1. ¿Qué diferencias hay entre la implementación de threads a nivel de Kernel y como biblioteca de usuario? ¿Qué ocurre en ambos casos con una llamada que debería bloquear al proceso como una lectura?

Implementación como biblioteca de usuario:

- Puede implementarse en sistemas operativos que no soporten threads.
- Cada proceso necesita su tabla de threads privada para poder monitorear a los threads de ese proceso.
- Se mantiene información acerca de cada thread individual: program counter, stack pointer, registros, estados y demás.
- Se puede hacer un switching de un thread a otro sin la necesidad de hacer un trap a nivel kernel.
- Cada proceso puede tener su propio algoritmo de scheduling para los threads.
- Si un thread comienza, ningún otro thread va a correr en CPU hasta que éste ceda voluntariamente la misma.

Tiene ciertos problemas cuando ocurre una system call bloqueante. Sería inaceptable dejar que el thread realice la system call, ya que bloquearía el proceso, afectando a otros threads. Una alternativa sería cambiar las system calls para que sean no bloqueantes, pero eso implicaría un cambio en el sistema operativo. Otra alternativa más viable sería realizar un wrapper para las llamadas bloqueantes, que sólo se realizarían cuando éstas son seguras.

Implementación a nivel kernel:

- No hay tabla de threads en cada proceso.
- El kernel tiene una tabla de threads que mantiene información de todos los threads del sistema.
- El kernel le puede sacar tiempo de CPU para otorgárselo a otro thread.
- Debido al alto costo de crear y destruir threads, éstos se reciclan.
- Cuando un thread causa una page fault, el kernel puede chequear si el proceso tiene otro thread y correr ese mientras espera que la página requerida sea traída desde el disco.
- Las señales son enviadas a los procesos, no a los threads.

Cuando un thread bloquea, el kernel puede correr otro thread, tanto del mismo proceso como de otro.

2. ¿Cumple el Intel Pentium con las condiciones de Popeck y Goldberg para virtualizar una plataforma? ¿Cómo se realiza esta virtualización?

Condicion de Popeck y Goldberg -> Se puede construir VM si las Instrucciones sensibles (las qe dependen de los recursos o los configuran) son un subconjunto de las Privilegiadas (las qe pueden generar traps si esta en modo usuario).

No. Cuenta con instrucciones sensibles que no son privilegiadas, violando una de las condiciones de Popeck.

Para realizar la virtualización de esta máquina se requiere de una extensión que la permita, que en este caso es el Intel IVT, que genera containers en los que la ejecución de una instrucción sensible provoca un software trap. Luego, con la trampa se debe emular el comportamiento de la máquina autónoma del sistema operativo guest.

3. En Android, ¿qué es una Activity y cómo es su ciclo de vida?

Una activity es una aplicación que se comunica por medio de una pantalla con el usuario. Generalmente, esta comunicación se realiza fullscreen, pero puede ser una pantalla flotante. Una aplicación está compuesta por una o más activities, pero sólo una puede estar activa a la vez. El resto de las activities se guarda en un stack.

Ciclo de vida:

- Active: está al tope del stack e interactuando con el usuario.
- Paused: visible pero sin foco.
- Stopped: queda en memoria pero ya terminó. Candidata al kill.
- Inactive: fuera de la memoria. Debe lanzarse nuevamente.

4. ¿Cuál es el principio de funcionamiento de los hipervisores de tipo II?

Corre bajo el control de un sistema operativo anfitrión. Puede virtualizar cualquier ambiente y modifica el programa que está corriendo.

5. ¿Qué es y que cosas debe contener un Object File tanto para un ejecutable como para una Biblioteca (por ejemplo, los object file formats pe, elf o coff)?

Es un formato de archivos de computadora usado para el almacenamiento de código objeto y datos relacionados.
Hay muchos formatos distintos. Originalmente, cada tipo de computadora tenía su propio formato único, pero con la venida de Unix y otros sistemas operativos portables, algunos formatos, como COFF y ELF fueron definidos y usados en distintos tipos de sistemas. Es posible usar el mismo formato como entrada y salida del linker y también como librería y formato del archivo ejecutable.

Un object file contiene 5 tipos de información:

- Header information: información general acerca del archivo, tales como el tamaño del código, nombre del archivo fuente del cual fue traducido y fecha de creación.
- Código objeto: instrucciones binarias y data generada por un compilador o ensamblador.
- Relocation: una lista de los lugares en el código objeto que tienen que ser modificador cuando el linker cambia la dirección del código objeto.
- Symbols: símbolos globales definidos en el módulo, símbolos a ser importador de otros módulos o definidos por el linker.
- Debugging information: otra información sobre el código objeto, no necesitada para el linkeo pero sí para el uso de un debugger. Esto incluye archivos fuentes e información sobre números de línea, símbolos locales, descripción de estructuras de datos usadas por el código objeto, tales como definiciones de estructuras de C.

No todos los formatos contienen todos los tipos mencionados y es posible tener formatos útiles con poca o ninguna información más allá del código objeto.

6. En pseudocódigo muy simple, indique cómo hace para llamar a una función de biblioteca funcl() de la que no conoce el nombre de la biblioteca en que se encuentra hasta el momento de ejecución.

- Se abre la biblioteca, cuyo nombre es conocido al momento de ejecución, mediante dlopen().

- Se obtiene la función funcl() mediante el manejador de la biblioteca y la función dlsym().
- Se realiza la llamada a la función.

7. ¿Qué diferencias hay entre paravirtualización y traducción binaria?

Paravirtualizar consiste en reemplazar, en el sistema operativo guest, las instrucciones delicadas por llamadas al hipervisor. En este caso, el sistema operativo huésped debe ser modoficado para saber que va a correr en un entorno virtualizado. El huésped utiliza una API especial para comunicarse con la capa de virtualización e interactuar directamente con el hard. Si bien tenemos claros beneficios por el lado de la performance, se pierde compatibilidad ya que el SO huésped debe ser modificado. Requiere código fuente.

Traducción binaria, en cambio, consiste en examinar bloques básicos (código con un punto de entrada, uno de salida y sin jumps), tarea realizada por el hipervisor, y reemplazar las instrucciones delicadas por llamadas al hipervisor. El código traducido lo guarda en caché para aumentar la performance. Esto genera más sobrecarga que el enfoque de paravirtualización, pero no limita a usar sólo SO modificados, podemos usar cualquier SO. No requiere código fuente.

8. ¿Cuál es la arquitectura general de Android?

La arquitectura del sistema operativo está dividida en 4 capas:

- Aplicaciones.
- Frameworks.
- Bibliotecas y Android Run Time.
- Kernel Linux.

9. ¿Qué es RAID, stripping y mirroring? Describa las diferencias entre RAID propiamente dicho y Software RAID o Fake RAID.

RAID es una tecnología de almacenamiento que combina múltiples discos en una sola unidad lógica. Los datos son distribuidos a través de los discos en una de las varias maneras, denominadas “niveles RAID”, dependiendo de la redundancia y performance requerida.

Data stripping es la técnica de segmentar datos secuencialmente lógicos, como un archivo, de manera que segmentos consecutivos son almacenados en dispositivos de almacenamiento distintos. Este mecanismo es útil cuando un proceso que accede a un dispositivo necesita acceso a datos de manera más veloz de la que este dispositivo puede proveer. Permite acceder a los datos de manera concurrente.

Disk mirroring es la replicación de volúmenes de discos en discos duros separados para asegurar la disponibilidad continua. La replicación se realiza a través de microcódigo en el controlador del disco o mediante software de servidor. Sirve como copia de seguridad de los datos ante alguna falla de hardware.

En los Fake RAID, el procesador debe usar su tiempo para las operaciones RAID, que se realizan en una capa entre el File System y el Device Driver. Este controlador toma las funciones de RAID durante el boot. Una vez que el kernel de un sistema operativo está cargado, el control pasa al sistema operativo. Esto se debe a que Windows no puede bootear desde software RAID.

En los RAID originales, se requiere un controlador dedicado por parte de los discos. Este controlador debe tener un back end hacia los discos y un front end hacia el host.

10. Explique la diferencia entre un thread y un proceso en un sistema con threads nativas a nivel de kernel.

Los procesos agrupan los recursos usados. Cada proceso se ejecuta en un espacio de memoria separado.

Los threads son hilos de ejecución que comparten el agrupamiento de recursos, entre ellos, el espacio de memoria. Cada thread mantiene su propia información de estado.

Los threads son subconjuntos de los procesos.

Ejemplo de código:

Procesos

```
int variable_compartida;

void func() {
	extern int variable_compartida;
	for (int i = 0; i < 10; i++) {
		variable_compartida++;
	}
}


int main() {
	extern int variable_compartida;
	pid_t p1, p2;
	int status;

	variable_compartida = 0;

	if ((p1 = fork()) == 0) { func(); exit(0); }
	if ((p2 = fork()) == 0) { func(); exit(0); }

	for(int i = 0; i < 10; i++) {
		variable_compartida++;
	}

	waitpid(p1,&status,0);
	waitpid(p2,&status,0);

	std::cout << "variable_compartida vale " << variable_compartida << std::endl;
```

```
}
```

Salida:
variable_compartida vale 10.

Threads

```
#include <pthread.h>
#include <iostream>
#include <unistd.h>
#include <stdlib.h>

int variable_compartida;

void *func(void *arg) {
	extern int variable_compartida;
	for (int i = 0; i < 10; i++) {
		variable_compartida++;
	}

	return NULL;
}

int main() {
	pthread_t th1,th2;
	extern int variable_compartida;

	variable_compartida = 0;

	pthread_create(&th1,NULL,func,NULL);
	pthread_create(&th2,NULL,func,NULL);

	for (int i = 0; i < 10; i++) {
		variable_compartida++;
	}

	pthread_join(th1,NULL);
	pthread_join(th2,NULL);

	std::cout << "variable_compartida vale " << variable_compartida << std::endl;
}
```

Salida:
variable_compartida vale 30.

11. Explique con un ejemplo las estructuras de las tablas de páginas directa, invertida y

multinivel.

Tabla de páginas directa: se utiliza una tabla hash, donde el valor del hash es el número de la página virtual. Cada entrada en la tabla hash contiene una lista enlazada de elemento que dirigen a la misma posición.

Tabla de página invertidas: combina una tabla de frames y una de páginas en una sola estructura. Una entrada por cada frame. La búsqueda es asociativa por contenido.
Tabla de página multinivel: se tiene en memoria las tablas que están siendo utilizadas y se deja en disco aquellas que no están siendo referenciadas.

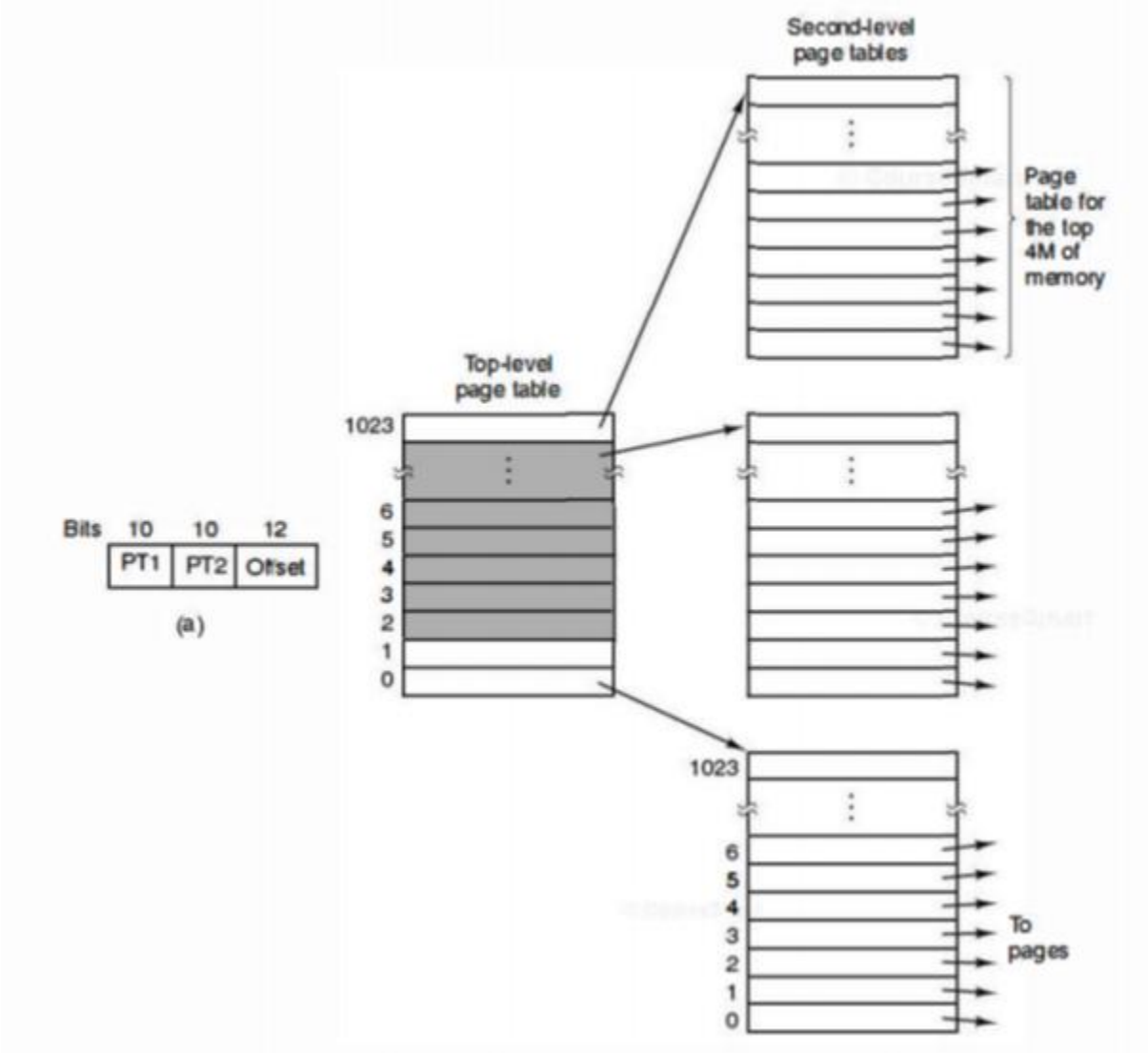


tabla de página multinivel

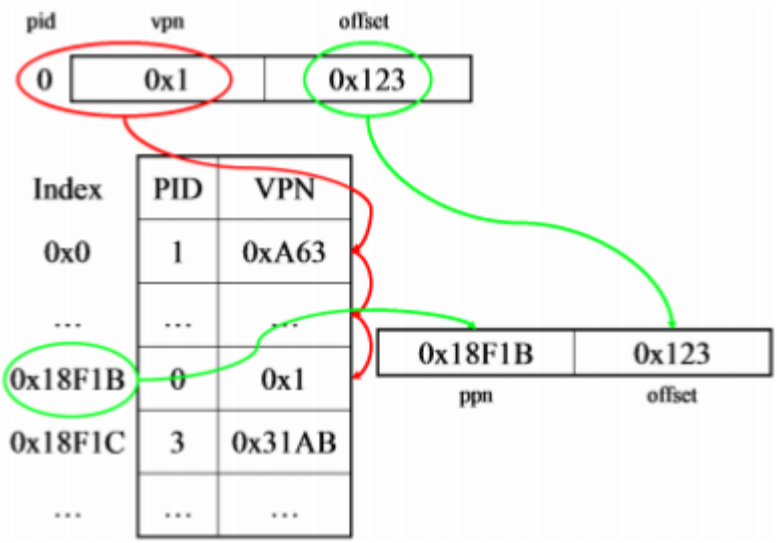


tabla de página invertida

12. ¿Qué es la link-edición?

Es la mezcla de las direcciones de cada módulo utilizado en una aplicación, en un único espacio de direcciones, produciendo como salida una biblioteca, un programa objeto o un programa ejecutable.

13. ¿Qué diferencias hay entre paravirtualización y los hipervisores tipo I y II?

La fiabilidad de la Paravirtualización puede verse obstaculizada debido a la complejidad que implica la ejecución de un kernel del sistema operativo modificado para realizar las interacciones con el Hypervisor. En muchos casos, el proveedor del sistema operativo, que pueden ser reacios a desarrollar más productos, debe ser compatible con esta modificación del kernel.

La fiabilidad de los hipervisores es muy alta. Debido a la falta de modificación del sistema operativo invitado, hay menos complejidad que equivale a una mayor fiabilidad y una mayor aceptación por parte de los proveedores del sistema operativo.

Las opciones del sistema operativo paravirtualizado son limitadas y, por tanto, hay menos opciones para proporcionar una solución. La gama de opciones disponibles puede limitar la utilidad efectiva. Por otro lado, la paravirtualización imparte aumentos de rendimiento sobre la virtualización de hardware completa.

En comparación con los hipervisores, la amplia gama de sistemas operativos guest utilizables y la falta de personalización del sistema operativo guest, la virtualización completa de hardware proporciona un mayor rango de utilidad.

En última instancia, ni la paravirtualización ni la virtualización completa de hardware exponen problemas de fiabilidad o utilidad en el otro. Cada uno tiene sus ventajas y desventajas.

14. ¿Qué son y cómo se acceden los archivos mapeados a memoria? ¿Tienen alguna ventaja respecto de los archivos tradicionales?

Los archivos mapeados a memoria se ven como parte de la memoria, se manejan junto con la memoria virtual y permiten compartir archivos.

Tienen 3 propósitos los cuales son:
1. Cargar y ejecutar archivos .exe y bibliotecas de enlace dinámico (DLL).
2. Acceder a archivos en disco rápidamente sin necesidad de usar buffers.
3. Permitir a varios procesos compartir la misma información en memoria.

15. ¿Qué diferencia hay entre la system call fork() y las exec() (de cualquier tipo)?

- **Responda con base en un ejemplo de código.**
- **Indique en su respuesta el contenido antes y después de la ejecución de las áreas PCB, TXT y U_AREA.**

La función exec() reemplaza la imagen del padre copiada en el hijo por la propia área static, TXT y STACK. Esta función se encarga de ejecutar el comando que corresponde al proceso creado.

Realiza la siguiente serie de pasos:
- Verifica las autorizaciones.
- Lee el encabezado para obtener los tamaños del segmento y el total.
- Captura los argumentos y el medio del solicitante.
- Libera la memoria anterior y asigna la nueva.
- Copia la pila en la nueva imagen de memoria.
- Copia los segmentos del texto y datos en la nueva imagen de memoria.
- Indica al kernel que ahora el proceso es ejecutable.

La función fork() crea un nuevo proceso (proceso hijo), y copia su U_AREA, TXT, stack, y BSS en el nuevo proceso creado. A su vez, el nuevo proceso cuenta con un PCB, con la pid con la que fue creado. La system call fork() devuelve 0 al proceso hijo creado (que puede conocer su pid mediante getpid()) y devuelve el pid del proceso hijo al proceso padre que lo creó.

Esta system call realiza los siguientes pasos:
- Verificar si la tabla de procesos está repleta.
- Intentar asignar memoria al proceso hijo.
- Copiar la imagen del proceso padre al proceso hijo.
- Meter el mapa de memoria del derivado en la tabla de procesos.
- Elegir un pid para el proceso hijo.
- Indicar al kernel y el sistema de archivos acerca del proceso hijo.
- Informar del mapa de la memoria del proceso hijo al kernel.

Código:

```
if ((pidhijo = fork ()) == 0) {
	std::cout << std::endl << "-->Es el HIJO con pid = " << getpid() << std::endl;
	exit(0);
}else {
	std::cout << "PADRE con pid = " << getpid() << " hijo con pid = " << pidhijo << std::endl;
	exit(0);
}
```

Respuesta alternativa:

La system call fork es la única manera de crear un proceso nuevo en POSIX. Crea un duplicado del proceso original, incluyendo todos los file descriptors y registros. La llamada a fork devuelve un valor, que es 0 en el hijo y el PID del nuevo proceso en el padre. La llamada exec reemplaza la imagen por el archivo pasado en el primer parámetro, es decir, se ejecuta el archivo especificado con los parámetros pasados.

Por ejemplo:

```
read_command (command, parameters);
if (fork() != 0) {
	/* Parent code */
	waitpid (-1, &status, 0);
}else {
	/* Child code */
	execve(command, parameters, 0) ;
}
```

Estado del PCB (Process control block): En el padre, al realizarse el fork el proceso se bloquea, esperando a que termine el proceso hijo. En el hijo, por el otro lado, el proceso esta corriendo. Por otro lado, los process id del padre y del hijo, son distintos.
Estado de U AREA (user area): el user area, en estos casos es distinto ya que, el programa que realiza el fork es distinto a aquel llamado mediante execve. Al ser procesos distintos, el puntero a la tabla de procesos en la U AREA es distinto.

16. En Android, ¿cómo se separan los recursos de cada aplicación, cómo se evita que un error afecte al resto?

Cada aplicación corre en su propio proceso con su propia copia de Dalvik. Los procesos son provistos por el kernel y manejados por el Android Run Time. Para mantener la respuesta del sistema, el sistema Android puede matar sin aviso a procesos que considera que no están respondiendo (y las aplicaciones contenidas dentro del mismo).

17. ¿Cuándo un proceso está en estado Zombie? Describa una forma de lograrlo.

Un proceso está en estado zombie cuando el mismo ha finalizado, se ha liberado la memoria y los recursos utilizados, pero su entrada en la tabla de procesos permanece. Suele ser un breve período. Los procesos zombies que perduran denotan un error por parte del proceso padre al realizar una llamada al system call wait.

Los procesos zombies nunca recibieron la señal de finalización por parte del proceso padre que lo creo. Se pueden deber a errores de programación, a situaciones no contempladas por el programador y generalmente provocan lentitud y/o inestabilidad en el sistema.

```
int main () {
	pid_t child_pid;

	child_pid = fork();
	if (child_pid > 0) {
		sleep (60);
	}else {
		exit (0);
	}

	return 0;
}
```

18. Describa la diferencia entre Storage Area Network y Network Attached Storage (SAN y NAS). De ejemplo de los protocolos utilizados en cada caso.

NAS conecta un file system remoto a una red, proveyendo el acceso a clientes heterogéneos.
Provee tanto almacenamiento como un file system. Aparece ante un sistema operativo cliente como un servidor. Los servicios que provee son orientados al almacenamiento y no están diseñados para funcionar como un servidor multipropósito.

Provee servicios basados en archivos. Generalmente es una versión reducida empotrada de algún Sistema Operativo (Nexenta, FreeNAS, etc.). Usa protocolos como SMB/CIFS, NFS o AFP.
SAN es una red dedicada que provee acceso a datos por bloques. Conecta dispositivos remotos que el SO ve como locales, e implementa el file system. No provee la abstracción de archivos, sólo operaciones en bloques. De esta manera, el SO ve al SAN como un disco que puede ser formateado con un file system y montado al sistema operativo.

Consolida las “islas de discos” con conexiones de red. Pueden ser discos, RAIDs o alguna arquitectura no RAID. Usan protocolos como ISCSI, HyperSCSI, ATA_over_Ethernet o InfiniBand. Requieren de un software de administración. Algunos proveen capacidades RAID.

20. Ud. debe preparar unos programas para entregar en otra máquina, cuya versión del SO desconoce. Tampoco conoce las versiones de las bibliotecas instaladas en el sistema en que estos programas deben correr. Se tiene la posibilidad de compilar programas con linkeo

- **estático.**
- **dinámico en tiempo de carga.**
- **dinámico en tiempo de ejecución (donde la biblioteca se elige externamente).**

Indique qué posibilidad usaría y por qué para:
a) Bibliotecas estándar del sistema.
b) Bibliotecas desarrolladas por la organización cliente.

a) Para poder tener acceso a las mismas desde la computadora del cliente, sin previo conocimiento del sistema operativo ni sus bibliotecas, es necesario realizar un linkeo estático que copie el contenido de las subrutinas necesarias en el código a ejecutar.
b) En este caso, el linkeo estático no es una opción ya que no se cuenta, al momento de la codificación, con las bibliotecas a utilizar. Entre las otras dos se opta por el linkeo en tiempo de ejecución que permite manejar error referidos a la ausencia de una biblioteca del cliente.

21. ¿Qué papel cumple el lpc (local procedure call) en windows?

Interface “no documentada” para comunicarse con los Environment Subsystems y los System Support Processes.

- Es una cola de mensajes. Tiene un mecanismo especial para mensajes cortos.
- La usan las subsystem dll para implementar sus funciones.
- En Vista se extiende a ALPC (Advanced LPC).

22. ¿Como se realiza un write en un Log File System?

Las escrituras se hacen siempre en la cabeza de un log circular como transacciones.

- Se crean entonces versiones del mismo archivo.
- Si éstas son accesibles, es un Versioning File System.
- Si se reconstruye el archivo, se lo llama Journaling File System
- Ext3 o Ext4 son Journaling File System.
- JFFS se usa en las memorias flash para compensar su corta vida útil.

23. En un ambiente Windows: ¿Que entiende por servicio y por LPC (Local Procedure Call)?

Los servicios son creados y administrados por el SCM (Service Control Manager).

- Parecidos a los daemons de UNIX.
- En general no interactúan con el Desktop ni con el usuario.
- Los servicios pueden tener tres nombres: registry, service tool, process name.

Los servicios son programas o aplicaciones cargadas por el propio sistema operativo. Estas aplicaciones tienen la particularidad que se encuentran corriendo en segundo plano (Background). Por defecto, con la

instalación, se instalan y ejecutan una cierta cantidad de servicios. Dependiendo de nuestras necesidades, podemos necesitarlos a todos o no.

El local procedure call facility es un servicio provisto por Windows NT para alivianar la cola de mensajes entre procesos en una misma computadora.

24. Explique las diferencia entre bibliotecas
a) estáticas
b) dinámicas con carga en tiempo de carga.
Para cada caso indique cómo se resuelve la búsqueda de las rutinas en Linux y Windows.

Estáticas:

- -Cuando los subprogramas se incluyen en el ejecutable.

El link-editor es quien las incorpora al archivo ejecutable.
-En Linux lo hace el GNU ld, invocado al usar --static.
-En Windows, terceras partes hicieron un linker para .net. (Para evitar instalar el .net framework).

-Dinámicas:
Pueden cargarse:

- -antes de cargar el programa.
- -en el momento de cargar el programa.
- -durante la corrida del programa.

En Linux hay un “search path” formado por: posición fija indicada en el programa + archivo de configuración + variable de ambiente. -El orden de búsqueda es inverso (última modificación manda).
-
En Windows, ActiveX busca en el registry. El resto en el indicado en una llamada a SetDllDirectory() , System32, System y Windows.

25. Ejemplifique en un pseudocódigo parecido a C como se usan las bibliotecas dinámica con enlace en tiempo de ejecución.

En tiempo de ejecución el programa "decide" que biblioteca cargar y debe ubicar la biblioteca con dlopen(), y el procedimiento dentro de la biblioteca con dlsym().

Si proc es compartido, en la versión de carga dinámica con resolución en link-edición se usa la copia que esta en la memoria si hay alguna.

Pseudocódigo:

```
main
...
dlopen(lib_name,RTLD_LAZY);
...
lib_func=dlsym(lib_handle, "util_uno");
...
(*lib_func)();
```

...

26. ¿Cómo se acceden los archivos mapeados a memoria? Ejemplifique con un pseudocódigo parecido a C.

Lectura:

```
int main(int argc, char* argv[]) {
	int fd = open(argv[1], O_RDONLY,S_IRUSR );
	if (fd == -1) {
		perror ("Al abrir el archivo ");
		exit(2);
	}

	int* ar;
	void* addr = mmap(NULL,1024,PROT_READ,MAP_SHARED,fd,0);
	if (addr == MAP_FAILED) {
		perror("mmap");
		exit(1);
	}

	close(fd);
	ar = (int*) addr;

	for (int i = 0; i < 100; i++) {
		std::cout << "a[" << i << "]=" << ar[i] << ", ";
		if (i % 10 == 9) {
			std::cout<<endl;
		}
	}

	munmap(addr,len);
}
```

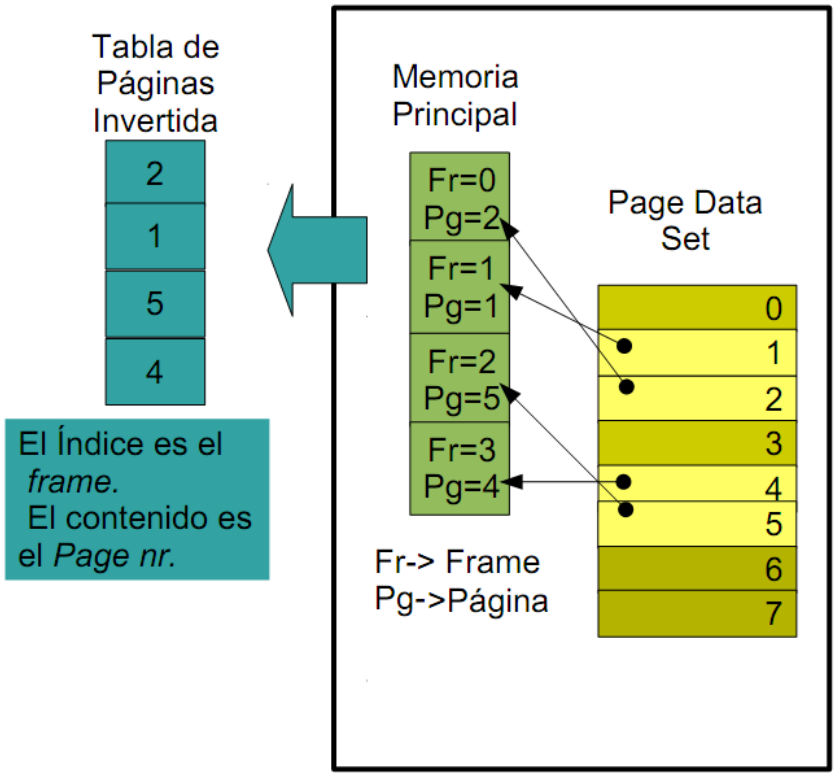
Escritura:

```
int main(int argc, char* argv[]) {
	int fd = open(argv[1],O_RDWR|O_CREAT,S_IRUSR | S_IWUSR);
	if (fd == -1) {
		perror ("Al abrir el archivo ");
		exit(2);
	}

	lseek (fd, 1024, SEEK_SET);
	write (fd, "", 1);
	lseek (fd, 0, SEEK_SET);

	int* ar;
	void* addr = mmap(NULL,1024,PROT_WRITE,MAP_SHARED,fd,0);
```

```
        close (fd);

        if (addr == MAP_FAILED) {
                perror("mmap");
                exit(1);
        }

        ar = (int*) addr;
        for (int i = 0; i < 100; i++) {
                ar[i] = 10000 + i;
        }

        std::cout << "Escrito el arreglo al memory-mapped file" << std::endl;
        munmap(addr,len);
}
```

27. Explique en un diagrama el funcionamiento de la tabla de páginas invertida .

- Combina una tabla de frames y una de páginas en una sola estructura.
- Una entrada por cada frame.
- El índice es el número de frame.
- Contiene el número de la página en memoria.
- La búsqueda es asociativa por contenido (TLB).
-



28. En Windows ¿Qué es el idle thread (o idle process o system process)?

Es un hilo que se encuentra en ejecución todo el tiempo, en background, y no puede ser terminado. Se encarga de controlar el ahorro de energía y el uso del CPU.

Se activa cuando no hay threads listos para ejecutarse.

- Verifica y termina los DPC (Deferred Procedure Call) que hubiera.
- Habilita interrupciones.
- Después de un tiempo llama a las rutinas de ahorro de energía.

29. En Windows: ¿Cual es la relación entre procesos y threads, que función tiene cada uno, cual es la unidad de scheduling?

Cada proceso de Windows es representado por una estructura executive process (EPROCESS). Además de contener muchos atributos relacionados a un proceso, un EPROCESS contiene y apunta a un número de estructuras relacionadas. Por ejemplo, cada proceso tiene uno o más threads, cada uno representado por una estructura executive thread (ETHREAD).

A nivel de sistema operativo, un thread Windows es representado por un objeto executive thread. El objeto executive thread encapsula una estructura ETHREAD, que contiene una estructura KTHREAD como su primer miembro. La estructura ETHREAD y las otras estructuras a las que apunta existen el el espacio de direcciones del sistema, con la excepción del thread environment block (TEB), el cual existe en el espacio de direcciones de proceso.

Windows implementa un sistema de scheduling apropiativo, regido por prioridades. Al menos uno de los threads de mayor prioridad corre siempre, con la advertencia de que ciertos threads de alta prioridad, listos para correr pueden estar limitados por los procesos en los cuales pueden estar habilitados para correr.

El código de scheduling de Windows está implementado en el kernel. No hay ningún módulo o rutina scheduler. Sin embargo, el código está esparcido en el kernel, en el cual ocurren los eventos relacionados con el scheduling. Las rutinas que realizan estas tareas son llamadas colectivamente kernel dispatcher

30. ¿Quien implementa el software de RAID y quien implementa el file system (el controlador o el Sistema Operativo) en
a) Un sistema con SAN?
b) Un sistema con NAS?
En cada caso haga un diagrama indicando que protocolo puede usarse en cada comunicación.

NAS provee almacenamiento y un file system. Esto contrasta con SAN (Storage Area Network), el cual provee sólo almacenamiento basado en bloques y deja el manejo del file system en el lado del cliente. Respecto al software RAID, cualquiera de las dos arquitecturas soporta dicha implementación.