CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)
Facultat d'Informàtica de Barcelona,
Dept. d'Arquitectura de Computadors,
curs 2019/2020 – 2Q

2n Control Parcial **21 d'abril de 2020**

- L'examen és individual
- Lliureu al Racó un document pdf, docx, odt o txt amb les respostes.
- L'examen és amb llibres i apunts.
- Temps d'examen 1h i 40 minuts

1. **Què és un Sistema Operatiu?**

   Al 1982 es va llançar el mercat oficialment el MSDOS, el sistema operatiu del IBM PC i ordinadors compatibles. Aquesta màquina tenia un processador Intel 8088, de 8 bits i només un mode d'execució. A continuació mostrem un petit fragment del codi del MSDOS:

```
ENTRY:  ;System call entry point and dispatcher
        POP     AX              ;IP from the long call at 5
        POP     AX              ;Segment from the long call at 5
        POP     CS:[TEMP]       ;IP from the CALL 5
        PUSHF                   ;Start re-ordering the stack
        CLI                     ;Disable interrupts
        PUSH    AX              ;Save segment
        PUSH    CS:[TEMP]       ;Stack now ordered as if INT had been used
        CMP     CL,MAXCALL      ;This entry point doesn't get as many calls
        JA      BADCALL
        MOV     AH,CL
        ...
```

   El nombre de bits del processador no era rellevant per a la pregunta, però vistes les vostres respostes, faig un petit aclariment. Si ens diuen que un processador és de 8 bits vol dir que el bus de dades és de 8 bits, ie, l'i8088 només podia llegir 8 bits cada vegada. Això, la Bus Interface Unit (BIU), el diferenciava de l'i8086 de 16 bits, tot i que tots dos tenien la mateixa Execution Unit (EU) i eren compatibles quant a codi assemblador. Feien servir adreçament segmentat, amb registres de segment de 16 bits i 4 bits d'offset. En conseqüència, la mida màxima de l'espai d'adreces era d'1 MB (2^{20}).

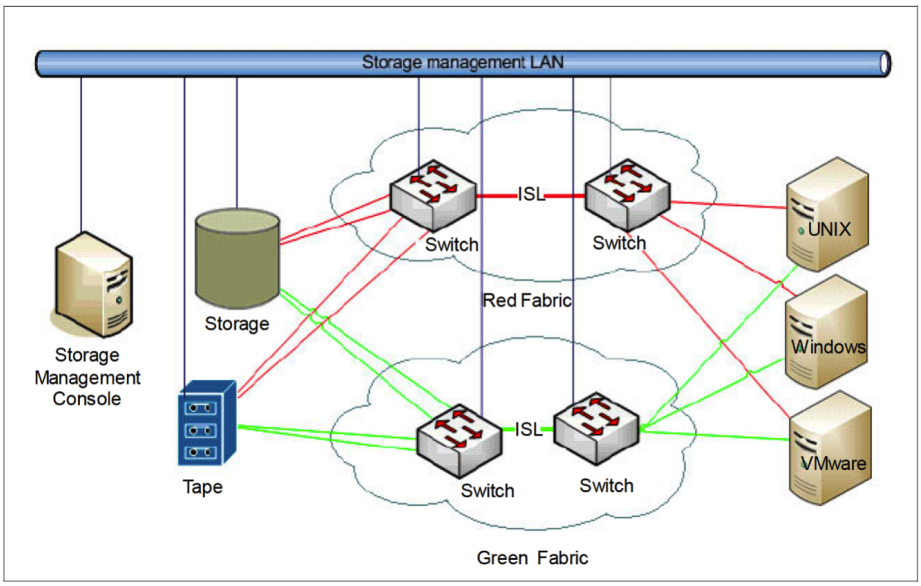
   Un sistema operatiu és un programa que fa d'intermediari entre l'usuari i la màquina. Proporciona un entorn d'execució entre convenient i eficient per executar programes. Gestiona la màquina d'una manera segura i proporciona protecció als usuaris. Protecció és un mecanisme per a controlar l'accés de programes, processos o usuaris als recursos de l'ordinador. La seguretat protegeix la integritat de la informació emmagatzemada al sistema, així com els recursos físics d'aquest d'accessos no autoritzats. Protecció i seguretat no es poden aconseguir amb un sol mode d'execució.

Sense dos modes d'execució el sistema operatiu no pot comprovar si el procés que està executant té privilegis suficients per llançar segons quines instructions. Quan tenim dos modes d'execució i executem una syscall, el codi de kernel comprova, dins la rutina d'atenció al `trap`, que el procés està en mode privilegiat, si no és així genera una interrupció (a Intel, la `int 13`) associacda a la *General Protection Fault* (GPF). La resposta del sistema operatiu acostuma a ser treure al procés del sistema. Al codi es pot apreciar que les interrupcions estan inhibides (la instrucció `CLI`, clear interrupt-enable, posa a 0 el flag IF, veure el comentari al mateix codi). No es produeix aquest mecanisme de control.

Linux va ser creat el 1991 per Linus Torvalds quan era estudiant. S'inspirà en un micro-kernel didàctic anomenat MINIX, creat pel professor Andrew S. Tanenbaum el 1987. La primera versió de Linux corria sobre l'i386, aquest processador permetria anar i tornar fàcilment del mode protegit. L'i386 heretà la part de 16 bits de l'i286 (que introduí els modes d'execució a Intel) i afegí els 32 bits. Arrancava en *real-address mode*, inicialitzava les estructures de dades de sistema i els registres, preparant-se per passar a *protected mode*. Per fi es podia implementar la protecció i seguretat, punt bàsic en la definició de sistema operatiu.

2. SAN o NAS

A la figura teniu l'esquema d'una xarxa, a on "storage" és la unitat d'emmagatzematge que guarda els fitxers dels usuaris de diferents sistemes operatius.



(a) ¿Quin entorn creus que representa, SAN o NAS?

Es tracta d'una SAN. La pista està en el *out-of-ban management* típic de les *Storage Area Networks*. A la figura es poden veure dues xarxes, la fiber-channel i l'ethernet (dedicada a la gestió de la SAN).

(b) Explica breument, basant-te en el dibuix, com funciona.

Cada client (Unix, Windows i VMware) connecta directament als discos, via fiber-channel. Cadscun d'ells veu un dispositiu de blocs propi. Un sistema de fitxers específic per aquest tipus d'instal·lacions gestionarà l'accés concurrent de tos els clients de la SAN.

L'array de discos està connectat directament a la xarxa fiber-channel i a l'ethernet, la qual cosa permet gestionar la SAN fins i tot si algun disc ha caigut. Tot el tràfic de gestió està separat del tràfic de dades.

(c) Quina és la diferència fonamental entre SAN i NAS?

La diferència fonamental és que NAS accedeix a nivell de fitxer i SAN a nivell de bloc.

(d) Cita algun sistema de fitxers pensat explícitament per aquest entorn.

Hi ha molts. Alguns dels esmentats a classe són: cxfs i ocfs.

3. **Enginyeria inversa**

Hem executat la comanda strace sobre un binari. A partir d'aquest extracte de la seva sortida, intenta deduir alguna de les crides de més alt nivell que s'han executat.

```
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7ffa6d1ea000
mprotect(0x7ffa6d1eb000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7ffa6d9e9fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ffa6d9ea9d0
, tls=0x7ffa6d9ea700, child_tidptr=0x7ffa6d9ea9d0) = 6216
sched_setscheduler(6216, SCHED_FIFO, [1]) = 0
futex(0x7ffa6d9ead18, FUTEX_WAKE_PRIVATE, 1) = 1
sched_get_priority_min(SCHED_FIFO)      = 1
sched_get_priority_max(SCHED_FIFO)      = 99
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7ffa6c9e9000
mprotect(0x7ffa6c9ea000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7ffa6d1e8fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7ffa6d1e99d0
, tls=0x7ffa6d1e9700, child_tidptr=0x7ffa6d1e99d0) = 6217
sched_setscheduler(6217, SCHED_FIFO, [2]) = 0
```

Hi ha dues crides a `clone(2)` per crear sengles threads dins el mateix grup de threads (CLONE_THREAD) amb els flags típics (quant a compartició de memòria: shared file descriptors CLONE_FILES, file system information CLONEFS, table of signals handler CLONE_SIGHAND) de crear un flux d'execució (i no pas un procés). Per tant no pot ser la implementació d'un `fork(2)`. Més aviat seran quelcom semblant a:

```
ret = pthread_create(&thread->pthread, &thread->attr, func,
                             (void *)thread)))
```

Tal com vau poder comprovar amb l'exercici del factorial amb pthreads, la crida al sistema `mprotect(2)` pot estar involucrada amb la implementació de `pthread_mutex_init(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`. La pista definitiva que apunta als mutex és la línia `futex(0x7ffa6d9ead18, FUTEX_WAKE_PRIVATE, 1) = 1`; sabent que la família de funcions relacionades amb l'exclusió mútua de la llibreria pthread s'implementen

amb `futex(2)`.

4. **Memòria amb `mmap`**

   La crida al sistema, segons el manual, *allocate memory, or map files or devices into memory* , per tant, té funcionalitats semblants a `malloc` i `read`, respectivament. Compara, per separat, `mmap` amb cadascuna d'elles i assenyala situacions on faries servir una o l'altre.

   La crida al sistema `mmap(2)` assigna memòria a un procés a qualsevol punt de l'espai lògic d'aquest. Aquest fet la diferencia de `sbrk(2)` i `sbrk(2)`, més antigues, que assignen la memória dins el heap.

   (a) `mmap` vs `read`

   Fer servir `mmap` per llegir un fitxer a memòria és millor que llegir-lo amb `read` si el fitxer és gran (però no massa), es farà servir moltes vegades i es compartirà (cas de les llibreries). Fer-ho així evita la funció `copy_to_user` del kernel, i estalvia les crides a `lseek`. Però cal tenir en compte que: hi ha un overhead significatiu en la creació del mapeig, que el fitxer hi ha de cabre dins l'espai d'adreces del procés i que el mapeig serà sempre un nombre enter de pàgines.

   (b) `mmap` vs `malloc`

   La funció `malloc` no és una crida al sistema, sinó que pertany a la libC. Quan un procés crida a `malloc` per una mida petita, `malloc` crida a `brk` o `sbrk` amb una mida bastant més gran, per millorar el rendiment reduïnt el nombre de crides al sistema i administra les següents crides a `malloc` sense haver d'entrar en mode privilegiat. Però si la mida demanada és molt gran, `malloc` fa servir `mmap` i no pas `sbrk`, per evitar la fragmentació de memòria. Com en el cas anterior, l'overhead de `mmap` només es justifica per a mides grans. Aquest llindar de separació entre cridar a `sbrk` o a `mmap` és ajustable. Veure man mallopt(3).

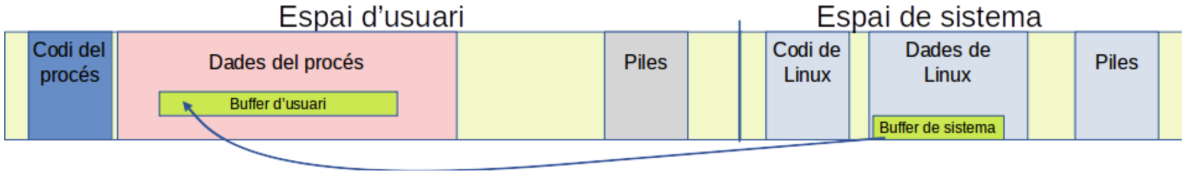
5. **Transferint dades** El sistema operatiu ha de transferir dades entre l'espai dels processos i el del sistema operatiu per tal de *i*) llegir informació dels dispositius amb seguretat i protecció i passar-la a l'usuari; o bé, *ii*) per llegir informació de l'espai d'usuari d'un procés i passar-la a dins del sistema operatiu per dur-la a un dispositiu. Així implementem les lectures i escriptures a dispositius, respectivament.

   Una de les rutines típiques de UNIX/Linux per dur a terme aquesta tasca és:

```
long  copy_to_user ( void __user * to, const void * from, unsigned long n );
```

   Aquesta funció rep el punter destí a l'espai d'usuari ("to"), el punter origen de la informació a l'espai de sistema ("from") i la longitud en bytes que volem copiar ("n"). A més, retorna el número de bytes que no s'han pogut copiar, per exemple perquè s'ha trobat amb què una de les adreces de l'espai de l'usuari no és vàlida. Per tant, si retorna 0 (zero) vol dir que tot ha anat bé. I si retorna una quantitat més gran que 0 (zero), vol dir que ha trobat un error en el byte "n - la quantitat retornada". En aquest cas posa la variable d'error ("errno") a EFAULT, per exemple, que és l'error que es retornarà a la crida a sistema

que es trobi amb aquesta situació. El dibuix que representa l'actuació d'aquesta funció és:



Responeu:

(a) Doneu una possible implementació bàsica d'aquesta funció "copy_to_user". Es valoraran les indicacions del control d'errors. Podeu usar una funció `bool bad_address (void * ptr);` que us retorna 1 (cert) si el punter "ptr" apunta a una adreça que l'usuari no és vàlida.

Per exemple, podriem copiar byte a byte, mirant per a cadascun que la seva adreça destí en el "to" sigui vàlida. Si no ho és, posem l'error a `EFAULT` i retornem els bytes que ens faltaven per copiar. Si tot va bé, retornem `n-i`, que serà zero (correcte).

```
long copy_to_user(void __user * to, const void * from, unsigned long n)
{
    int i;

    char * to_c = to;
    const char * from_c = from;

    for (i = 0; i < n; i++) {
       if (!bad_address(&to_c[i])) to_c[i] = from_c[i];
       else {
          set_errno(-EFAULT);
          return n-i;

     ++i;
    }

    return n-i;
}
```

S'hi poden fer optimitzacions, com ara veure si es pot fer la còpia de 8 bytes en 8 bytes i fer-la mentre la mida a copiar sigui més gran de 8.

O també es pot intentar fer el `bad_address` només un cop per pàgina, i fer còpies de pàgines senceres sense haver de fer comprovacions que son redundants.

(b) Tenint present el tema del com s'accedeix a les variables pròpies d'un thread (recordar la pràctica de Mach, exercicis 8 i 9), com podrieu implementar l'atribut "__user" que s'aplica al punter que apunta a l'espai de l'usuari (`void __user * to`, en l'exemple)?

A la pràctica de Mach vam veure que la llibreria de C de Hurd usa el segment "gs" dels processados Intel per accedir a variables que anomenem "thread local" i que estan a una secció especial (TLS). Això ens fa pensar que el SO podria tenir l'espai d'usuari accessible només a través del registre gs. Això protegiria l'espai d'usuari,

perquè complicaria els atacs que alteren dades dels processos o que agafen dades dels processos, perquè s'haurien de dissenyar per usar el registre gs per accedir-hi.