1. Preguntas y Respuestas

1.1. El kernel

1. **¿Que es el ejecución directa?**
   Ejectar un programa directamente en la CPU.
   Beneficios: rapidez, Problemas: seguridad, confiabilidad, portabilidad.

   Limitar la ejecución directa:

   - **Modo de operación dual:** es un mecanismo que proveen todas los procesadores para poder intercambiar entre user mode y kernel mode.
   - **Instrucciones Privilegiadas:** set de instrucciones que poseen cada modo de operación.
   - **Proteccion de Memoria:** como la memoria es compartida, el SO debe poder configurar el hardware de forma tal que cada proceso pueda leer y escribir su propia porción de memoria.
   - **Interrupciones por temporizador:** mecanismo le permita al kernel desalojar al proceso y volver a tomar el control del procesador.

2.

1.2. El proceso

1. **Describa que es un proceso: qué abstrae, cómo lo hace, cuál es su estructura. Además explique el mecanismo por el cual el proceso cree tener la memoria completa de la máquina cuando en realidad solo tiene lo necesario para su funcionamiento.**
   Es un proceso es sólo un programa en ejecución. Un proceso incluye:

   - Los Archivos abiertos
   - Las señales(signals) pendientes
   - Datos internos del kernel
   - El estado completo del procesador
   - Un espacio de direcciones de memoria
   - Uno o más hilos de ejecución. Cada thread contiene
     - Un contador de programa
     - Un Stack
     - Un Conjunto de Registros
     - Una sección de datos globales

   Abstrae los recursos del sistema como la CPU, la memoria y los dispositivos de entrada/salida. La abstracción del proceso provee ejecución, aislamiento y protección.

El mecanismo es la virtualización de memoria, que es una abstracción por la cual la memoria física puede ser compartida por diversos procesos.

2. **Cuál/cuáles mecanismos utiliza el kernel para garantizar el aislamiento entre procesos. Estos mecanismos están relacionados con el hardware, porque deben existir y donde se ve su funcionamiento.**
   El kernel utiliza la virtualización de memoria para garantizar el aislamiento entre procesos.

3. **¿Que es la virtuaizacion?**
   Es crear una abstracción que haga que un dispositivo de hardware sea mucho más fácil de utilizar. Existen dos tipos de virtualización:

   - **Virtualización de memoria:** Le hace creer al proceso que este tiene toda la memoria disponible.
     - Protección de Memoria: Memoria Virtual.
     - Traducción de Direcciones.
   - **Virtualizacion de procesador:** Consiste en dar la ilusión de la existencia de un único procesador para cualquier programa que requiera de su uso.

     De esta forma, se prove:
     - Simplicidad en la programación.
     - Aislamiento frente a Fallas.

4. **¿Cuales son los mecanismos de protección de memoria?**
   La memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

   Un componente clave de la memoria virtual son las direcciones virtuales, con las direcciones virtuales, para cada proceso su memoria inicia en el mismo lugar, la dirección 0.

   El hardware traduce la dirección virtual a una dirección física de memoria, se realiza por hardware (MMU).

5. **¿Que es el address space?¿Que partes tiene?¿Para qué sirve?. Describa el/los mecanismos para crear un proceso en unix, sus sycalls, ejemplifique.**
   El address space es el espacio de direcciones virtuales que un proceso puede utilizar. Está dividido en varias áreas: text, data, stack y heap. El propósito del address space es mantener separados los procesos y evitar que un proceso escriba en los datos de otro proceso.

   Para la creacion de un proceso:

   - única forma es llamando a la system call *fork*.

6. **¿Que es el stack ? Explique el mecanismo de funcionamiento del stack para x86 de la siguiente funcion int read(void *buff, size_t num, int fd);.Como se pasan los parametros, direccion de retorno?.**

   El stack o pila es una estructura de datos que almacena información de forma temporal y ordenada, siguiendo el principio LIFO (Last In, First Out), es decir, el último en entrar es el primero en salir. El stack se usa para guardar los datos locales de una función, las direcciones de retorno de las llamadas a funciones y los parámetros que se pasan a las funciones.

   Para la función read(void *buff, size_t num, int fd), que lee num bytes del archivo identificado por fd y los almacena en el buffer buff, se puede usar el stack para pasar los parámetros de la siguiente manera:

   - Se empujan los parámetros al stack en orden inverso, es decir, primero fd, luego num y finalmente buff.
   - Se llama a la función read con la instrucción call, que empuja la dirección de retorno al stack y salta a la etiqueta de la función.
   - Dentro de la función read, se accede a los parámetros usando el registro ebp (base pointer) como referencia. El registro ebp se usa para guardar el valor del registro esp (stack pointer) al entrar en la función, y así poder acceder a los parámetros y variables locales sin importar cómo cambie el esp durante la ejecución de la función
   - Se usa la convención cdecl para limpiar el stack después de la llamada a la función. Esta convención establece que el código que llama a la función es responsable de restaurar el esp al valor que tenía antes de empujar los parámetros. Esto se hace sumando al esp el tamaño total de los parámetros.

   Un posible código en ensamblador x86 para este ejemplo sería:

   ; Código que llama a la función read ; Supongamos que fd = 3 (stdin), num = 100 y buff apunta a una zona de memoria reservada mov eax, 3 ; fd push eax ; empujar fd al stack mov eax, 100 ; num push eax ; empujar num al stack mov eax, buff ; buff push eax ; empujar buff al stack call read ; llamar a la función read add esp, 12 ; limpiar el stack (3 parámetros de 4 bytes cada uno)

   ; Código de la función read read: push ebp ; guardar el valor anterior de ebp mov ebp, esp ; copiar el valor de esp a ebp ; Ahora los parámetros se pueden acceder como [ebp+8], [ebp+12] y [ebp+16] ; Aquí iría el código para leer del archivo y escribir en el buffer ; usando las instrucciones syscall o int 80h mov esp, ebp ; restaurar el valor de esp pop ebp ; restaurar el valor de ebp ret ; retornar a la dirección guardada en el stack

7.

1.3. La Memoria

1. **¿Que es la memoria virtual? ¿Qué mecanismos conoce, describa los tres que a usted le parezcan más relevantes?**
   La Memoria Virtual es un mecanismo de protección de memoria, provisto por el Hardware. La memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

   - **La memoria segmentada** es una técnica de gestión de memoria que divide el espacio de memoria de un proceso en segmentos lógicos más pequeños y coherentes, en lugar de tratarlo como un espacio de memoria continuo y uniforme. Cada segmento representa una porción lógica de la memoria y puede contener diferentes tipos de datos, como código, datos, pila, tabla de símbolos, etc.

     Cada segmento tiene un tamaño (Bound o registro límite o Segmento) y una dirección base (registro base) asociada. La dirección base indica la ubicación física donde comienza el segmento en la memoria física, mientras que el tamaño representa la longitud del segmento. En lugar de utilizar direcciones absolutas, se utilizan direcciones relativas dentro de cada segmento.

     La memoria segmentada ofrece varias ventajas. Permite una mayor flexibilidad en la asignación y el uso de memoria, ya que los segmentos pueden crecer o contraerse dinámicamente según las necesidades del proceso. También facilita el compartimiento de memoria entre diferentes procesos, ya que es posible compartir segmentos comunes entre ellos, lo que puede ahorrar espacio y mejorar la eficiencia.

     Sin embargo, la segmentación también puede presentar desafíos, como la fragmentación externa, que ocurre cuando hay espacios vacíos entre segmentos que no se pueden utilizar para almacenar otros segmentos. Esto puede llevar a un desperdicio de memoria. Además, la gestión de los segmentos y la traducción de direcciones pueden requerir una mayor complejidad en el hardware y el sistema operativo.

     En resumen, la memoria segmentada es una técnica de gestión de memoria que divide el espacio de memoria de un proceso en segmentos lógicos, lo que proporciona flexibilidad y compartición de memoria, pero puede implicar desafíos como la fragmentación externa.
   - **La memoria paginada** es una técnica de gestión de memoria en la que la memoria se divide en fragmentos de tamaño fijo llamados "page frames". En lugar de dividir la memoria en segmentos lógicos, como en la memoria segmentada, la memoria paginada la divide en páginas de tamaño uniforme. Cada página tiene un número de página virtual y una dirección física correspondiente.

     El mecanismo de traducción de direcciones en la memoria paginada es similar al de la memoria segmentada. Cada proceso tiene una tabla de páginas (page table) que contiene entradas que mapean las páginas virtuales a las direcciones físicas de los page frames en la memoria

física. Cuando un proceso accede a una dirección virtual, se utiliza la tabla de páginas para obtener la dirección física correspondiente.

La dirección virtual consta de dos componentes: el número de página virtual y el desplazamiento (*offset*) dentro de esa página. El número de página virtual se utiliza como índice en la tabla de páginas para obtener la dirección física del page frame correspondiente. Luego, se concatena el desplazamiento para obtener la dirección física completa.

La memoria paginada ofrece varias ventajas, como una mayor eficiencia en la gestión de la memoria y la capacidad de compartir páginas entre procesos, lo que permite la memoria compartida. También facilita la protección de la memoria, ya que cada página se puede asignar permisos individuales de lectura, escritura y ejecución.

Un aspecto importante de la memoria paginada es que proporciona una vista lógica de la memoria lineal para cada proceso, aunque las páginas pueden estar dispersas por toda la memoria física. Esto significa que las direcciones virtuales son continuas y lineales para el proceso, aunque las páginas físicas pueden estar ubicadas en diferentes ubicaciones físicas.

En sistemas de paginación multinivel, como el utilizado en la arquitectura x86, se pueden utilizar múltiples niveles de tablas de páginas para gestionar direcciones virtuales más grandes de manera eficiente. Esto permite una mayor flexibilidad y eficiencia en la gestión de la memoria.

En resumen, la memoria paginada es una técnica de gestión de memoria en la que la memoria se divide en páginas de tamaño fijo y se utiliza una tabla de páginas para traducir direcciones virtuales a direcciones físicas. Proporciona una vista lógica de la memoria lineal para cada proceso y ofrece ventajas como una gestión eficiente de la memoria y la capacidad de compartir páginas entre procesos.

- **Paged Segmentation (Segmentación paginada)** es una combinación de la segmentación y la paginación. Consiste en dividir el espacio de direcciones lógicas en segmentos de tamaño variable, y luego dividir cada segmento en páginas de tamaño fijo. Cada segmento tiene una tabla de páginas asociada, que se almacena en una tabla de segmentos. El proceso de traducción de las direcciones lógicas a físicas es, primero se busca el segmento en la tabla de segmentos, luego se busca la página en la tabla de páginas del segmento, y finalmente concatena el frame de la oage table con el offset para obtener la dirección física completa.
  - Reduce la fragmentación externa.
  - Mejora el rendimiento.
  - Proporciona un buen equilibrio entre flexibilidad y rendimiento

2.

1.4. Concurrencia

1. **¿Que es un thread?. Use su API para crear un programa que use 5 thread para incrementar una variable compartida por todos en 7 unidades/thread hasta llegar a 100**
   Un thread es una secuencia de ejecución atómica que representa una tarea planificable de ejecución. También una secuencia independiente de instrucciones ejecutándose dentro de un programa.

```
1
2
3
4
5
6
```

```c
#include <stdio.h>

int main() {
printf("hello, world\n");
}

```

Script 1: hola mundo.

2.

1.5. Definiciones sueltas

El sistema operativo tiene que poder configurar el hardware de forma tal que cada proceso pueda leer y escribir solo su propia memoria.