Tentamen i ID2206/ID2200/IS1350 Operativsystem
Onsdagen 2012-05-30 kl 0900-1300

SCS/ICT/KTH

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: Studenter som tenterar ID2200 besvarar inte frågor markerade med *****
IS1350: Studenter som tenterar IS1350 besvarar inte frågor markerade med **¤**

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstone en kortfattad motivering för att ge poäng
- Resultat beräknas finnas rapporterat i LADOK senast mån 25/6

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

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 4/6)

---

1) Systemanrop initieras/startas med en TRAP-instruktion. TRAP-instruktionen i sig tar normalt en parameter. Förklara vad den parametern är till för! (1p)

Den talar om för kärnan av vilken anledning man gjort TRAP. TRAP till kärnan görs inte bara vid systemanrop utan t.ex vid segmentation fault, sidfel etc.

2) Beskriv kortfattat vad som händer då man exekverar följande del av en process på en UNIX/Linux maskin

```
....
execlp("ls", "ls", "-la", NULL);
pid = fork();
......
```
(1p)

om det finns ett "ls"-kommando, dvs en exekverbar fil som heter ls, i någon av de kataloger som ingår i användarens sökväg (PATH) så exekverar processen "ls -la". Bara om ingen ls hittas kommer processen att skapa en ny barnprocess via fork().

3) Listor är en viktig datastruktur för implementationen av många delar av ett operativsystem. Antag att vi har en enkellänkad lista som termineras med att `next`-pekaren i det sista elementet är `NULL`. Datastrukturen för listelementen finns nedan. Skriv en "iterator"-funktion som givet en pekare till ett startelement i listan applicerar en godtycklig funktion med prototyp `void fun(listData)` på datat i varje element i listan från det utpekade elementet och till slutet av listan. (2p)

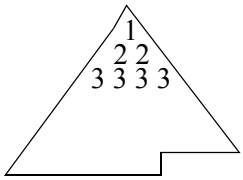
```
struct listElem {
   struct listElem *next;
   listData data;
};
```

```
void iterateAndApply(struct listElem *head, void (*fun)(listData))
{
    if(head != NULL) (*fun)(head->data), iterateAndApply(head->next, fun);
}
```

4) Beskriv grunderna för hur sidindelat virtuellminne fungerar. Beskriv fallen att systemet har respektive inte har hårdvarustöd för virtuellminne. (2p)

Se boken och föreläsningsanteckningarna från föreläsning 3

5) Antag att vi implementerat en mycket stor binär heap eller ett stort balanserat binärt träd genom att lagra elementen linjärt i en en-dimensionell vektor. Elementen lagras nivå-vis i vektorn enligt figuren nedan. Jämför och förklara den förväntade teoretiska och praktiska exekveringstiden för att traversera samtliga element i heapen/trädet om det görs enligt djupet-först eller bredden-först! (2p)



Nivåer i heap/träd

1 2 2 3 3 3 3 4 4 4 4 4 4 4 4 ...

Placering av elementen från de olika nivåerna i vektorn

Att traversera hela heapen/trädet kan göras för båda fallen i O(n) tid. i praktiken kommer en bredden först traversering/genomgång av alla element att ske sekvensiellt i minnet vilket ger god lokalitet. För en djupet först traversering/genomgång hoppar man från nivå till nivå i varje gren (1,2,3,4,5..... 2,3,4,5...) vilket för en stor heap som omfattar många sidor med data ger dålig lokalitet. I praktiken ger därför en breddenförst genomgång bättre exekveringstid om elementen lagras som i exemplet. (man kan naturligtvis lagra elementen så att en djupet-först algoritm ger bra lokalitet också)

6) I datorsystem med större virtuella adressrymder (t.ex 64, 128 bitar) implementeras sidtabellen ofta som en inverterad hashad sidtabell vilket påstås vara både minnes- och tids-effektivt. Kalle har observerat att man för varje process ändå måste hålla en datastruktur som håller reda på var processens sidor som ligger i SWAP-arean finns och att påståendet om den inverterade hashade sidtabellens minnes- och tidseffektivitet i praktiken inte stämmer. Förklara om Kalle har rätt eller fel! (2p)

Vad gäller datastrukturerna har Kalle rätt. Men om man har bra lokalitet kommer man i huvudsak att behöva accessa den inverterade sidtabellen och bara i enstaka fall hämta data från disken. Därför är det rimligt att anta att man får bra prestanda och att Kalle inte har rätt.

7) Förklara begreppen *sidfel* och *TLB-miss* och hur de relaterar till varandra! (1p)

Vid en TLB-miss finns ingen översättning från sidnummer till ramnummer i TLBn för den givna sidan. Det betyder inte att man i alla fal där man får en TLB-miss också har ett sidfel. Vid ett sidfel finns inte någon mappning från sidnummer till ramnummer i sidtabellen. Det kan bero på att sidan inte finns i minnet utan måste hämtas från disk (major pagefault) eller att sidan finns i minnet men att ramen tagits från processen av pageing-demonen (minor page fault)

8) Vilka är fördelarna/nackdelarna med lokala respektive globala sidutbytesalgoritmer? (1p)

Den största fördelen med en lokal algoritm är att den bara påverkar den process som behöver byta ut en sida i minnet eller få tillgång till fler ramar. Den kan därför vara lättare för andra användare att acceptera. En global algoritm kan göras effektivare och klara av att fler och större processer kan köra innan man får problem med dålig prestanda hos virtuellminnessystemet. Nackdelarna är att en lokal algoritm inte kan göras lika effektiv respektive att det kan vara svårare för andra användare att acceptera att de kan påverkas av andra användares processers sidfel.

9) För schemaläggning av processer/trådar kan oftast användaren ange en prioritet när processen/tråden skapas. i många system sker själva shemaläggningen ändå inte efter den av användaren angivna prioriteten utan efter en tillfällig prioritet (sk. current prioritet eller goodness). Förklara varför? (1p)

För att 1) kunna göra schemaläggningen rättvis så att även lågprioriterade processer får köra utan att svältas; samt 2) för att tillåta att man implementerar policys som t.ex prioriterar interaktiva processer högre då de I/O till dem blir klar.

10) ¤På moderna datorsystem är schemaläggningsenheten ofta en tråd. Men på sådana system bör schemaläggaren ändå ta hänsyn till processer (som alltså inte i sig schemaläggs) vid schemaläggningsbesluten. Förklara varför! (1p)

Moderna datorsystem kan antas ha fler-kärniga processorer och möjlighet att köra trådar sant parallellt. Man kan anta att trådar inom en process kommunicerar/synkroniserar med varandra. För att detta skall ske effektivt behöver trådarna inom processen få exekvera samtidigt. Därför bör/måste man ta hänsyn till vilken process trådarna tillhör vid schemaläggningen.

11) Förklara var och varför man hittar nedanstående information på just den platsen i filsystem baserade på i-noder (t.ex under Unix, Linux, Mac)
    i) Filnamn
    ii) Rättigheter
    iii) Tid när filen skapades
    iv) Tid när filen senast accessades (2p)

ii, iii och iv är egenskaper direkt knutna till filen och ligger därför i datastrukturen som implementerar filen, dvs i i-noden. En fil kan däremot vara känd under fler än ett namn och därför finns filnamn i directoryt/katalogen.

12) Vilken betydelse har det att en barnprocess ärver sina föräldrars fildescriptorer? (1p)

Det möjliggör för barnet att läsa/skriva till de filer föräldern har öppna, dvs för förälder och barn att kommunicera vi delade filer eller pipes.

13) Kalle hävdar att VFS (Virtual File System) lagret i filsystemsimplementationen under UNIX/(LINUX) bara behövs om man har distribuerade filsystem. Beskriv vad VFS lagret är, vilken funktionalitet det ger och red ut om Kalle har rätt eller fel i sitt påstående! (2p)

Det virtuella filsystenslagret är ett indirektionslager som gör det möjligt att ha inte bara ha distribuerade filsystem utan även också fler olika typer av lokala filsystem och partitioner. Det implementeras så att när en fil accessas identifieras den via en V-nod som i sin tur pekar ut vilket filsystem filen ligger i.

14) I bland annat Microsofts filsystem finns möjligheten för användaren att "defragmentera" hårddiskar. Förklara vad detta innebär och varför man vill göra det! (1p)

Filsystem är baserade på blockindelning/allokering av diskutrymmet. Alltså finns ingen extern fragmentering och den interna kan man inte göra något åt. Däremot kan en fils block komma att ligga utspridda på disken och genom att flytta om diskblocken försöker man få blocken som hör till varje fil att fysiskt ligga närmare varandra på disken. Det ger bättre prestanda genom färre/kortare diskarmsrörelser vid läsning/skrivning och underlättar READ-AHEAD av data.

15) Givet en sidstorlek om 200 bytes, ett primärminne om 3 ramar och följande minnesreferenser: 15, 700, 345, 406, 279, 110, 167, 532, 438, 650, 13, 153, 389, 912, 1037, 653, 789, 1136, 415
    a) Beräkna referenssträngen! (1p)
    b) Beräkna antalet sidfel för sidutbytesalgoritmerna Clock, FIFO, resp OPT! (1.5p)

a) 0, 3, 1, 2, 1, 0, 2, 3, 0, 1, 4, 5, 3, 5, 2
b) räkningarna måste redovisas! FIFO 11 sidfel, OPT 8, Clock 10

16) Beskriv vad det innebär att en global variabel eller funktion deklareras som `static` i C! Varför är det viktigt att kunna göra detta i implementationen av operativsystem? (2p)

Namnet tas bort från symboltabellen. Dvs namnet kan inte kommas åt av en länkare och blir bra synligt i "translation unit" dvs i källkodsfilen där variabeln deklarerats eller funktionen definierats, Detta är viktigt för att inte få namnkrockar när t.ex drivrutiner installeras eller generellt i ett så stort och komplext system som ett OS är.

17) Många system tillåter att man kan installera "stackade avbrottsrutiner"
    *a) Förklara vad detta innebär? (1p)
    *b) Red ut om detta kan ha någon betydelse för säkerheten i systemet! (1p)

a) det betyder att avbrottshanteraren för varje avbrott har en lista med funktioner som anropas i tur och ordning när ett avbrott hanteras till dess att en av funktionerna signalerar att avbrottet är färdighanterat. Root-användare kan installera nya rutiner/funktioner i listan. Dessa läggs normalt först i listan.
b9 det förenklar att installera funktioner som loggar vad som händer vid avbrott. Dessa läggsin först i listan och kan t.ex logga tangentbordstryckningar utan att användaren märker något. Dvs det är en möjlighet som förenklar för vissa typer av elak programvara (malware)

18) Kalle har upptäckt att när han försöker skriva ut var heapen och stacken startar för en applikation så får han olika, till synes slumpmässiga, startadresser för dessa varje gång han kör applikationen. Förklara varför! (1p)

Man kan anta att den virtuella adressrymden är mycket större än vad en enskild process praktiskt kan utnyttja.Därför slumpas i många fall den virtuella startadressen för stack(ar) och heap för att göra det svårare för "malware" (skadlig programvara som virus etc.) att hitta data och kunna angripa en process.

19) Beskriv två generella, garanterat deadlock fria lösningar till Dining Philospher problemet som maximerar paralleliteten. Bevisa att de är deadlock fria! (2p)

I dining philosophers så finns ett runt bord med N sittplatser och en gaffel per sittplats. På bordet står en skål med spagetti som filsosferna äter från. För att en filosof skall kunna äta krävs att han/hon får tillgång till två gafflar. För att skapa en deadlockfri lösning bryter man enklast mot något av Coffmans nödvändiga villkor för deadlock. Enklast är att bryta mot cirkulär väntan. Detta kan man t.ex göra genom att bara låta N-1 filosofer äta samtidigt eller att man numrerar alla gafflar i stigande nummerordning runt bordet och kräver att gafflarna tas upp i nummerordning, eller att man vid en plats måste ta upp vänster gaffel först medan man på alla andra platser tar upp höger gaffel först (samma resultat som numreringen).

20) *a) Förklara begreppen periodiskt respektive aperiodiskt realtidsystem! (1p)
*b) Vilken av typerna är lättast att schemalägga i fallet att man har ett hårt realtidsystem? Varför? (1p)

I ett periodiskt system återkommer de möjliga starttiderna för jobben med en viss period för varje process, Dvs alla framtida möjliga starttider är kända givet ett visst initialtillstånd. Man känner också alla deadlines. I ett aperiodiskt system vet man inte när jobb startas utan de triggas av asynkrona händelser. Detta gör att det är lättare att hitta optimala och fungerande schemaläggningar för rent periodiska system.

21) *Beskriv hur gränssnittet mellan drivrutiner och kärnan implementeras i UNIX, Linux, Windows, MacOS. (1p)

I alla fallen skall drivrutinen tillhandahålla en datastruktur i vilken man har pekare till de funktioner som implementerar gränssnittet. Normalt är det här en liten vektor med funktionspekare. De här funktionerna implementerar systemanropen mot enheterna. För t.ex. en hårddisk, som är ett sk. "block-device", innehåller gränssnitet i samtliga OS funktioner som motsvarar systemanropen på en fil, Dvs. open(), read(), write(), close()....

"Allting är svårt innan det är enkelt."
Johann Wolfgang von Goethe