Operativsystem: Lösningar till tentamen 2019-06-05

Observera att detta är förslag på lösningar. Det kan finnas andra lösningar som också är korrekta. Det kan hända att en del av lösningarna är mer omfattande än vad som krävs för full poäng på uppgiften, eller att de bara hänvisar till var man kan läsa svaren. Dessutom har det inträffat i världshistorien att lärare skrivit fel i lösningsförslagen. *Jag* har förstås aldrig gjort det, men andra. Är det verkligen någon som läser såna här inledande texter? Jag vet inte. Det kan vara så. Rabarber rabarber rabarber. Det har jag hört att statisterna får säga på filminspelningar när det ska vara bakgrundsorl från en restaurang eller liknande. Här nedan kommer lösningsförslagen till uppgifterna.

Uppgift 1 (2 p)

Vad är ett operativsystem?

Svar:

En enkel men kanske inte helt korrekt förklaring är att ett operativsystem är ett program som startas direkt varje gång datorn startas, och sedan håller reda på alla de andra programmen.

Kursboken skriver att operativsystemet fungerar som en mellanhand mellan användaren av en dator och datorns hårdvara, för att erbjuda en omgivning där användaren kan köra program bekvämt och effektivt.

Samma sak kan också uttryckas som att operativsystemet är programvara som skapar en virtuell maskin, dvs översätter datorns hårdvara till en mer praktisk och lättanvänd maskin. Processorn översätts till processer, det fysiska minnet översätts till en virtuell adressrymd per process, sekundärminnets block översätts till filer och mappar.

Uppgift 2 (3 p)

Vilka är operativsystemets viktigaste uppgifter?

Svar:

Om man ser operativsystemet som "ett program som håller reda på alla de andra programmen", så är dess uppgift förstås att hålla reda på de programmen. Om man ser operativsystemet som en "mellanhand mellan användaren av en dator och datorns hårdvara", för att erbjuda en omgivning där användaren kan köra program bekvämt och effektivt, så är uppgiften att erbjuda den omgivningen.

Men ska man vara mer konkret ska operativsystemet kunna:

- starta, avsluta och hantera processer (och trådar)
- schemalägga processerna, dvs bestämma när varje process ska köras på en processor
- hantera minne för processerna, särskilt om systemet använder virtuellt minne
- skapa, ta bort och hantera filer och filträd i ett filsystem
- skapa, ta bort och hantera användare
- hantera I/O till olika interna och externa enheter
- hantera delade resurser även i övrigt, till exempel nätverk och skrivare

- erbjuda säkerhet, dvs skydd mot både oavsiktlig och avsiktlig störning eller förstörelse av användarnas och systemets data och processer

Uppgift 3 (3 p)

Här är ett C-program för Linux, kallat "program 1", som läser en gigabyte data från en fil:

```
#include <unistd.h>
#include <fcntl.h>

#define KILO 1024
#define MEGA (KILO*KILO) // 1048576
#define GIGA (KILO*KILO*KILO) // 1073741824

char data[GIGA];

int main(void) {
    int fd = open("data.bin", O_RDONLY);
    for (int i = 0; i < KILO; ++i) {
        read(fd, &data[i * MEGA], MEGA);
    }
    close(fd);
}
```

I en annan version av samma program, kallat "program 2", byter vi ut for-loopen mot den här:

```
    for (int i = 0; i < MEGA; ++i) {
        read(fd, &data[i * KILO], KILO);
    }
```

(Skillnaden är alltså att vi bytt plats på "MEGA" och "KILO".)

Programmen gör samma sak, och de fungerar korrekt, men man kan förvänta sig att ett av programmen går betydligt snabbare än det andra.

a) Vilket program är snabbast?

Svar:

Program 1, det som läser en megabyte data åt gången.

b) Förklara varför skillnaden mellan programmen blir så stor!

Svar:

Program 2, som bara läser en kilobyte data åt gången, måste göra många fler systemanrop (anropen till **read**). Systemanrop är långsamma, eftersom man måste göra ett avbrott ("interrupt"), byta priviliger till kernel mode, och, beroende på omständigheterna, kanske ett fullständigt kontextbyte ("context switch"). Om de data som ska läsas inte finns buffrade av operativsystemet i primärminne, måste de hämtas från sekundärminne (SSD eller mekanisk hårddisk), och då kommer processen att placeras i en väntekö tills den hämtningen är färdig.

Uppgift 4 (5 p)

Operativsystemkärnan har, precis som många andra dataprogram, en hel del data som den håller reda på. Beskriv de viktigaste, och vilka datastrukturer man använder för dem.

Svar:

Det finns en datastruktur, den så kallade "task-structen", som representerar en process. Den innehåller processens nummer, ägare, prioritet, beskrivning av dess minnesanvändning, med mera. Task-structarna är ordnade i flera olika köer: en kö för processer som är klara att köra, flera olika köer för processer som väntar på någon resurs, och en samling för processer som kör.

Om systemet använder virtuellt minne har varje process en "sidtabell" (på engelska "page table") för översättning mellan virtuella och fysiska minnesadresser.

Det finns också en global tabell med alla öppna filer, och en tabell per process för den processens öppna filer, med referenser till den globala tabellen. Delar av filernas data finns också temporärt lagrade i buffertar, som operativsystemet håller reda på.

Hit kan man också räkna data från filsystemen, till exempel de "i-noder" som Unix använder.

Uppgift 5 (3 p)

En mikrokärna ("micro-kernel" på engelska) är ett sätt att bygga operativsystem. Vad innebär det att en operativsystem har en mikrokärna? Vilka fördelar och nackdelar medför det?

Svar:

En mikrokärna är en kärna i ett operativsystem som man försökt göra så liten som möjligt. Mikrokärnan innehåller bara grundläggande processhantering, som kommunikation mellan processer och schemaläggning av när processerna ska köras på processorn. Allt annat, som filsystem, drivrutiner för hårdvara, och fönsterhantering, placeras i vanliga processer ("user mode processes") som inte kör med kärnans rättigheter ("kernel mode") utan i "user mode" eller "user space".

En fördel med mikrokärnor är att om en av de funktioner som är placerade i vanliga processer utanför kärnan skulle krascha, så leder det inte till att hela systemet kraschar.

En nackdel är att prestanda kan bli sämre, eftersom kommunikationen mellan kärnan och processerna utanför kärnan kräver ett kontextbyte och kopiering av data.

Uppgift 6 (3 p)

Vad är en process? Hur skiljer sig en process från en tråd?

Svar:

En process är ett program som exekverar. Det har ett minnesutrymme och minst en exekveringstråd.

En tråd (ovan kallad exekveringstråd) består av en "programräknare", som anger platsen i minnet för den maskininstruktion i programmets körbara kod som just nu utförs, och en stack av funktionsaktiveringar. En process innehåller en eller flera trådar. Alla trådarna i en process delar processens gemensamma minnesutrymme, och just

detta att en process har ett eget minnesutrymme, medan trådar delar minne, brukar man ange som den stora skillnaden mellan trådar och processer. På ett flerprocessorsystem, med flera processorkärnor, kan flera trådar exekvera samtidigt, men på ett enprocessorsystem kan bara en enda tråd exekvera samtidigt, och då får de turas om genom att operativsystemet snabbt växlar mellan dem.

Uppgift 7 (2 p)

Här är ett C-program för Linux. Vad skrivs ut när programmet körs?

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
    int x = 1;
    fork();
    x = x + 1;
    printf("Hej! x = %d!\n", x);
    fork();
    x = x + 1;
    printf("Hopp! x = %d!\n", x);
}
```

Svar:

```
Hej! x = 2!
Hej! x = 2!
Hopp! x = 3!
Hopp! x = 3!
Hopp! x = 3!
Hopp! x = 3!
```

Kommentar: Det blir totalt fyra processer som körs. (Det andra fork-anropet görs i två olika processer.) Ordningen på utskrifterna kan variera lite, eftersom processerna körs parallellt.

Uppgift 8 (4 p)

a) Vad innebär virtuellt minne?

Svar:

Varje process som körs har sin egen adressrymd, så kallade virtuella adresser, som måste översättas till fysiska adresser för att avgöra var i det fysiska minnet som processens data verkligen är lagrade. Det kan vara en enkel översättning, till exempel att man lägger till en konstant, eller en mer komplicerad översättning med tabelluppslagning.

b) Varför måste man ha stöd i hårdvaran när man använder virtuellt minne?

Svar:

Annars blir det för långsamt. Om översättningen gjordes i mjukvara skulle det behövas minst en maskininstruktion för att räkna ut den fysiska adressen, och kanske många

fler om det är en komplicerad översättningsmetod eller om minnesgränser ska kontrolleras. Om man dessutom använder en tabell (som inte är mycket liten) för översättningen kommer den att behöva lagras i primärminne, och då behövs minst en minnesåtkomst för varje översättning. Det skulle fördubbla belastningen på primärminnet, eftersom varje minnesåtkomst nu kräver två minnesåtkomster.

c) Varför är sidstorlekar ("page size" på engelska) alltid jämna tvåpotenser? Visa med exempel!

Svar:

För att effektivt (dvs snabbt och med lite hårdvara) kunna dela upp den virtuella minnesadressen i sidnummer (på engelska "page number") och offset inom sidan. Den uppdelningen behövs för att göra översättningen från virtuell adress till fysisk adress.

Om sidstorleken exempelvis är 2^{11}, dvs 2048, är de 11 lägsta bitarna offset i den virtuella adressen offset inom sidan, och de övriga, högre bitarna sidnummer. Dessa högre bitar kan skickas direkt till uppslagningen i sidtabellen, utan att några matematiska operationer behöver utföras. (Man behöver bara dra sladdarna med bitarna i till rätt plats!), Uppslagningen ger ett ramnummer (på engelska "frame number") i det fysiska minnet. Bitarna i ramnumret kombineras med offset-bitarna till en fysisk minnesadress.

Om vi också antar att både det virtuella och det fysiska minne arbetar med 33-bitarsadresser, kan vi titta på översättningen av den virtuella adressen 7645000999 (decimalt). 7645000999 decimalt = 111000111101011010111010100100111 binärt, som består av sidadressen 11100011110101101101110 binärt (3732910 decimalt) och offset 10100100111 binärt (1319 decimalt).

Vi antar att sidan 3732910 är lagrad i frame 65536 decimalt (00000100000000000000 binärt). Uppslagningen i sidtabellen ger alltså framenummer 65536 decimalt (00000100000000000000 binärt), och vi kombinerar framenummer 00000100000000000000 binärt med offset 10100100111 binärt till den fysiska adressen 00000100000000000000010100100111 binärt = 134219047 decimalt.

Uppgift 9 (3 p)

I Unix finns en datastruktur som kallas i-nod (på engelska "i-node" eller "inode"). Vad används den till, och vad innehåller den?

Svar:

I-noden innehåller data om en fil i filträdet. En fil kan vara en normal fil med data eller körbar kod, men den kan också vara en filkatalog ("directory") eller en "speciell" fil ("special file"). Speciella filer kan representera en fysisk enhet (exempelvis en hårddiskenhet), eller fungera som ett gränssnitt mot operativsystemet (exempelvis **/proc**-filsystemet).

I-noden innehåller data om filen, som vilken användare som är filens ägare, vilka rättigheter (att läsa, skriva och exekvera filen) olika användare har, filens storlek, tid för senaste ändring, och referenser till var filens data är lagrade.

I-noden innehåller *inte* filens namn. Filnamn lagras i filkatalogerna.

Uppgift 10 (4 p)

När man ska lagra de data som hör till en process kan det vara svårt att hitta ett tillräckligt stort, sammanhängande utrymme i det fysiska primärminnet. Hur löser man det problemet på vanliga moderna datorer och med vanliga moderna operativsystem som Linux och Windows?

Svar:

Man allokerar inte sammanhängande fysiskt minne, utan delar in den virtuella minnesrymden i lika stora "sidor" (på engelska "pages"), exempelvis med storleken fyra kilobyte. Det fysiska minnet delas in i "frames" (på svenska kanske "ramar"), med samma storlek som sidorna, och en sida kan sedan placeras i vilken ledig frame som helst. Processens sidor behöver inte lagras sammanhängande eller i ordning. "Sidtabellen" (på engelska "page table") översätter mellan sidnummer och ramnummer, dvs mellan virtuella och fysiska adresser.

Thomas Padron-McCarthy (thomas.padron-mccarthy@oru.se), 12 juni 2019