Lösningsförslag till tentamen i ID2206/ID2200/IS1350 Operativsystem
(fd 2G1520/2G1504/6B2019)

Måndagen 2009-10-19 kl 0900-1400

Institution: ECS/ICT

Examinator: Robert Rönngren
Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.
Ange på omslaget vilken kurs du tenterar och vilken termin du läste kursen.

Varje inlämnat blad skall förses med följande information:
- Namn och personnummer
- Nummer för behandlade uppgifter
- Sidnummer

ID2200 (fd 2G1504)
- Studenter som tenterar ID2200 besvarar inte frågor markerade med *****

IS1350 (fd 6B2019)
- Studenter som tenterar IS1350 besvarar inte frågor markerade med **¤**

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng
- Resultat beräknas finnas rapporterat i LADOK och tentorna tillgängliga via studentcentrum senast tis 10/11

Betygsgränser:
- Godkänt (E) garanteras från 50% av den maximala poängen för respektive kurs

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 22/10)

Frågor: (som normalt bara kräver kortfattade, men motiverade, svar (1-10 rader))

1) Förklara hur ett systemanrop går till på assemblernivå. (1p)

Parametrarna till systemanropet läggs på platser som gör att operativsystemet kan hitta dem, t.ex i register. Koden för det systemanrop som skall utföras läggs också på en förutbestämd plats. Återigen antagligen i ett register. En TRAP-instruktion exekveras och kontrollen övergår till operativsystemet. I OS:et hanteras TRAP av en sk. TRAP-handler som börjar med att kontrollera varför TRAP-instruktionen exekverats - detta görs genom att kontrollera koden för systemanropet. När man vet att det är ett systemanrop och vilket anrop det gäller anropas koden för systemanropet i kärnan. När systemanropet genomförts återgår kontrollen till användarkoden efter TRAP-instruktionen.

2) Systemanropen för att skriva/läsa till/från en fil i POSIX-standarden (`write()` och `read()`) är tämligen enkla. Kalle tycker att man istället borde införa systemanrop som liknar `printf()/scanf()` istället. Diskutera om detta är en bra eller dålig ide! (1p)

Det är en dålig ide av flera anledningar. printf()/scanf() är inte bra på att hantera läsning/skrivning av binära filer utan är avsedda att hantera formatterad text. Dvs de kan sägas implementera en policy för vilken typ av filer de arbetar på. Ett OS bör tillhandahålla de enklaste primitiverna för att göra något - vill användaren ha annan funktionalitet byggs det bäst ovanpå bas-funktionaliteten. I det avseendet tillhandahåller read()/write() precis den funktionalitet man vill ha.

3) Kan en användarprocess göra I/O utan att göra ett systemanrop? (1p)

Normalt sett är svaret nej. I/O-enheter hanteras praktiskt taget alltid av device-drivers som exekverar i kärnan.

4) Hur implementeras blockerande respektive icke-blockerande skrivning mot en I/O enhet i ett operativsystem? (1p)

Vid icke-blockerande skrivning sker skrivningen till en buffert någonstans i kärnan - OS:et skriver vid lämpligt tillfälle ut innehållet i bufferten på I/O-enheten. Processen behöver alltså inte blockeras i väntan på att I/O-

enheten skall bli tillgänglig och klar med överföringen. Vid blockerande skrivning så tvingas användarprocessen vänta på att skrivningen till I/O-enheten skall bli färdig.

5) I de flesta OS med schemaläggning baserad på prioriteter sätter användaren en grundprioritet för processen. Ändå kan samma process ibland exekvera med annan prioritet. Förklara i vilka fall detta kan förekomma och varför det är så! (2p)

Detta kan ske t.ex då:
- en process får höjd/eller sänkt prioritet till följd av aging
- en process utför ett uppdrag åt en annan, högre prioriterad, process och tillfälligt lånar dennas prioritet sk. "priority inheritance"
- i många OS finns andra exempel som t.ex i Microsofts OS där em process kan få tillfälligt höjd prioritet då I/O den väntat på är klar

6) a) Antag att vi startar en process i ett system med sidindelat virtuellminne. När laddas normalt sett processens sidor i minnet? (1p)
b) Vad betyder det att man får ett sidfel? (1p)
c) Beskriv vad som händer då man åtgärdar ett sidfel! (1p)

a) När sidorna accessas första gången
b) Att en sida inte finns i primärminnet eller att man försökt göra en otillåten operation på sidan (t.ex skriva till en skrivskyddad sida)
c) När ett sidfel, som beror på att sidan inte fans i primärminet åtgärdas så görs följande steg:
- man får ett trap till operativsystemet
- OS:et hittar en ledig ram att lägga sidan i (vid behov kan det innebära att innehållet i en upptagen ram lyfts ut till disk)
- OS:et tar reda på var på hårddisken sidan finns och startar en DMA överföring till en plats i primärminnet
- När DMA överföringen är klar så mappas sidan in i processens adressrymd, processens sidtabell uppdateras för att visa att sidan nu finns i minnet och processen läggs i READY-kön

7) a) Beskriv hur en TLB används! (1p)
b) Många system har olika TLB:er för data respektive instruktioner. Varför? (1p)

a) en TLB (Translation Lookaside Buffer) fungerar som en cache för översättningar från <process och sidnummer> till ramnummer. Vid en adressöversättyning söker MMU:n i TLB:n för att se om det finns en översättning från sidnumret till ramnummer för den aktuella processen. Vid träff sker då adressöversättningen mycket snabbare än om man måste accessa sidtabellen. Vid miss i TLB:n sker en uppslagning i siodtabellen och TLB:n uppdateras
b) Därför att instruktioner och data ofta uppvisar olika lokalitet. Det betyder att instruktioner kan ha bra lokalitet och data dålig eller tvärt om. Om man använde samma TLB för översättningar för sidor innehållande instruktioner och data skulle då dålig lokalitet hos den ena typen kunna göra att ofta använda översättningar för den andra typen skulle slås ut ur TLB:n. Genom att ha olika TLB:er för data resp. instruktioner kan man därför totalt sett få bättre prestanda även om den ena typen uppvisar dålig lokalitet.

8) a) På vilka nivåer i ett datorsystem kan man implementera trådar? (0.5p)
b) Vilka för och nackdelar har det att implementera trådar på de olika nivåerna? (2p)

a) I användarläge eller i kärnan
b) Att implementera trådar i användarläge har den fördelen att det blir helt och hållet portablet eftersom det inte kräver någon extra funktionalitet av OS:et än de (standardiserade) systemanrop OS har. Nackdelarna är att schemaläggningen av trådarna ofta måste skötas explicit, dvs användaren måste i sitt program anropa funktioner då man skall kunna byta melan trådar och att OS:et inte känner till trådarna och därför t.ex inte kan schemalägga en annan tråd för exekvering inom processen om en tråd blockeras i ett systemanrop.
Att implementera trådar i kärnan har nackdelen att detta inte nödvändigtvis blir portabelt om inte systemanropen för trådarna är standardiserade. Fördelen är att schemaläggning av trådarna sker av kärnan och att man ofta kan utnyttja trådbegreppet för att göra kärnan i sig effektivare.

9) a) Vilka för - och nackdelar har det att basera ett filsystem på blockindelad respektive kontinuerlig allokering? (1p)
b) När man i Microsofts (mfl) filsystem gör en "defragmentering" av filsystemet vad är det då som händer och vad är det i implementationen av filsystemen som ger upphov till att de kan behöva "defragmenteras"? (2p)

a) Fördelarna med kontinuerlig allokering är att det kan ge snabbare access till filen eftersom läs/skrivhuvud et på disken inte behöver flyttas mellan olika delar av disken. Nackdelen är att man ofta får problem med extern fragmentering i system där filer kan ändra storlek. Det lämpar sig därför bäst för system som bara skrivs en gång. Blockindelad allokering löser problemen med extern fragmentering men ger å andra sidan upphov till intern fragmentering. Det senare är oftast inte något stort problem om inte blockstorleken är stor i förhållande till diskstorleken. Nackdelen är att en fil kan komma att spridas ut på olika platser på hårddisken och att de datastrukturer som krävs för att hålla ordning på i vilka block blir mer komplicerade än för fallet med kontinuerlig allokering.
b) Många filsystem försöker kombinera ideer från både blockindelad och kontinuerlig allokering. De är i grunden blockindelade men försöker allokera blocken för en fil så att de ligger kontinuerligt på disken för att öka prestandan vid läsning/skrivning. Det gör att man kan få liknande problem som vid kontinuerlig allokering om filer ändrar storlek. Den "defragmentering" som sker är att filernas block omallokeras så att filerna ligger i kontinuerliga sekvenser av block på disken.

10) a) Hur implementeras ett directory i UFS (UNIX File System)? (1p)
b) Vad kan man förvänta sig att hitta i ett UFS-directory för en vanlig fil i directoryt? (1p)
b) Varför använder man oftast en speciell funktion (`getdents()` eller liknande) när man läser innehållet i ett directory? (1p)

a) som en fil med en speciell markering i i-noden att det är ett directory
b) en översättning från filnamn till inodsnummer <file name, i-node number>
c) eftersom man kan ha olika typer av filsystem monterade så kan den interna implementationen av vad som finns i ett directory också skilja. detta och att man för att göra det enklare att tolka innehållet i directoryt vill kunna få det inläst och presenterat på ett filsystemsoberoende format gör att man ofta har speciella funktioner för att läsa och tolka innehållet i ett directory

11) I moderna operativsystem kan man ofta bestämma om olika delar av systemet skall kompileras och länkas in direkt i kärnan eller om de skall läggas som moduler som laddas in vid behov. Vilka är fördelarna och nackdelarna med de olika sätten att konfigurera kärnan på! (2p)

Att kompilera och länka in moduler in i kärnan har fördelarna att: de kan accessas snabbare och att man ev. kan låsa kärnan så att inga nya modler kan läggas till vilket kan vara bra av säkerhetsskäl - nackdlen är naturligtvis att det blir svårare att lägga till funktionalitet eftersom det kräver att hela kärnan byggs om och att kärnan kan bli onödigt stor. Att lägga funktionalitet i moduler gör att kärnan i sig blir mindre, att man får mer flexibilitet i hur kränan konfigureras men att säkerheten också kan bli lägre då modifierad kod kan laddas in vid behov. Första anropet till en modul kan också ta längre tid än om den varit inlämkad direkt i kärnan.

12) Vad händer när man anropar en funktion (metod) i ett dynamiskt länkat bibliotek för första gången! Beskriv hur detta implementeras i den kod som från början länkats in för funktionen/metoden (1p)

Funktionen anropas inte direkt utan via kod som brukar kallas "stub-kod". Den gör ofta anropet via en funktionspekare. Innan funktiopnen anropats första gången pekar den här pekaren inte på något giltigt utan på t.ex NULL Närett anrop sker kontrolleras om pekaren är satt att peka på den metoden man vill anropa - första gången är den alltså inte satt utan man anropar kod som letar rätt på funktionen (som antingen läses in från disk eller som kanske används av någon annan process och finns i minnet) och sätter funltionspekaren att peka på rätt funktion som sedan anropas.

13) Givet följande referenssträng, 1 5 0 6 5 4 2 6 7 3 0 4 1 7 6 , fyra ramar och ren "demand paging", beräkna antalet sidfel för FIFO, LRU resp. OPT. (1.5p)

Räkningarna skall redovisas för att få poäng. FIFO: 13 sidfel, LRU: 13 sidfel, OPT: 10 sidfel

14) a) Vad innebär begreppet lokalitet? Och vilka typer av lokalitet finns! (1p)
b) Hur påverkas olika delar av ett modernt datorsystem som kör ett operativsystem med virtuellt sidindelat minne av att en applikation har dålig lokalitet? (2p)

a) Lokalitet innebär att en delmängd av minnesrymden accessas. Man brukar skilja på rumslokalitet som innebär att det är en begränsad delmängd av minnesrymnden som accessas; resp. Tidslokalitet som innebär att accesserna är begränsade till att ske under någon viss tid.
b) Om inte både tids och rumslokalitet är bra så påverkas flera delar av ett modernt datorsystem. Då får man dålig prestanda i alla typer av cachar - både för data och instruktioner men även för t.ex TLB. Eftersom ett virtuellt sidindelat system använder sig av lokalitet fungerar det dåligt då man kan få: många TLB missar vilket normalt dubblerar minnesaccesstiden. Har man mycket dålig lokalitet ger det också många sidfel vilka kan ta väldigt lång

tid att hantera och som i värsta fall leder till sk. "thrashing" då mer tid spenderas på att hantera sidfel än til att exekvera processerna.

15) a) Var i den virtuella minnet hamnar variablerna `stina` respektive `lisa`? Förklara varför! (1p)
b) Var i den virtuella minnesrymden hamnar funktionen `oskar` och vad betyder det att den är `static`-deklarerad? (1p)

```
static int oscar(const int stina)
{
    static int lisa = 4;
    return lisa += stina;
}
```

stina: är en lokal parameter och läggs upp i aktiveringsposten på stacken varje gång funktionen anropas
lisa: är en lokal variabel som finn i en instans i dataarean och alltså behåller sitt värde mellan olika anrop av funktionen. den kan bara accessas inifrån funktionen oskar
oskar: ligger i textarean. att den är deklarerad static betyder att dess namn inte syns utanför den källkodsmodul där den är definierad

16) Kalle tänker implementera ett nytt och förhoppningsvis säkrare programmeringsspråk för att skriva operativsystem. Det skall bygga på C men inte ha pekare till funktioner. Är det här en bra eller dålig ide? Förklara varför? (2p)

Det är en dålig ide eftersom många delar i ett modernt operativsystem bygger på att funktioner kan anropas indirekt via funktionspekare. Det sker t.ex i hantering av avbrott, TRAP, för device-drivers och när moduler laddas i kärnan under drift

17) I många operativsystem kan drivrutiner och avbrottsrutiner vara "stackade". Vad menas med detta och vilka fördelar har det? (2p)

Det innebär att man kan ha en följd av funktioner som skall anropas när ett avbrott skall hanteras eller så man anropar en drivrutin. Fördelen är att man kan återanvända och bygga ihop funktionalitet t.ex för en drivrutin genom att ha en funktion som kan hantera dataöverföring på en buss, en annan som kanske tar hand om blockstorlek etc. Men också att man t.ex kan lägga till och utöka funktionalitet som t.ex att man genom att lägga till en funktion som körs först när ett TRAP sker kan kontrollera vilka systemanrop som sker och t.ex kunna lägga till begränsningar på vilka program en användare får exekvera.

18) a) Vilka är Coffmans fyra villkor för deadlock? (1p)
b) Om alla Coffmans villkor är uppfyllda i ett system vad betyder det för systemet? (1p)

a) Korta förklaringar till villkoren krävs

cirkulär väntan: att samtliga processer i den mängd processer vi studerar (som är inblandade i ett deadlock) väntar på en annan process i mängden
hålla och vänta: att en process kan ha allokerat resurser medan den väntar på at få allokera andra resurser
ingen pre-emption: att vi inte kan ta resurser från en process som allokerat dem
ömsesidig uteslutning: att en resurs bara kan ägas/användas av en process i taget

b) om det bara finns en resurs av varje resurstyp befinner sig systemet i ett deadlock. annars är villkoren nödvändiga men inte tillräckliga vilket betyder at om det finns mer än en resurs av varje resurstyp kan systemet befinna sig i deadlock men behöver inte vara det - dvs att villkoren är uppfyllda säger inte att systemet är i deadlock

19) Varför bör man om möjligt använda "analytiska metoder" snarare än vanlig debuggning när man letar efter synkroniseringsproblem? (1p)

synkroniseringsproblem är ofta beroende av att händelser i systemet schemaläggs i en viss ordning för att synkrioniseringsproblemen skall inträffa. därför kan användandet av debuggers och liknande påverka ett system med synkroniseringsproblem så att problemen inte uppstår när man debuggar systemet. det kan också vara så att det generellt sett är svårt att återskapa ett synkroniseringsproblem. att förstå problemet analytiskt är därför ofta en bättre lösning än att använda hjälpmedel för att se vad som hänt/händer

20) Var i ett operativsystem kan man anta att ett "spionprogram" försöker installera sig? Varför? (1p)
(ett spionprogram är ett program som vill kunna övervaka allt som användarna gör i systemet)

ett spionprogram som ska kunna övervaka allt användarna gör i ett system försöker se till att det tar kontrollen över "TRAP-handlern" dvs att det kommer att köras först då ett TRAP skall hanteras. detta eftersom inget intressant kan göras av en användare i ett system annat än via systemanrop

21) Varför påverkas oftast mjuka realtidsprocesser inte av aging-mekanismer? (1p)

Om en mjuk realtidsprocess fick sin prioritet sänkt pga en aging-mekanism för att den exekverats för mycket/ofta eller allmänt använt för mycket resurser i systemet skulle den inte få bra realtidsegenskaper

"Vad många av oss mest av allt behöver är någon som får oss att göra vad vi kan."
R. W. Emerson