Cognoms: _____ Nom: _____ DNI: _____

CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)

Facultat d'Informàtica de Barcelona, Dept. d'Arquitectura de Computadors, curs 2019/2020 – 2Q

1r Control Parcial 10 de març de 2020

L'examen és individual
Responeu cada pregunta en un full separat
Indiqueu COGNOMS, NOM i DNI (per aquest ordre)
L'examen és sense llibres ni apunts, no es poden consultar ordinadors portàtils o mòbils
Justifiqueu totes les respostes
Temps d'examen: 1 hora

1. Què és un sistema operatiu? (2 punts)

La teva resposta ha d'incloure una definició de sistema operatiu, així com els teus criteris per triar-ne un o cap. Per últim, fes una comparativa, avaluant pros i cons, entre un sistema monolític, tipus Linux, i un microkernel, tipus Mach.

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.
Des del punt de vista del programador, un criteri per triar el teu sistema operatiu potser la necessitat de suport al desenvolupament, és dir, la quantitat de llibreries de que disposa el sistema. Aquest criteri té a veure amb els costos de desenvolupament, pero òbviament pot haver d'altres criteris igual de vàlids.
Un microkernel és un kernel petit, eficient que proveeix els serveis bàsics (com ara gestió de processos i de memòria). La resta del sistema s'implementa en processos servidors, que passen molt de temps en mode usuari. Això els fa ser, usualment, més fiables que el monolítics (si un servei cau, no cau tot el sistema). Basant-se en aquests serveis es poden enmascarar altres sistemes operatius (per exemple BSD Unix).
Un kernel monolític és un kernel gran, amb moltes crides al sistema, amb un sol procés executant-se en un sol espai d'adreces. Això els fa ser, usualment, més ràpids que els microkernels. Afegir un servei implica usualment recompilar el kernel.

2. Mach (2 punts)

Mach ofereix cinc abstraccions de programació que són el maons bàsics del sistema. D'aquestes, et demanem que defineixis només les quatre primitives següents:

Thread	Task	Message	Port

Thread	es el segon component del procés. La part activa. L'entorn d'execució d'una task. Cada task pot suportar més d'un thread executant-se concurrentment, tots compartint els recursos de la task. Tots els threads tenen el mateix espai d'adreces de memòria virtual (VM), però es diferencien en l'estat d'execució, format per un conjunt de

	registres, tals com l'stack pointer (SP) i el program counter (PC).
Task	el procés clàssic de Unix, a Mach es divideix en dos. La part que fa de contenidor de recursos, tals com memoria virtual o ports de comunicacions, se'n diu task. És una entitat passiva, no s'executa a cap processador.
Message	Els threads de diferents tasques es comuniquen per missatges. Un missatge és una col·lecció de dades amb tipus
Port	el canal de comunicacions mitjançant el qual es comuniquen dos threads. Un port és un recurs, propietat d'una task. Un thread té accés a un port pel fet de pertànyer a una task.

3. Linux (2 punts)

A Linux existeix la crida al sistema int `sched_setaffinity(pid_t pid, size_t cpusetsize, const cpu_set_t *mask);`

a) Quins efectes tindrà l'execució de les següents línies de codi per la resta del programa?

```
CPU_ZERO(&mask);
CPU_SET(2, &mask);
CPU_SET(1, &mask);
sched_setaffinity (getpid(), 4, &mask);
```

```
CPU_ZERO(&mask);  //0000
CPU_SET(2, &mask); //0100
CPU_SET(1, &mask);  //0110
sched_setaffinity (getpid(), 4, &mask); /* cpus 1 and 2 set for
pid==getpid()*/
```

b) Descriu un escenari a on un *thread* en concret aprofiti aquesta *syscall* per millorar el seu rendiment.

Dedicant una CPU a un thread particular (és a dir, definint la màscara d'afinitat d'aquest thread per especificar una CPU única i definint la màscara d'afinitat de tots els altres threads per excloure aquesta CPU), és possible assegurar la màxima velocitat d'execució d'aquest thread.
Restringir un thread per executar-se en una sola CPU també evita el cost de rendiment causat per la invalidació de la cache que es produeix quan un thread deixa de executar-se en una CPU i passa l'execució a una CPU diferent.

c) Descriu un escenari a on els *threads* d'un procés en concret aprofitin aquesta *syscall* per millorar el seu rendiment.

Anàlogament, restringint cada thread a una CPU diferent.

d) A POSIX, existeix la crida `int pthread_setaffinity_np(pthread_t thread, size_t cpusetsize, const cpu_set_t *cpuset)`, ¿quan seria recomanable fer-la servir en comptes de `sched_setaffinity(...)`?

Quan `pthread_create ()` s'utilitza en el mateix codi i el programa s'ha compilat i enllaçat amb -pthread. Tingueu en compte que la seva funció és una extensió GNU no estàndard; d'aquí el sufix "_np" (no portable) del nom.

4. Threads (2 punts)

Per a cadascuna de les següents línies de codi, indica quina funció de més alt nivell estan implementant i a quin sistema operatiu.

a) CreateThread((LPSECURITY_ATTRIBUTES)security, stacksize,
_threadstartex,
(LPVOID)ptd, createflag,
(LPDWORD)thrdaddr))

Es la funció de la Windows API per crear un thread

b) clone(child_stack, CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid, tls, child_tidptr)

Crida per crear un thread a Linux. Amb aquests paràmetres és la implementació de pthread_create()

c) clone(NULL, NULL, SIG_CHLD, NULL)

Crida per crear un procés a Linux, implementa un fork()

d) thread_create(self, &kernel_thread)

Crida per crear un thread a Mach.

5. Eines de desenvolupament (2 punts)

Entre les eines de desenvolupament tenim l'enllaçador (linker). GNU proporciona 2 enllaçadors, segons la transparència que vam veure a classe:

ld / gold →linker

A la Wikipedia trobem aquesta explicació:

In software engineering, **gold** is a linker for ELF files. It became an official GNU package and was added to binutils in March, 2008 and first released in binutils version 2.19. **Gold** was developed by Ian Lance Taylor and a small team at Google. The motivation for writing **gold** was to make a linker that is faster than the GNU linker, especially for large applications coded in C++.

Unlike the GNU linker, gold doesn't use the BFD library to process object files. While this limits the object file formats it can process to ELF only, it is also claimed to result in a cleaner and faster implementation without an additional abstraction layer. The author cited complete removal of BFD as a reason to create a new linker from scratch rather than incrementally improve the GNU linker. This rewrite also fixes some bugs in old ld that break ELF files in various minor ways.

To specify gold in a makefile, set the LD or LD environmental variable to ld.gold. To specify gold through a compiler option, one can use the gcc option -fuse-ld=gold.

Responeu:

a) Per què tenim dues versions del “linker”?

Un equip de Google va veure que el GNU ld és molt lent enllaçant fitxers objecte, i una de les raons és que manipula les estructures dels binaris a través la la llibreria de suport GNU bfd. Llavors, en col.laboració amb l'autor del GNU ld, Ian Lance Taylor, van desenvolupar el GNU gold. Com que el requeriment era que fos més ràpid, van fer-lo sense passar per la llibreria bfd, i només orientat a fitxers de format ELF. Però, en Linux, encara hi ha aplicacions en format COFF o a.out, i per això no es pot discontinuar l'antic GNU ld.

b) Quines limitacions té el ***gold***, si el comparem amb l’***ld***?

Que només suporta fitxers de format ELF.

c) En cas de tenir una aplicació formada per multitud de fitxers (C i/o C++) - per tant, usant compilació separada, i un Makefile, i per la qual volem generar un fitxer executable en format ELF, ordeneu per ordre de preferència aquestes alternatives que tenim per enllaçar-los:

c.1) export LD=ld.gold
make

c.2) export CFLAGS=”-fuse-ld=gold”
make

c.3) make

c.4) gcc -fuse-ld=ld *.c *.cpp

Expliqueu el perquè de l’ordre que heu decidit.

la preferència és

c2) export CFLAGS="-fuse-ld=gold"
make

Perquè
- és millor informar al gcc que faci servir el gold
- els Makefiles haurien de fer CFLAGS+="altres opcions" per afegir opcions i no esborrar la que els ve de fora per la variable d'entorn

c1) export LD="ld.gold"
make

Perquè
- habitualment no usem $(LD) per enllaçar en els Makefiles, sinó que ho fem amb "gcc -o ... " perquè així el gcc ja ens inclou totes les llibreries que calen segons les altres opcions que li passem.
- Només si el Makefile usa $(LD) per enllaçar aquesta opció donaria com a

resultat que el programa s'enllacés amb el gold

c3) make

Perquè
- Encara que no enllaçaríem tampoc amb el gold, obtindriem l'executable correcte i funcional, potser després d'esperar més estona

c4) gcc -fuse-ld=ld *.c *.cpp

Perquè
- En aquest cas li estem dient al gcc que enllaçi amb "ld"!!, no amb gold
- A més ens saltem totes les opcions que pugui tenir el Makefile per compilar...
  -O -g ... altres ... -o <nom-executable> ...
- No es pot assegurar que la compilació sigui existosa, imaginem només que l'aplicació tingui alguns fitxers C/C++ en subdirectoris. Ja no els veuriem i tindríem símbols no resolts.

No obstant això, a la correcció de la pregunta 5c s'han donat per bons altres ordres, si estaven ben argumentats.