Lösningsförslag (så kortfattade att de inte alltid ger fullpoäng) till Tentamen i ID2206/ID2200/IS1350 Operativsystem
(fd 2G1520/2G1504/6B2019)

Måndagen 2007-12-17 kl 1400-1900

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 mån 21/1

Betygsgränser:
- Totalt antal poäng på tentamen är ID2206 31p, ID2200 28p, IS1350 28p
- Godkänt (E) garanteras från 50% av den maximala poängen för respektive kurs
- Möjlighet till komplettering vid underkänt resultat på tentamen tillämpas inte

---

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 18/12)

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

1) Vad används "heapen" respektive "stacken" till för ett C-program? (2p)

Heap: är ett datautrymme (i den virtuella dressrymden) som används för att allokera minne för dynamiska datastrukturer/objekt etc. under körningens gång, dvs under run-time
Stack: används för att lägga upp aktiveringsposter på vid anrop av funktioner/procedurer. En aktiveringspost innehåller: parametrar, lokala variabler, returadress samt ofta en pekare till nästa aktiveringspost (frame pointer)

2) ¤ Kalle skall implementera en kommandotolk (ett "shell") för LINUX. Han hävdar att det är bättre (mer effektivt) att skapa kärntrådar för att exekvera kommandona i än att skapa nya processer att exekvera kommandona i. Red ut (förklara) eventuella för- och nackdelar med Kalles idé! (1p)

Kalles ide fungerar inte. Ett kommando måste exekveras i en ny process - om inte skulle man ersätta kommandotolken med det program som hör till kommandot och alltså bara kunna köra ett enda program.....

3) Processschemaläggning handlar i grunden om att kunna flytta processer mellan tre grundtillstånd. Rita en bild med dessa tillstånd och beskriv övergångarna mellan de olika tillstånden. (2p)

Se Modern Operating Systems (Tannenbaum) 2.1.5 Process states och "figure 2.2"

4) Antag att en minnesaccess tar M tidsenheter och att det tar D tidsenheter att läsa in en sida från disk till en ledig ram (D >> M). Hur lång blir den genomsnittsliga (upplevda) minnesaccesstiden för en process om den för sina minnesaccesser genererar:
   i) 50% TLB missar och sidtabellen implementeras som en en-nivås sidtabell (0.5p)
   ii) 50% TLB missar och sidtabellen är en två-nivåers sidtabell (0.5p)
   iii) 10% sidfel där "victim-pages" är modifierade (0.5p)

Söktiden i TLB:n kan antas vara liten jämfört med minnesaccesstiden och kan försummas. Vi tar ändå med den i räkningarna som T.
i) Pträff(T+M) + (1-Pträff)(M+M) = 0.5(T+3M) = (vi försummar T) 1.5M

ii) Pträff(T+M) + (1-Pträff)(M+2M) = 2M
iii) Vid sidfel där den sida som ersätts i minnet är modifierad måste denna först skrivas till disk. Vi kan anta att tiden för att läsa eller skriva en sida från/till disk är densamma) Vi kan försumma både TLB och sidtabellsaccesser om D>>M.
0.9*M + 0.1*(M + antal_nivåer_i_sidtabell*M + D + D) = (försumma sidtabellsaccesser) M+0.2D

5) Vilken typ av accesskontroller sker normalt sett per sida i ett system med sidindelat minne? (1p)

Accesskontrollerna är normalt: i) får processen accessa sidan över huvud taget - om så ii) får man exekvera, skriva och/eller läsa innehållet i sidan.

6) Vilka sidor kommer att bytas ut om vi får två direkt på varandra följande sidfel och vi använder sidutbytes-algoritmerna FIFO, Second Chance, LRU resp. OPT (2p)

SidNr	Laddtid	Refererad tid	R	M
1	120	356	1	0
2	80	412	1	1
3	253	270	0	1
4	170	298	1	0
5	280	300	1	1
6	98	150	1	0

FIFO: Byt ut den sida som laddades in först: dvs sidorna 2 och 6
Second Chance: Gå igenom sidorna efter laddtid - om R biten är nollställd byt ut sidan annars nollställ R biten och sök vidare (cirkulärt): sida 3 och 2
LRU: Byt ut den sida som var längst sedan den refererades: sida 6 och 3
OPT: kräver information om alla framtida sidreferenser - dvs om man antar att det kommer fler referenser i framtiden än de två som nämns i texten går det inte att svara på vilka sidor som byts - om man antar att det inte kommer fler referenser än de två som nämns kan man byta två godtyckliga sidor

7) Vilken information hittar man i ett normalt "entry" (post) i ett directory i ett UNIX-filsystem? (Antag att posten är en hård länk.) (1p)

<filnamn, i-nodsnummer>

8) För många typer av filsystem som finns på moderna persondatorer finns möjligheten att "defragmentera" filsystemen. Om man "defragmenterar" hårddisken/filsystemet kan man i vissa fall få väsentligt bättre prestanda från filsystemet. Frågan är bara vad som egentligen sker? Och handlar det om defragmentering i egentlig mening eller är det något annat som sker? Beskriv vad som antagligen sker och varför prestanda förbättras! (2p)

Moderna filsystem använder nästan uteslutande blockbaserad allokering för filerna. Dvs den enda typ av fragmentering som finns är intern fragmentering. Intern fragmentering kan man knappast påverka med ett defragmenteringsprogram. Däremot kan man flytta filerna på disken så att blocken ligger "närmare" varandra och inte så spridda på disken, t.ex så att de hamnar på samma spår. Har man ett filsystem med i-noder eller liknande lösning kan man dessutom försöka lägga filerna så att de ligger i närheten av dessa, dvs på samma spår eller i samma cylinder. Den prestandaförbättring man kan åstadkomma beror på att man på så sätt kan minska diskarmsrörelserna och ev. kan möjliggöra read-ahead.

9) Vilka av diskarmsschemaläggningsalgoritmerna First-Come-First-Serve, Shortest Seektime First respektive SCAN fungerar bra att använda i praktiken? (1p)

FSFC schemalägger inte diskarmsrörelserna så att man optimerar dessa - utöver detta så kan den användas. SSF är inte lämplig att använda pga svältningsrisk. SCAN är en bra avvägning av bra diskarmsschemaläggning och avsaknad av svältningsrisk. Dock kräver den mer "intelligens" av disk-controllern och är alltså lite mer komplicerad att implementera

10) Beskriv hur I/O med pollning går till! (1p)

Vi antar att man styr I/O enheten via ett data- och ett kontrollregister.

i) loppa och läs av kontrollregistret till dess att man ser att dataregistret är ledigt.
ii) om det handlar om skrivning - skriv in data som skall skrivas till I/O enheten i dataregistret
iii) skriv in koden (dvs sätt bitarna) i kontrollregistret för den operation som skall utföras
iv) loopa och läs av kontrollregistret till dess att det signalerar (dvs en eller flera bitar i registret indikerar) att operationen utförts och enheten är redo att utföra nästa operation
v) om det handlade om läsning - läsa av, kopiera, data från dataregistret
vi) vid behov upprepa från i)

11) En drivrutin som använder DMA för överföring av data till/från primärminnet består nästan alltid av två delar. Beskriv vilka delarna är och vad de gör! (2p)

Vid DMA så instrueras DMA att föra över data till/från primärminnet och en I/O-enhet. När operationen är klar genererar DMA enheten ett avbrott. En drivrutin kan därför logiskt sett delas in i två delar, i UNIX kallas dessa för top-half respektive bottom-half device driver. Top-half delen är den del som OS:et anropar för att få en operation utförd, den sätter upp DMA och normalt blockeras anropande process i väntan på att DMA skall bli klar. Bottom-half delen, dvs avbrottsrutinen, körs när avbrottet som signalerar att DMA-överföringen är klar kommer. Den ansvarar för att ta hand om ev. data som behöver tas om hand om och för att se till att den process som eventuellt väntade på att I/O-operationen skulle bli klar kommer att läggas i ready-kön så att den kan fortsätta exekvera.

12) Varför går det oftast snabbare att byta (göra context-switch) mellan två kärntrådar än att byta mellan två processer? (1p)

Det beror på att man vid ett byte mellan två kärntrådar endast byter ut registeruppsättningen i den process som exkverar, dvs programräknare, stackpekare och generella register. Vid ett processbyte är den context som skall bytas mycket större och inkluderar även sidtabell, filhantering och ev statistik/quotas etc.

13) Sockets används i de flesta system för client-server kommunikation över nätverk.
i) Vilken information måste en klient känna till för att kunna kommunicera med en server över Internet? (1.5p)
ii) Kalle har skrivit en Client-server applikation som kommunicerar via socket där klienten skall skicka ett heltal till servern. Koden i klienten respektive servern återfinns nedan. Kalle har problem då heltalet som tas emot på servern inte är det samma som klienten skickade - vad kan problemen bero på? (1p)

Client	Server

`long tal;`
`....`
`send(socket, (void *) & tal, sizeof(long), 0);`
	`long num;`
	`....`
	`recv(socket, (void *) & num, sizeof(long), 0);`

i) Serverns IP-adress och portnummer, samt protokollet som man använder för kommunikationen (dvs vad servern förväntar sig/kan tolka som vi skickar till den och hur svaren ser ut) samt hur data skall föras över, dvs vilket format data skall packas på under överföringen (motsv. marshalling)
ii) klient och server har antagligen olika representation av datatypen long - t.ex olika format vad avser little/big endian och eller olika längd (antal bitar)

14) Synkroniseringsproblem finns inte bara i datorsystem utan också i vår omgivning. Förklara och visa varför deadlock och svältning kan eller inte kan uppstå i en trafikkorsning mellan två vägar med:
i) Högerregeln (1p)
ii) Stopplikt på alla fyra infarter till korsningen (1p)

i) Högerregel: om det kommer fram fordon samtidigt på alla infarter till korsningen och att dessa t.ex vill köra rakt igenom korsningen så kommer dessa att vänta cirkulärt på varandra, dvs på att någon annan skall köra först (uppfyller definitionen på ett deadlock - en mängd processer där samtliga processer i mängden väntar på att någon (annan) process i mjängden skall göra något innan den kan fortsätta). Om vi t.ex antar att det hela tiden kommer en ström av bilar från öster som kör mot väster så kommer bilar som kommer från söder och vill köra norrut att tvingas vänta, dvs dessa kan svältas ut. Dvs både svältning och deadlock kan uppstå.
ii) Om man antar att två bilar inte kan komma fram exakt samtidigt till korsningen så kommer det att finnas en strikt ordning för samtliga fordon i vilken de får köra genom korsningen, dvs vi bryter villkoret med cirkulär väntan. Pss sätt vet man att om det finns N fordon i kön före ett fordon X på en infart, kan X maximalt få vänta på att 4N fordon skall köra genom korsningen inna det självt får köra genom, dvs väntan är begränsad och det föreligger ingen risk för svältning. Dvs bevisbart fritt från svältning och deadlock

15) Du har haft virus på din dator. Vilka av följande filtyper kan du kopiera tillbaka från din senaste backup och använda utan risk?
i) `?.c` , ii) `?.ps` , iii) `?.o`, iv) `?.doc` (1p)

.c C-kod kan innehålla källkodsvirus, som kan åstadkomma skada den dag koden kompileras och exekveras, .ps PostScript är ett språk som interpreteras eller exekveras - dvs kan innehålla virus, .o kompilerad men ännu ej länkad kod kan innehålla virus som kan åstadkomma skada den dag .o filen länkas och exekveras, .doc Word-fil kan innehålla macro-virus. Dvs ingen av filtyperna kan kopieras tillbaka utan risk för att de är virussmittade.

16) Varför kan det ge bättre säkerhet att använda ett "integritetskontrollprogram" för att skydda sig mot virus än att använda en vanlig "virus scanner" (1p)

En scanner kan aldrig upptäcka alla virus och känner bara igen virussignaturer som finns i dess databas, dvs nya virus kan passera oupptäckta. Vid integritetskontroll försöker man säkerställa att filerna som skulle kunna infekteras av virus är intakta, dvs de integritetsskyddas så att man upptäcker om de smittats av virus (dvs förändrats) Detta skyddar mot alla typer av existerande och framtida virus. (problemet är att att samtliga filer som skulle kunna utgöra infartsvägar för ett virus måste skyddas.... och man måste kunna säkerställa att dessa inte redan var infekterade då de integritetsskyddas.....)

17) * ¤ Realtidsystem klassificeras ofta som hårda respektive mjuka. Vad är den främsta skillnaden mellan dessa två typer av realtidssystem? (1p)

I ett hårt realtidsystem måste samtliga deadlines hållas annars fungerar inte systemet som avsett - I ett mjukt system kan man i vissa fall tolerera att deadlines missas, dvs systemet fungerar korrekt, eller tillräckligt bra ändå.

18) * ¤ Behöver man i ett hårt realtidsystem för bildbehandling ta hänsyn till om man manipulerar bilderna radvis eller kolumnvis? (1p)

I ett system med sidindelat minne kan det vara helt avgörande för prestanda att man accessar stora matriser (bilder) på så sätt att man undviker att generera onödiga sidfel. I ett hårt realtidssystem använder man inte sidindelat minne - det enda man i detta fall behöver ta hänsyn till är om det finns eventuella data-cachear och hur man i så fall minskar antalet cache-missar.

19) * I både UNIX, Linux och Microsofts OS implementerar man gränsnittet som OS-kärnan använder för att anropa funktioner i devicedrivern på liknande sätt. Beskriv hur det implementeras i C! (1p)

Gränssnittet består av en struktur (struct i C) som innehåller pekare till funktioner för open(), read(), write(), close(), ev. seek() och motsvarande funktioner för systemanrop som man förknippar med manipulering av filer.

20) I UNIX får alla processer en lika stor time-slice för sin exekvering - men hur är motsvarigheten till time-slice implementerad i LINUX? (2p)

I Linux får varje process en individuell time-slice som kallas "quantum". quantum beräknas varje gång det inte längre finns några körbara processer att exekvera, dvs då alla processer antingen är blockerade (dvs de har kvar tid på sitt förra quantum) eller har förbrukat sitt nuvarande quantum. Det nya quantum beräknas då som

nytt_quantum = kvarvarande_gammalt_quantum/2 + prioritet

processer som ofta gör I/O prioriteras alltså genom att de dels får längre quantum och de får också lite högre faktisk prioritet, sk. godness, som är den prioritet processer schemaläggs efter. Beräkningstunga processer får å andra sidan tillgång till CPUn (dvs tidskvanta) direkt proportionellt mot grundprioriteten

21) Från och med Microsofts operativsystem NT/WINDOWS 2000 så använder man sig av "dynamisk länkning", något som t.ex. syns i namnen på t.ex biblioteket för systemanrop "`ntdll.dll`". Beskriv vad dynamisk länkning är och vad man kan tjäna på att använda det. (2p)

Kod för funktioner som inte finns i källkoden utan ligger i t.ex bibliotek länkas inte in vid kompilering/länkning av programmet, istället läggs sk. stub-funktioner in. Dessa har till uppgift att första gången funktionen anropas se till att rätt kod görs tillgänglig. Detta sker genom att OS:et kontrollerar om någon annan process redan använder funktionen i fråga och har en kopia i minnet av den - i det fallet delas koden mellan processerna (detta är normalt inget problem om koden är read-only, vilket den nästan alltid är) - om inte länkas en egen kopia funktionen in och laddas in i minnet. Det man vinner på dynamisk länkning är att laddmodulen (dvs den exekverbara modulen) blir mindre - vilket kan vara av stor betydelse för program som är stora och komplexa med mycket funktionalitet - men

där bara en mindre del av funktionaliteten i praktiken används. Man sparar också primärminne genom att inte behöva ladda in flera kopior av samma kod om flera processer använder samma funktioner.

"Det är detaljerna som är skillnaden mellan bra och utmärkt"
H.H.