Utkast till lösningar till Tentamen i 2G1520 Operativsystem tisdag 2005-12-20 kl 1500-1900

Institutionen för Mikroeletronik och Informationsteknik

Examinator: Robert Rönngren

Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.

Ange på omslaget vilken kurs du tenterar (2G1520 eller äldre kurs) och vilken termin du läste kursen.

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

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng
- Resultat beräknas anslås senast 060116

Betygsgränser:
- Totalt antal poäng på tentamen är 33
- Godkänt (3) garanteras från 17 poäng
- Student som ligger inom 1p från godkänt gränsen äger rätt att inom 14 dagar från det att tentamensresultaten anslagits begära muntlig komplettering som om nöjaktigt utförd kan höja betyget till G

Lösningsförslag: anslås på kursens webbsida dagen efter tentamen

---

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

1) a) Vad är en sk. zombie-process i UNIX och hur skapas en sådan? (1p)
   b) Varför kan man tvingas boota om en maskin om det skapas många zombie-processer (0.5p)

a) En zombie-process är en avslutad process som ännu inte städats bort ur OS:ets datastrukturer då dess förälder inte väntat på den på korrekt sätt
b) En zombie-process tar upp en plats i processtabellen och när den är full kan inte nya processer skapas och då tvingas man oftast starta om maskinen

2) Vad är respektive: relokerbar kod och reentrant kod? (2p)

Relokerbarkod = flyttbar kod - alla adresser i koden är angivna relativt kodstart - inga absolutadresser inom module anges
Reentrant kod - kod i vilken flera processer kan exekvera samtidigt - koden i sig får inte modifieras av processerna

3) Minneshanteringen i ett modernt operativsystem sker normalt sett på flera nivåer. En nivå (begrepp) som förekommer är swappning.
   a) Förklara begreppet! (1p)
   b) Vilka är de två (2) troligaste tillfällena då swappning förekommer? (1p)

Swapping - att flytta hela processer mellan disk-minne
Swapping förekommer då: SWAP-OUT: en process inte varit aktiv på länge utan t.ex varit blockerad i väntan på I/O en längre tid eller då fysiskt minne behöver frigöras om maksinen är överlastad och man t.ex thrashar SWAP-IN: när lasten minskar och man kan låta en utswappad process starta igen, alt. då I/O väntan är över

4) Antag att vi i ett virtuellminnessystem använder en två-nivåers sidtabell. Hur många minnesaccesser går då åt i snitt för att ladda värdet på en variabel, som finns i primärminnet, till ett register om vi:
   a) Inte har något speciellt stöd för virtuellminneshantering i datorarkitekturen (0.5p)
   b) Har en TLB med 90% träffsannolikhet (1p)

a) tre accesser: en i den yttre sidtabellen, en i den inre och en för data självt
b) 0.9*(bara acessa data) + 0.1(två minnesaccesser i sidtabell + data) = 0.9+0.3 = 1.2 accesser i snitt

5) Givet följande: Virtuell adressrymd 2^{48}, Fysisk adressrymd 2^{24}, sidstorlek 64 KB, 4 bytes storlek på en post i en (yttre eller inre) sidtabell och en genomsnittslig process använder 4MBminne fördelat på 1MB stack, 2MB data och 1MB text. Hur stor blir då den del av datastrukturerna för adressöversättning som måste hållas tillgängligt för processorn i minne om man har en två-nivåers sidtabell? (2p)

Normalt mappar man en inre sidtabell på en sida, då kan en inre sidtabell accessa 64K/4 sidor - 16K sidor eller $2^{14,}2^{16} = 2^{30}$ = 1 GB. Då krävs pga den virtuella minneslayouten en inre sidtabell för stacken och en för text+data samt en yttre sidtabell. Adressuppdelningen blir: 16 bitar offset, 14 bitar att adressera den inre sidtabellen, kvar blir 18 bitar att adressera den yttre sidtabellen som blir $2^{18,}2^2 = 2^{20}$ = 1MB stor. Svar: 1MB+128KB (Not: man kan minska minnesutrymmet genom att göra en annan uppdelning av adressen)

6) I Windows NT användes från början olika sidutbytesalgoritmer beroende på vilken processorarkitektur man körde på. På en-processors x86i arkitekturer användes Clock och på arbetstationer med Alpha processorer samt på multi-processors x86i arkitekturer användes FIFO. Vilka är de rimliga skälen till att man valt olika algoritmer på dessa olika arkitekturer? (1p)

En enprocessors x86i har normalt mindre minne än de andra arkitekturerna. En observation man kan göra är att sannolikheten att slänga ut "fel" sida om en sida väljs godtyckligt minskar med ökande primärminne/antal ramar. Clock är också en algoritm som är mer precis i sitt val av "victim page" och som även har en större overhead per sida/ram. Sammantaget för detta att det är rimligt att välja en mer kostsam men också mer precis algoritm för mindre primärminnen än vad som behövs för stora primärminnen.

7) Beskriv, stegvis, vad som görs i operativsystemet då en fil öppnas via `open()` om filen ligger i "workingdirectory" i UNIX! (2p)

Eftersom filen ligger i WD så är det rimligt att anta att informatione om i-nod för WD finns i OS:ets interna tabeller. Då söks directoryt igenom för att hitta rätt i-nod för filen och denna läses in från disk till OS:ets lista över i-noder för öppna filer. En post skapas också i tabellen över öppna filer vilken i sin tur pekar ut i-nods tabellen. I processen sätts första lediga post i dess fildescriptortabell att peka ut posten i tabellen över öppna filer (man bör rita en figur/RR)

8) Kalle hävdar att om man gör "rm" på en fil i LINUX så kan filens diskblock direkt återanvändas för att lagra en annan fil. Stämmer det? (1p)

rm räknar ned räknaren för hårda länkar till filen i dess i-nod, endast om den räknaren går ned till noll kan filens diskblock frigöras och återanvändas

9) När man gör konsistenskontroller på filsystem kan man göra "block konsistens" och/eller "filsystems konsistens" kontroller. Vad kontrollerar man i de två fallen och vilka typer av fel kan man upptäcka? (2p)

Block konsistens: ett block får bara förekomma en gång i antingen en fil, i listan med dåliga block eller i listan med fria block. Om ett block finns i fler än en av dessa så har man ett allvarligt fel i filsystemet - finns blocket både i en fil och i listan med dåliga block har man en fil som antagligen inte går att läsa, finns blocket i både dåliga block listan och i fri-listan bör blocket kontrolleras och föras till bara den ena listan beroende på om det är OK eller ej, finns blocket både i en fil och i fri-listan bör blocket tas bort från fri-listan.
Filsystemskonsistens: man kontrollerar att det finns så många refrenser till en fil från directoryna som räknaren i dess i-nod anger. Finns fler referenser än räknaren anger måste denna uppdateras pss om räknaren är högre än antalet referenser, finns inga referenser kan filen tas bort.

10) Vad har det för betydelse för den som implementerar Operativsystem om datorarkitekturen stödjer precisa avbrott eller inte? (1p)

Vid ett precist avbrott behöver inte mjukvaran reda ut hur tillståndet var vid avbrottet för den instruktion som avbröts, dvs man behöver inte reda ut om en instruktion t.ex utförts till hälften utan instruktionen kan enkelt återstartas (alt. vet man att den utförts i sin helhet)

11) Beskriv hur avbrottsstyrd I/O går till! (2p)

Vid avbrottstyrd I/O startar den process som begärt I/O upp I/O operationen och blockeras sedan. Medan processen är blockerad kan andra processer schemaläggas. När I/O enheten är klar genereras ett avbrott som behandlas i en avbrottshanterare. Normalt resulterar detta i att processen kan gå från tillståndet blockerad till tillståndet klar (ready) för att senare schemaläggas.

12) Vilka data/datastrukturer måste en tråd som exekverar inom en process ha egna privata kopior av och varför? (1p)

En tråd måste ha en egen stack för att kunna göra sina egna procedur/funktionsanrop och egna kopior av alla register inkluderat stackpekare och programräknare.

13) Om en tråd exekverar `fork()` kan man tänka sig två alternativ: i) den nya processen blir en exakt kopia av den gamla med lika många trådar; eller ii) den nya processen får en tråd. Vilka är för och nackdelarna med de två alternativen? Har det någon betydelse om trådarna är kärntrådar eller trådar på användarnivå? (2p)

Den nya processen kommer alltid att få samma antal användartrådar som föräldraprocessen eftersom OS:et inte är medvetet om dessa. Vad gäller kärntrådar kan man argumentera för att lika många kärntrådar skall skapas i barnprocessen eftersom fork() normalt skall skapa en exakt kopia av föräldern, men om man antar att den nya processen med hög sannolikhet kommer att byta program så skulle man kunna argumentera för att man bara behöver skapa en kärntråd för den nya processen (dock kan det bli problematiskt om den ya processen skall fortsätta exekvera samma program som föräldern....)

14) Om man vill bygga feltoleranta system har det då någon betydelse om man väljer att implementera dessa baserat på processer eller trådar? (1p)

Processer är den minsta enheten i vilken man försöker kapsla in fel - därför är det troligt att två rpocesser inte direkt påverkar varandra vid fel medan två trådar i samma process mycket väl kan påverka varandra om den ena får fel och därmed påverkar deras gemensamma omgivning i form av processen. Processer har alltså egenskaper som är att föredra.

15) Beskriv hur man kan implementera UNIX-semantik i ett distribuerat filsystem! (1p)

Genom att tillåta flera att läsa filen samtidigt men bara tillåta en enda användare/process i taget att uppdatera filen. Jfr readers-and-writers problematiken

16) Beskriv de villkor som måste vara uppfyllda för att baklås (deadlock) skall kunna inträffa! (2p)

Coffmans villkor - se boken eller föreläsningsanteckningarna

17) Beskriv det vanligaste sättet att skapa en digital signatur som vem som helst kan verifiera men som är mycket svår att förfalska! (2p)

Man använder ett private-public key krypto där E(D(x)) == D(E(x)). Den som skall signera dokumentet skapar ett digitalt hashvärde för dokumentet. Detta hashvärde måste vara sådant att det är högst osanolikt att samma hashvärde kan genereras även om dokumentet modifieras. Hashvärdet krypteras sedan med den som skall signera dokumentets privata nyckel. För att verifiera signaturen kan en läsare beräkna hashvärdet för dokumentet utan signaturen och sedan dekryptera signaturen och se att det hashvärde som då får överrensstämmer med det beräknade.

18) Vad gör det enklare att implementera ett drivrutinsvirus än ett boot-sektor virus? (1p)

Ett boot-sektor virus måste normalt ta och behålla kontrollen över avbrottsvektorer etc. under det att OS:et bootas (dvs då OS:et modifierar avbrottsvektorern) medan ett drivrutinvirus laddas in då OS:et redan bootat och kan därmed enklare ta kontroll över avbrottsvektor etc.

19) I realtidsystem kan man få problem med prioritetsinversion. Vad är det och hur brukar man åtgärda problemen? (2p)

Om en högprioriterad process begär att en lågprioriterad process skall utföra något för dess räkning kan den lågprioriterade processen, och därmed den högprioriterade, få vänta på en process med mellanhögprioritet. Detta löses normalt genom att den lågprioriterade får ärva den högprioriterade processens prioritet då den utför arbete åt denna.

20) Kalle skall implementera hanteringen av drivrutiner för block-devices i NetBSD (Unix variant) han behöver hjälp med förslag på lämpliga datastrukturer för att implementera drivrutinsvektorn. Visa/deklarera en lämplig datastruktur i C som kan användas som klarar de 4 viktigaste operationerna på ett block-device. Beskriv också hur (med vad) Kalle skall indexera drivrutinsvektorn! (2p)

```
struct device {
      int (*open)(char*, int);
      int (*read)(int, char*, int);
      int (*write)(int, char*, int);
      int (*close)(int);
};

struct device deviceDrivers[MAXDEVICES];
```

Indexeras med major device number

21) Beskriv den händelseutveckling som var avgörande för att LINUX skulle slå igenom på bred front och inte de fria versionerna av UNIX! (1p)

UC Berkley släppte, då deras forskningsanslag tog slut, sins UNIx distrubutioner fria under namnet Berkley System Dsitribution (BSD). AT&T vidtog då legala åtgärder gentemot University of California för intång på upphovsrätten eftersom man ansåg sig äga rättigheterna till grunderna i UNIX. Samtidigt dök LINUX upp och där kunde inga sådan krav enkelt resas eftersom det inte var uppenbart att kod kopierats från UNIX in i LINUX. De flesta företag som sökte ett UNIX-liknande OS föredrog då att använda LINUX eftersom det ite var förknippat med samma legala risker även om LINUX inte var like fullfjädrat som BSD vid den tiden.

"Alla såg äpplen falla men det vara bara Newton som frågade sig varför"
B M Baruch