Ingeniería Informática. Curso 3º.
Sistemas Operativos
Examen Final. TEORIA.
2 de Febrero de 2010

1. **[INTRODUCCION]**

a) Razones para desechar por inconveniente o errónea el uso de tal rutina.

- El proceso desde que se llama a la función puede terminar su rodaja de tiempo en mitad de la sentencia "for" – efecto de la multiprogramación

- Se puede estar realizando una operación de entrada/salida mediante DMA mientras se ejecuta el bucle ¨for", el tiempo de los robos de ciclo del DMA incrementará el tiempo de ejecución de dicha función.

- Si salta una interrupción mientras se ejecuta el bucle "for", el tiempo de ejecución de la rutina de tratamiento de la interrupción incrementa el tiempo de ejecución del bucle.

- Teniendo en cuenta que el tiempo se refiere a la función "retardo", no se ha tenido en cuenta ni el prólogo ni el epílogo de dicha función.

- En sistemas con memoria virtual, se podría dar una falta de página en la ejecución de dicha función (o del bucle "for").

- Si el sistema dispone de memoria cache, el tiempo que tarda la función en ejecutarse la primera vez es distinto al tiempo que tarda el resto de las veces (lo mismo es válido para el bucle "for").

- En el caso de que el bucle (o la función) se ejecutase en un determinado ordenador y deseásemos ejecutarla en otro, se debería tener en cuenta las características del nuevo sistema; además de lo dicho anteriormente se debería volver a medir, ya que la velocidad del procesador podría ser distinta. Otro detalle de la arquitectura que también podría influir es el tiempo de acceso a memoria, aunque el procesador sea el mismo, la memoria puede necesitar más (o menos) tiempo de acceso.

b) Es el mecanismo para solicitar un servicio al sistema operativo.

Al igual que una llamada a procedimiento o función normal produce un desvío del flujo de control, la diferencia fundamental radica en que la llamada al sistema se ejecuta en modo privilegiado; para ello se una instrucción especial de la CPU (ejemplo TRAP), que causa que el procesador pase a modo supervisor previamente. Para implementar una llamada al sistema se utiliza el mecanismo de excepciones o interrupciones software.

Cuando se invoca una llamada al sistema, la ejecución del programa que la invoca se interrumpe y sus datos se guardan normalmente en su BCP, para poder continuar ejecutándose cuan la llamada al sistema termine.

Ejemplo: fork(), exec(), read()

2. **[FICHEROS]**

Se necesitan 6 operaciones de lectura:

1. *Lectura del bloque correspondiente al directorio raiz para encontrar la entrada usr y su número de i-nodo.*
2. *Lectura del i-nodo correspondiente a la entrada usr para encontrar el bloque correspondiente al directorio usr.*
3. *Lectura del bloque correspondiente al directorio usr para encontrar la entrada curso3 y su número de i-nodo.*
4. *Lectura del i-nodo correspondiente a la entrada curso3 para encontrar el bloque correspondiente al directorio curso3.*
5. *Lectura del bloque correspondiente al directorio curso3 para encontrar la entrada SO y su número de i-nodo.*
6. *Lectura del i-nodo correspondiente a la entrada SO para encontrar el bloque correspondiente al directorio SO.*
7. *Lectura del bloque correspondiente al directorio SO para encontrar la entrada alumnos.txt y su número de i-nodo.*
8. *Lectura del i-nodo correspondiente a la entrada alumnos.txt*

a) *Organización: boot sector + 2 FAT's + directorio raíz + bloques de datos*
1 sector o bloque de boot (arranque)
2 FATs (original y copia) que ocupan 9 bloques cada una:
80x2x18 = 2880 bloques = 1,47 MB
2880x1,5 = 4320 bytes; 4320/512 = 8,4375 → 9 bloques
1 directorio raíz que ocupa 224x32 = 7168 bytes = 14 bloques
2880 – 1 – 9 – 9 – 14 = 2847 bloques de datos para ficheros y directorios

3. **[ENTRADA/SALIDA]**

a)
FIFO: A-B-C-D-E-F
SSTF: E-B-A-C-D-F
SCAN: A-C-D-F-E-B

Tiempo para SCAN: 37,2 + 0,2 +4,2 + 0,2 + 42,2 + 0,2
Atraviesa 150 cilindros en total.
Para la petición A:
*T posicionamiento: 70 cilindros * 0,5 = 35*
T transferencia: 4ms/pista : 20 sectores/pista = 0,2
T medio búsqueda : tiene que dar como término medio media vuelta (4ms/pista) = 2
Para la petición C:
T transferencia: 4ms/pista : 20 sectores/pista = 0,2
Para la petición D:
T transferencia: 4ms/pista : 20 sectores/pista = 0,2
T búsqueda : 4
Para la petición F:
T transferencia: 4ms/pista : 20 sectores/pista = 0,2

b) *50 lin/pag*80 car/lin = 4000 car/pag*
*4000 car/pag*6 pag/min = 400 car/seg => cada 2,5 mseg se imprime un carácter*
La rutina de interrupción necesita 0,05mseg. Luego si tiene sentido; no se pierden caracteres y se aprovechan 2,45mseg (2,5-0,05) que se utilizarían en hacer polling (espera activa).

4. **[PROCESOS]**

a) *Cambio de contexto:*
Fork(): cuando el proceso hijo obtiene mayor prioridad
Exec(): nunca
Wait() si entra en bloqueo al haber procesos hijos sin terminar
Exit() siempre
Situaciones de fallo:
Fork():
1. *La propia llamada puede provocar desbordamiento de pila en el proceso llamante.*
2. *Esta llamada crea un duplicado del proceso que la llama para lo cual necesita duplicar el espacio de memoria del proceso, esto puede provocar desbordamiento de memoria.*
3. *También se pueden haber agotado los bloques descriptores de proceso (BCP).*

Exec():

1. *La propia llamada puede provocar desbordamiento de pila en el proceso llamante.*
2. *Esta llamada cambia el código del programa que se ejecuta, puede ocurrir que el fichero no se encuentre (no exista), que el fichero no sea ejecutable o que no tenga permiso de ejecución.*
3. *También puede darse desbordamiento de memoria, ya que puede ocurrir que el código del programa no quepa en el espacio reservado.*

Wait():

1. *La propia llamada puede provocar desbordamiento de pila en el proceso llamante.*
2. *Esta llamada espera a que uno de sus hijos termine; dará error si el proceso llamante no ha creado ningún hijo.*
3. *El proceso hijo cuando termina devuelve en una variable información sobre su estado, si el argumento &status no es una dirección válida se dará un error.*

Exit():

1. *La propia llamada puede provocar desbordamiento de pila en el proceso llamante.*

b) *Un proceso a lo largo de su vida activa puede estar en tres estados: en ejecución (en la CPU), preparado (en la cola de preparados) y bloqueado (en alguna otra cola de espera). Si, por ejemplo, un proceso se pasa el 90% de su vida bloqueado, casi nunca ocupará la cola de preparados ni la CPU. Y eso le ocurre a los procesos intensivos en E/S, que pasan casi toda su existencia bloqueados en espera de un evento de E/S. Si tomamos como ejemplo un sistema interactivo típico con varios usuarios que suman esos cien procesos, que sólo de vez en cuando necesitan ejecutar instrucciones en CPU, la cola de preparados estará vacía casi todo el tiempo, y cuando alguno de dichos procesos intensivos en E/S quiera ejecutar instrucciones, será muy poco probable que tenga que competir con otro proceso por la CPU. De ahí una media tan baja en la ocupación de la cola de preparados. En realidad, el tamaño de la cola de preparados no tiene por qué ser proporcional al número de procesos activos.*

5. [**COMUNICACION Y SINCRONIZACION**]

a) Inicialización:

```
sem_init(huecos, 1);
sem_init(elementos, 0);
n=0;
```

Productor:

```
produceDato ();
lock(m);
while (n==1) cond_wait(huecos, m);
unlock(m);
enviaDato();
lock(m);
n=1;
cond_signal(elementos);
unlock(m);
```

Consumidor:

```
lock(m);
while (n==0) cond_wait(elementos, m);
unlock(m);
utilizaDato();
lock(m);
n=0;
cond_signal(huecos);
unlock(m);
```

b) Se presentan dos soluciones distintas:

Solución 1 (4 semáforos):

P1: cuerpo; V(a); V(a);
P2: P(a); cuerpo; V(b); V(b);
P3: P(a); cuerpo; V(c); V(c);
P4: P(b); cuerpo; V(d);
P5: P(b); P(c); cuerpo;
P6: P(c); P(d); cuerpo;
Todos los semáforos iniciados (a..d) a 0.

Solución 2 (5 semáforos):

P1: cuerpo; V(s2); V(s3);
P2: P(s2); cuerpo; V(s4); V(s5);
P3: P(s3); cuerpo; V(s5); V(s6);
P4: P(s4); cuerpo; V(s6);
P5: P(s5); P(s5); cuerpo;
P6: P(s6); P(s6); cuerpo;
Todos los semáforos (s2..s6) iniciados a 0.