se bloquea o continua sin recepción. La combinación más utilizada es con envió no bloqueante y recepción bloqueante. Si se bloquean las dos se llama Rendezvous.

En System V la función ftok() asigna un id, mecanismo para trabajar con IPC.

**Productor – Consumidor**: Hay un área de memoria común donde el productor deposita elementos y los consumidores toman elementos. La sincronización está dada por: El consumidor no debe tomar un elemento si el buffer esta vacío, el productor no debe colocar si el buffer está lleno.

**Semáforos**: Herramienta de sincronización que protege la sección critica. Es de tipo entero y tiene dos operaciones: P y V. Ambas operaciones son atómicas, es decir, cuando un proceso las ejecuta no puede haber otro ejecutándolas. Los semáforos sirven entonces para sincronización y exclusión mutua. Alternativa en la espera: Busy waiting, autobloqueo, cola asociada al semáforo.

**Memoria compartida**: Comúnmente cada proceso tiene su espacio de direcciones y no pueden acceder al espacio de otro. Esto se conoce como protección de la memoria. Sin embargo, con memoria compartida se puede lograr. Permite la transferencia de datos entre procesos, es decir comunicación. Requiere mecanismos de sincronización.

**Deadlock**: Un conjunto de procesos están en deadlock cuando cada uno de ellos está esperando por un recurso que está siendo usado por otro proceso del mismo conjunto.

**Clases de recursos**: Conjunto de instancias de un recurso. El ciclo es solicitud, uso y liberación

Para que exista deadlock se tienen que dar 4 situaciones:

* **Exclusión mutua**: Uso exclusivo del recurso
* **Retención y espera**: El proceso tiene el recurso hasta obtener todos los que le faltan
* **No apropiación**: No se pueden sacar los recursos a un proceso
* **Espera circular**

Existen tres métodos para el tratamiento de deadlock:

* Usar protocolo que asegure que nunca se entrara en deadlock
* Permitir el deadlock y luego recuperar
* Ignorar el problema y esperar que nunca ocurra un deadlock

**Prevenir**: Por lo menos una de las cuatro condiciones no podrá mantenerse. Se logra poniendo restricciones en la forma en que los procesos requieren los recursos.

**Evitar**: Asignar cuidadosamente los recursos, manteniendo información actualizada sobre requerimiento y uso de recursos.

**Prevención**:

* **Condición de exclusión mutua**: Si no se asigna recursos de manera exclusiva, no habrá deadlock. Pero deberá tener en cuenta que hay recursos compartibles y no compartibles. Mantener la exclusión mutua para los no compartibles.
* **Condición de retención y espera**: Antes de comenzar la ejecución se debe asegurar que el proceso tenga todos los recursos requeridos y la única manera de poder requerir recursos es cuando no tiene ninguno. Las desventajas son que hay baja utilización de recursos y posibilidad de inanición de alguno de los procesos
* **Condición de no apropiación**: Tenemos un recurso que no puede asignarse a un proceso y queda en wait y se liberan el resto de los recursos. Se desbloqueará cuando el recurso pedido más todos los que tenía estén disponibles. Otra opción: Si P1 solicita R1 y R1 lo tiene P2 que se encuentra bloqueado a la espera de R2 entonces R1 se lo quita a R2 y se le asigna a P1, Sino se bloquea P1
* **Condición de espera circular**: Se define un ordenamiento de los recursos. Luego un proceso puede requerir recursos en un orden numérico ascendente

**Evitación**: Hay mucho procesamiento lógico, dado que el SO tiene un detalle de cómo se requieren los recursos, en qué momento se requieren, la demanda máxima, etc.

**Estado seguro**: Si se pueden asignar recursos a cada proceso de un conjunto de alguna manera, evitando el deadlock. Entonces seguro garantizo que no hay deadlock. Estado de deadlock es estado inseguro. Pero no todos los estados inseguros son deadlock

**Algoritmo del banquero**: Se aplica para sistemas con múltiples instancias de cada recurso. Los procesos declaran el número máximo de instancias de cada recurso que necesitaría. Ese número no puede exceder el total de instancias de recursos de ese tipo en el sistema. El SO decidirá en que momento asignarlos, garantizando un estado seguro.

**Detección y recuperación**: Se requiere un algoritmo que examine si ocurrió un deadlock y otro para recuperarse del deadlock.

Si hay un ciclo hay deadlock. Si hay recursos con una sola instancia se usa el grafo de asignación, sino, el algoritmo del banquero.

Para recuperarse de un deadlock el operador puede resolverlo manualmente o esperar a que el sistema se recupere automáticamente, abortando todos o algunos procesos o hasta que el ciclo desaparezca o sacándole recursos a procesos

**Rollback**: Consiste en volver hacia una instancia segura el proceso que le sacamos los recursos. Para eso existen checkpoints. Se tiene un detalle de los recursos asignados en ese momento y los pasos de recuperación:

* Detectar el deadlock
* Detectar recursos que se solicitan
* Detectar que procesos tienen esos recursos
* Volver atrás antes de la adquisición del recurso
* Asignación del recurso a otro proceso

Otra forma es eliminar procesos: abortar todos o algunos procesos hasta que el ciclo desaparezca. Los criterios para elegir un proceso victima son:

* Menor cantidad de tiempo de CPU consumido hasta el momento
* Mayor tiempo restante estimado
* Menor cantidad de recursos asignados hasta ahora
* Prioridad más baja
* Lo ideal es elegir un proceso que se pueda volver a ejecutar sin problemas

**Protección y seguridad**: No son sinónimos. Mientras que la seguridad es un concepto más general, la protección se refiere a los mecanismos del SO para resguardar la información dentro de una computadora, para controlar el acceso de los procesos a los recursos existentes.

Proceso para acceder al sistema: Se resuelve a través de autenticación con control sobre una base de datos de usuario

Proceso para acceder a recursos del sistema: Se resuelve a través de permisos, control de accedo y capacidad (o listas de control de acceso)

**Políticas**: Definen lo que quiero hacer, en base a mis objetivos

Mecanismos: Definen como lo hago

Una política, múltiples mecanismos

**Objetos:** Recurso de hardware o software. Tienen id único. Los procesos aplican operaciones sobre los objetos

Dominio: Conjunto de pares (objeto, derecho). Cada par especifica un objeto y un subconjunto de operaciones que se pueden realizar con el

**Principio de need to know o POLA**: que los procesos accedan solo a los objetos que necesitan para completar su tarea. La relación entre un proceso y un dominio puede ser estática (conjunto de objetos a los que accede es fija) o dinámica (varia)

**Matriz de acceso**: Controla la pertenencia de objetos a dominios y sus derechos

Operaciones:

* **Switch:** Un proceso puede cambiar de un dominio a otro
* **Copy:** Se asocia a una celda de la matriz. El proceso ejecutándose en ese dominio puede copiar los derechos de ese objeto a otro dominio
* **Owner:** Permite agregar nuevos derechos y borrar ya existentes
* **Control**: Indica que pueden modificarse y borrarse derechos dentro de una fila

**Lista de acceso por objetos**: Para cada objeto una lista de control de acceso

**Lista de capacidades**: Es una lista de objetos del dominio con sus derechos

**ELF**: Formato de archivo para ejecutables, estándar en formato de archivo, capacidades de linkeo dinámico y carga dinámica, facilita la creación de librerías compartidas y la representación del archivo es independiente de la plataforma

**Tipos:**

* **Reubicables**: Contiene código y datos en un formato adecuado para ser linkeado con otros objetos para crear un ejecutable u objeto compartido
* **Ejecutables**: Es un formato para archivos aptos para ejecución
* **Objetos compartidos**: Mantiene código y datos en un formato adecuado para ser linkeado a dos contextos

**Secciones del ELF:**

* **Header**: Reside al comienzo del archivo, describe la organización del archivo
* **Sections y segments**: Son divisiones lógicas dentro del archivo que agrupan la información, las secciones son utilizadas durante el linking y los segmentos durante la ejecución. Las sections son una colección de información de tipo similar, cada sección representa una porción del archivo. Los segments tienen significado para el kernel, los utiliza para ejecutar un programa, puede corresponderse con una o varias sections

**Tabla de símbolos:** Es una lista de todos los símbolos que se definen o referencian dentro del archivo

**Tabla de strings**: Es una lista de los strings que se utilizan dentro del archivo

Relocations: Información necesaria durante el proceso de linking dinámico

**PE**: Ejecutable portable. Se utiliza en las diferentes versiones de Windows. Capacidad de linkeo dinámico, carga dinámica y facilita la creación de librerías compartidas. La información está representado por un conjunto de estructuras de datos, estas estructuras son las mismas que se utilizan para llevar el archivo a memoria

**Formato**: Conjunto de headers y secciones que indican al Loader como cargar el archivo en la memoria.

**Secciones**: División lógica dentro del archivo que contiene código y datos

**RVA**: Cada archivo tiene una dirección preferida de carga en el espacio de direcciones. Se utilizan RVAs para direccionar elementos del archivo, indica un desplazamiento en memoria, relativo a la dirección de inicio donde el contenido del archivo PE es cargado en la memoria virtual

**Grid**: La idea del procesamiento distribuido es compartir potencia de procesamiento. Para compartir ciclos de CPU se debe contar con una infraestructura distribuida. La idea de Grid es muy parecida a una red eléctrica, cualquier hardware que tenga conexión a la red debe poder ser integrada a la Grid. Tecnología para usar coordinadamente todo tipo de recursos en forma no centralizada

**Middleware:** Es una capa intermedia entre aplicaciones de usuarios y recursos computacionales y de almacenamiento, asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, hardware, etc.

**Desventajas**: Recursos heterogéneos, administración de recursos son procesos que deben controlarse externamente, las aplicaciones deben estar preparadas para ejecutarse sobre una GRID, precio de los recursos

**Ventajas**: Colaboración entre grupos dispersos geográficamente, funcionamiento de aplicaciones a gran escala

**Virtualización:** Capa de abstracción sobre el hardware que permite obtener una mejor utilización de los recursos. Tantos SO como máquinas virtuales.

Cada máquina virtual tiene sus propios recursos virtuales. El SO guest ve un conjunto consistente de hardware, no el real. Se encapsulan en archivos, por lo que es fácil de almacenar y copiar y pueden moverse de un servidor a otro rápidamente.

**Emulación ¡= Virtualización**

El primer concepto permite toda la funcionalidad del procesador deseado a través de software, pero es lento.

La virtualización trata de particionar un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador, es más rápido.

En la virtualización, el guest emitirá una instrucción privilegiada que no debe ser ignorada.

Las instrucciones privilegiadas generan interrupción.

Las sensibles solo se pueden ejecutar en modo kernel

**Hypervisor**: Plataforma de virtualización que permite múltiples SOs corriendo en un host al mismo tiempo. Interactúa con el hardware y realiza la multiprogramación

**Tipo I**: Se ejecuta en modo kernel. Cada máquina virtual se ejecuta como un proceso de usuario en modo usuario, modo kernel virtual y modo usuario virtual, cuando la máquina virtual ejecuta una instrucción sensible se produce una trap que procesa el hypervisor. Se ejecuta sobre el hardware

**Tipo II:** Se ejecuta como un programa de usuario sobre un SO host, interpreta un conjunto de instrucciones de máquina. El SO host es quien se ejecuta sobre el hardware. Las instrucciones sensibles se sustituyen por llamadas a procedimiento que emulan las instrucciones

**Paravirtualizacion**: Se trata de tener SO guests modificados, para mejorar el rendimiento. El SO guest es como un proceso de usuario que hace llamadas al SO

**Multiprocesadores con memoria compartida**: Dos o más CPUs comparten el acceso a la RAM. Los programas se ejecutan en cualquier CPU. Hay sincronización de procesos, administración de recursos y planificación. Cada procesador puede direccionar toda la memoria

**Velocidad de acceso**:

* **UMA**: Arquitectura basada en bus. CPUs y RAM usan el mismo bus, por lo que se debe esperar a que el bus no este ocupado. Es aceptable cuando hay pocas CPUs. Solución uso de caches y memorias privadas.
* **NUMA**: Único espacio de memoria visible por todas las CPUs. El acceso a la memoria remota es más lento que el acceso a la memoria local

**Tipos de SO**:

* **Cada CPU con su SO**: Se comparte el código de SO
* **Maestro – Esclavo**: Única copia del SO y de su información. Todas las llamadas del SO se redirigen a una CPU, con muchas CPUs hay cuello de botella en el maestro
* **SMP**: Única copia del SO en memoria, cada CPU puede ejecutarlo, equilibrio entre procesos y memoria, no hay cuello de botella, el problema es que dos o más CPUs ejecutando código del SO en un mismo instante de tiempo. Utilizar locks para las estructuras del SO, todo el SO como una gran sección critica o lock por estructuras, mejora el rendimiento, problemas de deadlock

**Multicomputadora**: CPUs con acoplamiento fuerte, no se comparte memoria, cada CPU tiene la suya, PCs con interfaz de red de alto rendimiento