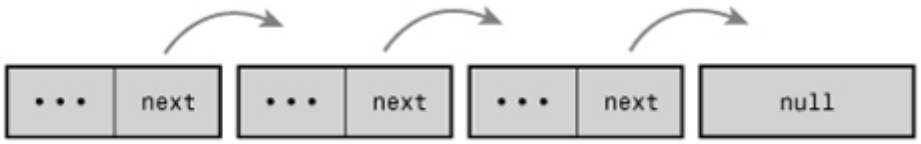
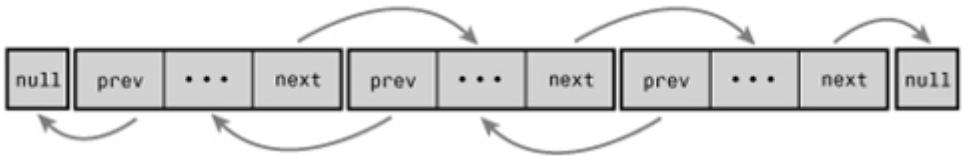
KERNEL DATA STRUCTURES

- **LINKED LIST**

Es una estructura de datos que permite almacenar y manipular un número variable de elementos, llamados nodos. Los elementos de una lista enlazada se crean e insertan dinámicamente en la lista. Esto permite gestionar un número variable de elementos desconocido en tiempo de compilación. Por lo tanto, los elementos deben estar enlazados entre sí; así, cada elemento de la lista contiene un puntero al siguiente elemento. A medida que se añaden o eliminan elementos de la lista, simplemente se ajusta el puntero al siguiente nodo.

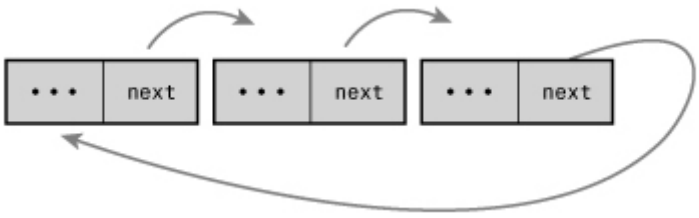
- **LISTA ENLAZADA Y LISTA DOBLEMENTE ENLAZADA**





- **LISTA ENLAZADA CIRCULAR**

Normalmente, dado que el último elemento de una lista enlazada no tiene un siguiente elemento, se establece para que apunte a un valor especial, como NULL, para indicar que es el último elemento de la lista.



- **MOVERSE POR UNA LISTA ENLAZADA**

El movimiento a través de una lista enlazada ocurre linealmente. Se visita un elemento, se sigue el siguiente puntero y se visita el siguiente elemento.

- **LIST HEADS**

Con simples cambios de código, nuestra estructura es ahora manejable por las rutinas de listas enlazadas del núcleo. Pero antes de que podamos usar esas rutinas, necesitamos un puntero canónico para referirnos a la lista como un todo, un puntero head.

- **RECORRER LISTA**

La forma más básica de iterar sobre una lista es con la macro list_for_each(). La macro toma dos parámetros, ambas estructuras list_head. El primero es un puntero utilizado para apuntar a la entrada actual. El segundo es el list_head que actúa como el nodo head de la lista que se quiere recorrer. En cada iteración del bucle, el primer parámetro apunta a la siguiente entrada de la lista, hasta que se haya visitado cada entrada.

- **COLAS**

El productor introduce los datos en la cola y el consumidor los saca de la cola.

- **KFIFO**

Proporciona dos operaciones principales: enqueue (in) y dequeue (out). El objeto kfifo mantiene dos desplazamientos en la cola: un desplazamiento de entrada y un desplazamiento de salida. El desplazamiento de entrada es la ubicación en la cola en la que se producirá el siguiente encolamiento . El desplazamiento de salida es la ubicación en la cola desde la que se producirá el siguiente desencole.

- **Encolar datos**

unsigned int kfifo_in(struct kfifo *fifo, const void *from, unsigned int len);

Esta función copia los bytes len empezando por desde en la cola representada por fifo. Si tiene éxito, devuelve el número de bytes en la cola. Si hay menos de len bytes libres en la cola, la función copia sólo hasta la cantidad de bytes disponibles.

- **Desencolar datos**

unsigned int kfifo_out(struct kfifo *fifo, void *to, unsigned int len);

Esta función copia como máximo len bytes de la cola apuntada por fifo al buffer apuntado por to. En caso de éxito, la función devuelve el número de bytes copiados. Si hay menos de len bytes en la cola, la función copia menos de lo solicitado.

- **MAPS**

Es una colección de keys únicas, donde cada clave está asociada a un valor específico. La relación entre una clave y su valor se denomina mapa

• Add (key, value)

• Remove (key)

• value = Lookup (key)

- **ÁRBOLES BINARIOS**

Árbol en el que los nodos tienen como máximo dos aristas de salida, es decir, un árbol en el que los nodos tienen cero, uno o dos hijos.

- **ÁRBOLES BINARIOS DE BÚSQUEDA**

Árbol con un ordenamiento específico impuesto a sus nodos. El ordenamiento suele definirse mediante la siguiente inducción:

- El subárbol izquierdo de la raíz contiene sólo nodos con valores menores que la raíz.

- El subárbol derecho de la raíz contiene sólo nodos con valores mayores que la raíz.

- Todos los subárboles son también árboles de búsqueda binarios.

Un árbol de búsqueda binario es, por lo tanto, un árbol binario en el que todos los nodos están ordenados de tal manera que los hijos de la izquierda son menores que su padre en valor y los hijos de la derecha son mayores que su padre.

- **ÁRBOLES NINARIOS DE BÚSQUEDA AUTOBALANCEADOS**

Un árbol de búsqueda binario equilibrado es un árbol de búsqueda binario en el que la profundidad de todas las hojas(nodos sin hijos) difiere como máximo en uno.

- **ÁRBOLES RED-BLACK**

Es un tipo de árbol de búsqueda binaria autoequilibrado. Los árboles rojo-negro tienen un atributo de color especial, que es rojo o negro y se mantienen semibalanceados al exigir que se cumplan las siguientes seis propiedades:

1. Todos los nodos son rojos o negros.

2. Los nodos de las hojas son negros.

3. Los nodos de las hojas no contienen datos.

4. Todos los nodos que no son hojas tienen dos hijos.

5. Si un nodo es rojo, sus dos hijos son negros.

6. El camino de un nodo a una de sus hojas contiene el mismo número de nodos negros que el camino más corto a cualquiera de sus otras hojas.

En conjunto, estas propiedades garantizan que la hoja más profunda no tenga más del doble de profundidad que la hoja más superficial.

Si las operaciones de inserción y eliminación cumplen estas seis propiedades, el árbol sigue estando semibalanceado.