El **SO** es un programa que administra el HW, gestiona recursos y controla la ejecución de los programas para que no surjan problemas. Cuando se ejecuta un programa se crea un **proceso** que es la parte activa de este, y que contiene un conjunto de instrucciones y de datos asociados.

El SO controla y gestiona todos los procesos en ejecución mediante sus **BCP**. Este bloque es único para cada uno, que se crea cuando se ejecuta el proceso y se destruye cuando termina. Tiene un conjunto de entradas: ID, estado (punto del ciclo de vida), prioridad, contador de programa (siguiente instrucción), datos de contexto (información contenida en los registros del procesador), apuntadores de memoria, información de D-E/S, información de contabilidad, etc.

La asignación del procesador a un proceso se hace mediante **planificadores** de procesos. El PLP se encarga de crear el BCP, cargar el proceso y destruirlo. El PCP se basa en un conjunto de algoritmos y técnicas para asignar el procesador a los procesos en forma eficiente y justa. El PMP controla el estado de la MP.

El PCP puede trabajar con algoritmos apropiativos, con el que se dan cambios de contexto, o no apropiativos, es decir que cuando un proceso se ejecuta no libera al procesador hasta que termina. Algoritmos no apropiativos: FIFO y SJN. Algoritmos apropiativos: RR y SRT. Prioridad y HRN permiten ambas políticas.

Para controlar el **estado** en que se encuentra un proceso de su ciclo de vida, se utiliza un modelo de estados:

- 2 estados: Cuando se crean los procesos se ubican en NE y cuando el SO determina que el proceso puede ejecutarse, se le asigna el procesador y pasa a E. Si el proceso se interrumpe vuelve a la cola de NE, si no pasa a la salida. La interrupción evidencia la multiprogramación. Los algoritmos de políticas no apropiativas utilizarían este modelo sin la flecha de interrupción.

-5 estados: Los procesos creados se ponen en estado Nuevo, cuando el SO determina que es posible admitirlos los pasa a la cola de Listos para asignarle el procesador. Si el proceso está en espera de un determinado suceso, se lo pasa a estado Bloqueado hasta que se lleve a cabo dicho suceso. Cuando esto sucede se lo pasa a Listo nuevamente para competir por el procesador.

- 7 estados: Al modelo anterior se le agrega dos estados más para liberar espacio en MP. Cuando el SO detecta una posible congestión en MP, pasa algunos procesos en estado Bloqueado a Bloqueado Suspendido en MS. Cuando ocurre el evento esperado, el proceso pasa a Listo Suspendido para luego ir a la cola de Listos cuando el SO determine que la MP tiene espacio libre.

Los procesos en ejecución pueden llegar a tener recursos compartidos. En este caso se debe analizar la condición de **concurrencia** sobre los procesos que compartan un mismo recurso para establecer un orden de ejecución.

La parte del código que involucra a un recurso compartido se denomina sección crítica, la cual no se puede ejecutar cuando otro proceso se encuentre en su sección crítica por el mismo recurso. Para que esto se cumpla, el SO debe garantizar una **exclusión mutua**, cuando ambos procesos quieran acceder a este recurso mediante mecanismos de concurrencia.

Para que se cumpla la exclusión mutua se deben garantizar las siguientes condiciones: nunca dos procesos en concurrencia pueden estar en su SC al mismo tiempo, no se deben realizar suposiciones sobre velocidad de ejecución de los procesos, ningún proceso fuera de su SC debe impedir que otro acceda a su SC, ningún proceso puede acceder por si solo a su SC.

Puede ocurrir que ninguno de los procesos tenga los suficientes criterios o atributos para apropiarse del recurso y continuar con la ejecución. Por eso, el sistema queda improductivo por tiempo indefinido, a lo que se denomina **interbloqueo**.

Para que exista el interbloqueo se deben cumplir estas 4 condiciones: Exclusión mutua, posesión y espera, no apropiación y espera circular. Si hay interbloqueo, el SO puede ignorar, prevenir, evitar o detectar esta situación:

- Para ignorar se vuelve a arrancar el sistema.

- Para prevenir, no debe permitir que una de las 4 condiciones se cumpla, por ejemplo: permitir accesos de solo lectura a archivos compartidos, no iniciar los procesos hasta que tenga todos los recursos necesarios, aplicar prioridad para apropiaciones de recursos, y enumerar los recursos en forma creciente para la no espera circular.

- Se puede evitar el interbloqueo si se tiene cierta información de los recursos que necesita el proceso mediante los siguientes métodos: grafos para analizar las asignaciones de recursos, algoritmo de banquero (cada proceso declara cuantos recursos va a utilizar), algoritmo de evitación (examina las asignaciones para evitar la espera circular).

- Detectar y recuperar consiste en eliminar los procesos de menor prioridad, los que llevan menos tiempo en ejecución, los que se restauraran más rápido, y tratar de liberar los recursos más críticos o con menos procesos asignados.

Un proceso está formado por un **hilo** y un proceso o tarea. El hilo es la parte que se ejecuta y la tarea es la unidad de propiedad de recursos que contiene la imagen del proceso.

Multihilo es la capacidad que tiene el SO de procesar varios hilos de un solo proceso. Todos los hilos de un proceso comparten el espacio en memoria, el estado, los recursos y los datos del proceso. Sigue existiendo un único BCP, pero cada hilo tiene su BCH, un estado de ejecución de hilo (listo, ejecución, bloqueado), un contexto de hilo, una pila de ejecución, almacenamiento estático para sus variables, y acceso a la memoria y a los recursos de su proceso.

Al compartir el mismo espacio de direcciones y los recursos, los hilos deben estar sincronizados ya que si hay una alteración de un recurso por parte de cualquiera de los hilos, afectaría al resto de los hilos. Por esto también todos los hilos deben pasar a estado suspendido al mismo tiempo, y si la tarea termina, todos los hilos dentro de esta también.

Los beneficios de trabajar con hilos están en el rendimiento. Lleva menos tiempo crearlos dentro de un proceso existente que crear un proceso totalmente nuevo, lleva menos tiempo finalizarlos y hacer cambios de contexto, y es mejor la comunicación entre diferentes programas.

**Colas múltiples**: Se divide a la cola de procesos en múltiples colas, las cuales tienen diferentes algoritmos de planificación. Cada proceso será asignado a una cola dependiendo de su tipo, necesidades, prioridades y tamaño requerido en memoria. Existe un algoritmo apropiativo entre colas que selecciona la cola que va a utilizar el procesador.

**Colas múltiples con retroalimentación**: Se divide a los procesos en varias colas jerárquicamente ubicadas por prioridad. Todos los procesos que llegan se colocan en la cola de mayor prioridad, y luego van a ser promovidos a una de menor prioridad hasta que terminen, regulados por el algoritmo de cola y entre cola. Generalmente el mecanismo entre colas es RR apropiativo.

En las primeras dos **generaciones** no existía el SO, si no que se trabajaba con tarjetas perforadas y procesamiento de los procesos en serie o lotes.

En la segunda generación aparece el SO de tiempo compartido, junto con los conceptos de multiprogramación y multiprocesamiento. Este SO permite que varios usuarios ejecuten programas en simultáneo, y le da la sensación a este de que la computadora es para él solo. Esto evidencia la multiprogramación y el particionamiento de memoria, es decir que varios programas usuarios se encuentran en MP y utilicen un solo procesador. Multiprocesamiento consiste en utilizar varios procesadores al mismo tiempo en un mismo sistema.

En la tercera generación se empiezan a aprovechar los tiempos de espera de operaciones de E/S con la técnica de Spooling. Se protege al SO en MP y surge el SO en tiempo real, el cual requiere una respuesta inmediata de los procesos.

La PC aparece en la cuarta generación, junto con una interface amigable para usuarios no profesionales y de bajo costo en HW. Se implementan los SO en red, distribuido y paralelo:

- El SO en red permite la interconexión de diferentes computadoras, que reportan a un SO central, para poder acceder a los servicios y recursos, y controlar el acceso a estos.

- El SO distribuido, utiliza la red para conectar varios procesadores, que tienen una copia de micro Kernel, a un núcleo central para permitir distribuir los procesos entre ellos. Son múltiples pequeños SO que para el usuario representan uno solo. Tienen mayor tolerancia a fallos y dan un servicio 24/7, ya que todos los servidores tienen el mismo núcleo y se puede modificar su uso.

- El SO en paralelo consiste en conectar vía bus varias CPU a un solo SO, es decir, multiprocesadores. Esto permite que se puedan ejecutar a la vez dos o más procesos que estén compitiendo por un recurso.

En la actualidad se avanza en la nanotecnología con SO portables y microcodificados, y en robótica

**Planificadores de múltiples procesadores**: Lo ideal es tener una cola de listos con todos los procesos y un procesador como planificador, que se ocupe de asignar los procesos a los demás procesadores y controlar los recursos de E/S.

**Planificación en tiempo real**: Los duros deben completar una tarea crítica en un lapso de tiempo garantizado. Esto requiere que el planificador sepa con exactitud cuánto tarda en ejecutarse cada función del SO. Los blandos son menos restrictivos, solo requieren que los procesos críticos tengan mayor prioridad que otros y poca latencia de despacho.

**Modelos de evaluación de algoritmos**:

- Determinista: Evaluación analítica sencilla y rápida, con resultados exactos. La desventaja es que requiere demasiados conocimientos.

- Colas: Asocia a cada recurso colas de procesos en espera y con las métricas de las ráfagas de CPU y E/S, y los tiempos en que los procesos llega al sistema, calcula el rendimiento promedio, el aprovechamiento y el tiempo de espera. Puede resultar difícil trabajar con las matemáticas de algoritmos.

- Simulaciones: Implica programar un modelo del sistema que muestra y compara las estadísticas de las actividades de los dispositivos, los procesos y el planificador. Es una forma excelente de comparar dos algoritmos con las mismas entradas, pero es muy costoso y requiere mucho tiempo de procesador y espacio en memoria.

- Implementaciones: La única forma exacta de evaluar es codificarlo, colocarlo en el SO y ver cómo funciona. Pero esto es costoso ya que hay que codificarlo y modificar al SO, crear estructuras de datos, y ver la reacción de los usuarios.

Para que los algoritmos de planificación tengan un buen desempeño deben permitir la menor cantidad de cambios de contexto, ser lo más justo posible con todo tipo de procesos, poder predecir cómo será la ejecución, permitir el máximo número de usuarios y capacidad de ejecución, tener un equilibrio en el uso de los recursos y proteger las prioridades.

**Algoritmos de concurrencia**

- Deshabilitar interrupciones: HW. Se le concede los privilegios de administración al programador del proceso. Consiste en apagar las interrupciones del SO y el procesador, que sirven para notificar eventos como la culminación del proceso. Si el desarrollador olvida habilitar este evento, el proceso en ejecución quedara como dueño del procesador y el SO no se enterara de su finalización.

- Variables de candado: Espera activa. Utiliza una variable candado para controlar el ingreso a las SC. Cuando la variable se encuentra en 1, el proceso quedara ciclando en nulo. Al comienzo de las SC se presenta una consulta sobre el valor de esta variable. Si está en 0, el proceso cambia su valor a 1 e ingresa a su SC. Si ocurre un cambio de contexto antes de que acabe su SC, el otro proceso no podrá ingresar a la suya. Si el cambio se da antes de que el proceso actualice su valor a 1, el otro proceso podrá ingresar a su SC, y se incumplirá la primera condición.

- Variable de turno: Espera activa. El primer proceso podrá ingresar a su SC cuando la variable esté en 0, y el otro proceso cuando esté en 1. Las condiciones y asignaciones se encuentran cruzadas y por lo tanto las ejecuciones serán alternadas. Se viola la condición 3: Supongamos que el proceso A termina su SC, cambia le valor de la variable a 1 y ejecuta su SNC. Si este proceso desea ingresar nuevamente a su SC antes de que el proceso B haya pasado por la suya, no podrá ingresar ya que la variable debe valer 0.

- Solución de Peterson: Espera activa. Es un solo algoritmo compartido para ambos procesos que utiliza las variables compartidas turno y candado, de las cuales una depende del proceso a ejecutar y la otra de su adversario. Cuando el proceso quiera ingresar a la SC deberá llamar a la función enter\_region(), y cuando salga invocara a la función leave\_region(). Por más que ambos procesos llamen al mismo tiempo a la función, la doble condición hace que esta solución sea válida.

- Dormir y despertar: Espera no activa. El acceso a la variable count es inestricto. En caso de que el buffer esté lleno y se produzca un cambio de contexto luego de que el productor verifique que la variable count es igual a N, se produciría una falla, ya que cuando el consumidor verifique que el buffer tiene espacio, va a despertar al productor que debería estar durmiendo, pero esta señal se pierde ya que no llego a ejecutarse este evento. Luego, el productor se adueña del procesador nuevamente y se va a dormir, ya que había verificado anteriormente que el buffer estaba lleno.

- Semáforos: Solución para dormir y despertar. Utiliza 3 semáforos: empty (cantidad de ranuras vacías), full (cantidad de ranuras llenas), y mutex (binaria para controlar el acceso). Controla estas variables con las funciones atómicas up() y down(). Empty y full pueden tomar valores desde 0 al más infinito. Si se quiere hacer un down de estas cuando están en cero, el proceso se bloqueara. Por lo tanto, el consumidor quedara bloqueado hasta que el productor realice un up(full), siendo full igual a 0, y el productor se bloqueara hasta que el consumidor haga un up(empty), valiendo 0 anteriormente.

- Monitores: Son contenedores de SC. El código del programador tendrá llamadas a las SC que se encuentran en el monitor, controladas con semáforos.

- Instrucción TSL: SW y HW. Se utiliza con multiprocesadores, para SO en paralelo y distribuido. Bloquea el acceso del procesador a un espacio de memoria.