BUBBLE SORT

```
//ordenamos (bubble sorting)
    for(int i = inicio; i < fin-1; i++) {
        for(int j = i+1; j < fin; j++) {
            if(numeros[i] > numeros[j]) {
                aux = numeros[i];
                numeros[i] = numeros[j];
                numeros[j] = aux;
            }
        }
    }
```

PRIMOS

```
//proceso 2 verifica numero PRIMO
void proceso2(pid32 pid1){
  int n = receive(); //n es el i que mande en el proceso principal
  int msg = 1; //es primo

  if(n == 0 || n==1){
    msg=2; //no es primo
  }else{
    for(int i = 2 ; i<n ; i++ ){
      if(n % i == 0 ){
    msg = 2; // no es primo
      }
    }
  }
  send(pid1, msg); // le mando la respuesta al proceso principal
  exit();
}
```

BISIESTO

```
//proceso 3 veridica si es BISIESTO

void proceso3(pid32 pid1){
  int i = receive();
  int msg=4; // no es bisiesto
```

```
  if(i%4 ==0 && i%10 !=00){
    msg=3; // es bisiesto
  }
  send(pid1, msg); // le mando la respuesta al proceso principal
  exit();
}
```

PALINDROMO

```
int esPalindromo(const char *s) {
    int i = 0;
    int j;
    while (s[i] != '\0') i++;  // Calcular longitud
    j = i - 1;
    i = 0;
    while (i < j) {
        if (s[i] != s[j])
            return 0;
        i++;
        j--;
    }
    return 1;
}
```

TEORICAS PARCIAL 2024

a. Defienda o critique la siguiente frase (justificar su postura): Una rutina de atención de interrupciones puede bloquear, por ejemplo, al tener que accionar algún hardware y esperando a que responda.

"Una rutina de atención de interrupciones puede bloquear, por ejemplo, al tener que accionar algún hardware y esperando a que responda"
Esta frase es verdadera. Un proceso que quiere enviarle algo a algun hardware puede suceder que antes de continuar necesite la respuesta del hardware que todo está correcto. Para esto la opcion optima seria bloquear a dicho proceso para que no consuma tiempo de cpu, y que otros procesos sigan con su flujo de ejecucion. Sin embargo, esto no es absolutamente necesario, pues puede ser que el proceso siga realizando tareas y en el momento que el hardware termine, envia un mensaje diciendo que termino y de esta manera el proceso atienda dicha interrupcion.

Lo que no debe suceder es que en este punto se bloquee absolutamente toda la cpu por mucho tiempo, pues esto hace que ningun otro proceso pueda seguir con su ejecucion, y en general este tipo de operaciones son muy costosas en tiempo. Por ello, a lo sumo, debe bloquear al proceso que ha llamado a la operacion (por ejemplo, porque solicito algun dato al teclado).

b. ¿Considera que el programa convertir_a_gris.c original visto en la práctica, podría ser portado sin mucho esfuerzo a MAC OS? ¿O a FreeBSD?. Si/no, porqué. Justifique su respuesta.

Cualquier programa escrito en un un so tipo UNIX podra ser portado facilmente a cualquier otro so del mismo estilo. Esto se debe a que todos siguen el standard POSIX, en el cual se define como deberia llamarse las syscalls en todos estos sistemas. De esta manera al portar un programa en c de Linux a MAC OS o FreeBSD por ejemplo, lo podriamos hacer sin problemas, debido a que todos estos son de tipo UNIX

c. Defienda o critique la frase. Justificar: En XINU los procesos no pueden utilizar el mecanismo de comunicación inter procesos “memoria compartida”, porque el sistema operativo no provee tal funcionalidad.

En XINU los procesos no pueden utilizar el mecanismo de comunicación inter procesos “memoria compartida”, porque el sistema operativo no provee tal funcionalidad"
Esta frase es verdadera. Esto se debe a que los procesos en XINU no funcionan como procesos como tal, sino que son mas parecidos a los threads que usamos en Linux. Esto se debe a que en xinu los "procesos" comparten la seccion de data del so. Por esta razon no habria necesidad de crear una seccion de memoria compartida, ya que simplemente usando cualquier tipo de variable global podemos compartir recursos entre si.

d. Defienda o critique la siguiente declaración: En un driver de dispositivo de E/S utilizar un buffer (como el implementado en la práctica) no tiene mucho sentido en realidad, ya que un proceso puede inmediatamente consumir una entrada (o generar una salida) cuando el driver obtiene un dato un de entrada via una interrupción (o emitió un dato de salida).

"En un driver de dispositivo de E/S utilizar un buffer (como el implementado en la práctica) no tiene mucho sentido en realidad, ya que un proceso puede inmediatamente consumir una entrada (o generar una salida) cuando el driver obtiene un dato un de entrada via una interrupción (o emitió un dato de salida)"
Esto es completamente falso. Los drivers estan divididos en dos mitades, el lower-half y el upper-half. El lower-half se encarga de comunicarse con el hardware, mientras que el upper-half se comunica con las aplicaciones. Es trabajo de los programadores realizar una sincronizacion correcta entre estas

dos mitades. El sentido de hacer esto es para que no sea el proceso que solicita un servicio a un dispositivo el que directamente obtiene el datos, sino que le pide al proceso del lower-half que le de el dato y se sincronicen entre ellos. Si el proceso del upper-half accediera directamente al hw, estaria este reaizando siempre interrupciones directamente al sistema y siempre es mejor usar esta comunicacion. Esto lo haran mediante el buffer, por ejemplo, en donde el procesos del upper-half solicita algun dato o envia algun dato, se bloquea (si se requiere), y espera que el proceso del lower-half realice las acciones acciones necesarias y desbloquee al otro proceso si fue bloqueado para que continue con su ejecucion, sabiendo que recivio/mando el dato que queria.

e. Con lo aprendido durante el cursado, ¿considera que los desarrolladores de Android (*) habrán tenido en cuenta POSIX?. ¿Por qué lo harían?. Justifique su respuesta.
(*) (Google, quien posiblemente su principal misión es generar ingresos, y podrían o no querer ser compatibles con otros sistemas).

Lo mas posible es que los desarrolladores de android si hallan tenido en cuenta POSIX. Esto se debe a que si algun usuario que estaba utilizando otro so de tipo unix, quiera pasarse a Android por ejemplo, lo mas posible es que lo haga solamente si no debe empezar de cero con todos sus programas que tiene.
Al utilizar POSIX esto sera mucho mas sencillo, ya que los mismos programas seran facil portados por usar tambien este standard. Pero si android cambiaria todo y realizaria su propia catalogo de funciones, lo mas posible es que los usuarios no quieran pasarse aqui, debido a que tienen que empezar con sus programas de 0. Y los usuarios en general prefieren tener todo rapido, lo antes posible y no tener que aprender todo desde cero.

Ejercicio 3. *Implementación de sistemas de archivos.*

En un disco de 100GB se cuenta con una partición en donde se ha creado un sistema de archivos de tipo UNIX/Linux. Se sabe que cada bloque en el disco es de 8KB. También, se conoce que cada inodo contiene 13 entradas directas, una indirecta, y una doble indirecta. Los punteros a bloques de disco ocupan 4 bytes. El inodo del directorio raíz ya se encuentra en memoria (el sistema operativo ya puede acceder a él). Todas las entradas de un directorio caben en un sólo bloque del filesystem/disco. Las entradas de directorio están compuestas 32 bytes: 4 bytes para el nro de inodo y 28 bytes para el nombre del archivo/directorio.

a. ¿Cuál es el tamaño máximo de un archivo en este sistema de archivos?
b. ¿Cuántos archivos y directorios caben debajo de un directorio?
c. ¿Sería conveniente este sistema de archivos para almacenar películas bajadas con torrent?

a) Cual es el tamaño maximo de un archivo en este sistema de archivos?

8192bytes / 4bytes = 2048 -> cantidad de punteros que entran en un bloque

TAMAÑO TOTAL DEL SISTEMA DE ARCHIVOS = Tamaño total entradas directas + Tamaño total entrada indirecta + Tamaño total entrada doble indirecta

Tamaño total entrada directa:
13 entradas directas * 8192 bytes = 106,496 bytes

Tamaño total entrada indirecta:
1 * 2048 * 8192 = 16.777.216 bytes

Tamaño total entrada doble indirecta:
1 * 2048 * 2048 * 8192 bytes= 34.359.738.368 bytes :D

Tamaño máximo del sistema de archivos: 106.496 + 16.777.216 + 34.359.738.368 = 34,376,622,080 bytes = 34381522.08 KB = 34.38152208 GB

b) Cuantos archivos y directorios caben debajo de un directorio?

tamaño bloque = tamaño directorio

8192 b / 32 b = 256 archivos/directorios

tamaño bloque / tamaño por entrada de directorio

2023

1. Si las páginas del sistema de memoria virtual en los sistemas Linux de los laboratorios son de 4KB, ¿cuales son los 12 bits menos significativos de la dirección física que corresponde a la dirección virtual obtenida en el ejercicio 1.b. ? Es decir, luego de que el sistema realizó la traducción de la dirección virtual a física. Utilice el sistema hexadecimal para especificar la respuesta.

INICIO DE DIRECCION VIRTUAL: 0x7c2ff17ac000me
Los 12 bits menos significativos de una dirección (virtual o física) cuando las páginas son de 4KB, son los últimos 3 dígitos hexadecimales.

2. En C en Linux existen las funciones shm_open(), ftruncate() y mmap() para crear un segmento de memoria compartida que luego se puede compartir con otros procesos para comunicación entre procesos. Indique en XINU cuáles serían las funciones en C equivalentes, y si no existen, expliqué cómo se logra ese mecanismo de comunicación entre procesos.

3. Sean los siguientes diagramas de un UNIX file system:

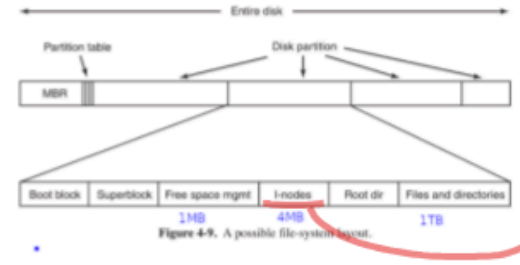
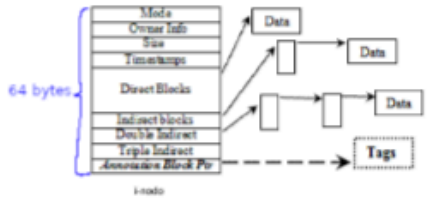


Figure 4-9. A possible file-system layout.

Next might come information about free blocks in the file system, for example in the form of a bitmap or a list of pointers. This might be followed by the i-nodes, an array of data structures, one per file, telling all about the file. After that might come the root directory, which contains the top of the file-system tree. Finally, the remainder of the disk contains all the other directories and files.



¿Cuántos archivos máximo puede almacenar este sistema de archivos?

16.777216 MB -> si queres tener 65536 archivos en un TB

4 MB / 64 Bytes = 65536 cantidad de inodos

Teniendo en cuenta que hay un i-nodo por archivo, y la cantidad maxima de inodos es 65583 (en 4MB). Para ocupar TODOS los inodos posibles todos los archivos deberian pesar 16.777216 MB, por lo tanto si los archivos varian en tamaño variaria el numero maximo de archivos que podrían haber.

recu 2023

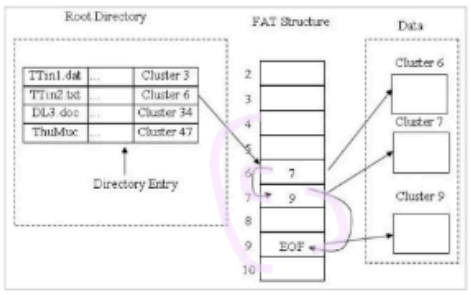
1. ¿Qué problemas podría ocasionar a su sistema XINU que un programa no utilice apropiadamente el paradigma open(), read(), close()?. Por ejemplo, si no se utiliza apropiadamente en la resolución del ejercicio 2. Justifique su respuesta.

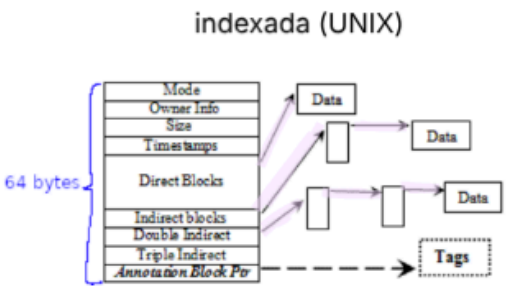
- En XINU, **no seguir el paradigma open() → read()/write() → close()** puede causar **fugas de descriptores, bloqueos de dispositivos, lecturas inválidas y corrupción de datos**.
- En un sistema tan controlado y limitado como XINU, esto **afecta no solo al proceso, sino a todo el sistema**.

2. Es de conocimiento general que los sistemas Windows de empresas y personas sufren ataques de distintas índoles todo el tiempo. Millones de agujeros de seguridad en Windows se han reportado en internet en los últimos 30 años. Si usted desarrolla dos programas en Windows (prog1 y prog2), y dos programas en XINU (prog1 y prog2), responda: ¿en qué sistema es más sencillo realizar un ataque desde prog1 a prog2?. Justifique su respuesta.

Sí, XINU es vulnerable porque no usa memoria virtual, lo que significa que **no hay aislamiento entre procesos y todos usan direcciones físicas reales**.
Esto permite que prog1 acceda o destruya datos de prog2 fácilmente, algo que en Windows sería mucho más difícil sin explotar vulnerabilidades del sistema.

3. Sean las siguientes estructuras de dos sistemas de archivos: FAT y un FS UNIX.





FAT FS UNIX

- Suponga que el i-nodo de un archivo deseado ya está en RAM en el UNIX FS.
- Suponga que la FAT está en RAM.
- Suponga que en ambos sistemas el tamaño máximo para un archivo son iguales, y los bloques de datos son del mismo tamaño.

Si sólo se desea conocer los números de bloques en el disco que corresponden a los datos del archivo deseado (no su contenido), ¿en cuál de los dos sistemas de archivos sería más veloz obtener todos los números de bloques de datos que pertenecen al archivo?. Justifique.

Con los 12 punteros directos en unix, ya accede los primeros 12 bloques, mientras que con la tabla fat, tendria que ir accediendo bloque por bloque, lo que terminaria demorando mas

2024

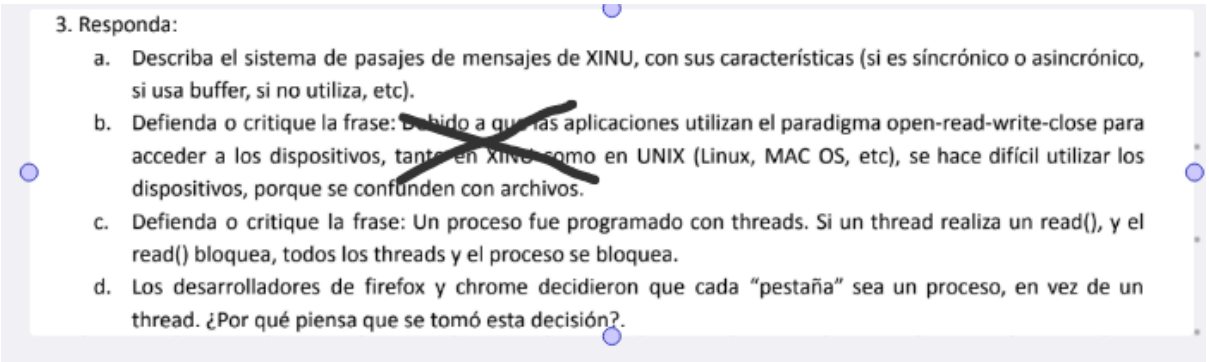
3. Responder:

a. Defienda o critique la siguiente frase (justificar su postura): Una rutina de atención de interrupciones puede bloquear, por ejemplo, al tener que accionar algún hardware y esperando a que responda.
b. ¿Considera que el programa convertir_a_gris.c original visto en la práctica, podría ser portado sin mucho esfuerzo a MAC OS? ¿O a FreeBSD?. Si/no, porqué. Justifique su respuesta.
c. Defienda o critique la frase. Justificar: En XINU los procesos no pueden utilizar el mecanismo de comunicación inter procesos "memoria compartida", porque el sistema operativo no provee tal funcionalidad.
d. Defienda o critique la siguiente declaración: En un driver de dispositivo de E/S utilizar un buffer (como el implementado en la práctica) no tiene mucho sentido en realidad, ya que un proceso puede inmediatamente consumir una entrada (o generar una salida) cuando el driver obtiene un dato un de entrada via una interrupción (o emitió un dato de salida).
e. Con lo aprendido durante el cursado, ¿considera que los desarrolladores de Android (*) habrán tenido en cuenta POSIX?. ¿Por qué lo harían?. Justifique su respuesta.
(*) (Google, quien posiblemente su principal misión es generar ingresos, y podrían o no querer ser compatibles con otros sistemas).

MIS RESPUESTAS:(mias nomas yo de)
a)
"Una rutina de atención de interrupciones puede bloquear, por ejemplo, al tener que accionar algún hardware y esperando a que responda"
Esta frase es verdadera. Un proceso que quiere enviarle algo a algun hardware puede suceder que antes de continuar necesite la respuesta del hardware que todo

está correcto. Para esto la opcion optima seria bloquear a dicho proceso para que no consuma tiempo de cpu, y que otros procesos sigan con su flujo de ejecucion. Sin embargo, esto no es absolutamente necesario, pues puede ser que el proceso siga realizando tareas y en el momento que el hardware termine, envia un mensaje diciendo que termino y de esta manera el proceso atienda dicha interrupcion.
Lo que no debe suceder es que en este punto se bloquee absolutamente toda la cpu por mucho tiempo, pues esto hace que ningun otro proceso pueda seguir con su ejecucion, y en general este tipo de operaciones son muy costosas en tiempo.
Por ello, a lo sumo, debe bloquear al proceso que ha llamado a la operacion (por ejemplo, porque solicito algun dato al teclado).
b)
Cualquier programa escrito en un un so tipo UNIX podra ser portado facilmente a cualquier otro so del mismo estilo. Esto se debe a que todos siguen el standard POSIX, en el cual se define como deberia llamarse las syscalls en todos estos sistemas. De esta manera al portar un programa en c de Linux a MAC OS o FreeBSD por ejemplo, lo podriamos hacer sin problemas, debido a que todos estos son de tipo UNIX
c)
En XINU los procesos no pueden utilizar el mecanismo de comunicación inter procesos “memoria compartida”, porque el sistema operativo no provee tal funcionalidad"
Esta frase es verdadera. Esto se debe a que los procesos en XINU no funcionan como procesos como tal, sino que son mas parecidos a los threads que usamos en Linux. Esto se debe a que en xinu los "procesos" comparten la seccion de data del so. Por esta razon no habria necesidad de crear una seccion de memoria compartida, ya que simplemente usando cualquier tipo de variable global podemos compartir recursos entre si.
e)
Lo mas posible es que los desarrolladores de android si hallan tenido en cuenta POSIX. Esto se debe a que si algun usuario que estaba utilizando otro so de tipo unix, quiera pasarse a Android por ejemplo, lo mas posible es que lo haga solamente si no debe empezar de cero con todos sus programas que tiene.
Al utilizar POSIX esto sera mucho mas sencillo, ya que los mismos programas seran facil portados por usar tambien este standard. Pero si android cambiaria todo y realizaria su propia catalogo de funciones, lo mas posible es que los usuarios no quieran pasarse aqui, debido a que tienen que empezar con sus programas de 0. Y los usuarios en general prefieren tener todo rapido, lo antes posible y no tener que aprender todo desde cero.

recu2024



3. Responda:

a. Describa el sistema de pasajes de mensajes de XINU, con sus características (si es síncrónico o asincrónico, si usa buffer, si no utiliza, etc).

b. Defienda o critique la frase: Debido a que las aplicaciones utilizan el paradigma open-read-write-close para acceder a los dispositivos, tanto en XINU como en UNIX (Linux, MAC OS, etc), se hace difícil utilizar los dispositivos, porque se confunden con archivos.

c. Defienda o critique la frase: Un proceso fue programado con threads. Si un thread realiza un read(), y el read() bloquea, todos los threads y el proceso se bloquea.

d. Los desarrolladores de firefox y chrome decidieron que cada "pestaña" sea un proceso, en vez de un thread. ¿Por qué piensa que se tomó esta decisión?.

a. Sistema de pasajes de mensajes de XINU
XINU usa pasaje de mensajes asincrónico entre procesos.
- No usa buffer: solo se puede tener un mensaje pendiente por proceso.
- Si un proceso envía cuando el receptor no ha recibido, el mensaje queda almacenado en una variable.
- Si el receptor ya tiene un mensaje, el envío falla.
b. Frase sobre open-read-write-close:
Crítica válida en parte.
El paradigma hace que todos los dispositivos se vean como archivos, lo que simplifica la interfaz, pero puede confundir a los desarrolladores, especialmente con dispositivos que no se comportan como archivos secuenciales
c. Frase sobre read() bloqueando a todos los threads:
Depende del modelo de hilos.
En sistemas con hilos del espacio de usuario (user-level threads), sí se bloquea todo el proceso si un hilo hace read().
En hilos del núcleo (kernel threads), otros hilos pueden seguir ejecutándose aunque uno se bloquee.
d. Pestañas como procesos (Firefox/Chrome):
Usar un proceso por pestaña aísla fallos (si una falla, no afecta a las otras), mejora la seguridad (sandboxing) y aprovecha mejor los núcleos de CPU. Aunque es más costoso en memoria, da mejor estabilidad y rendimiento.