Lösningar till tentamen i 2G1520 (fd 2G1504) Operativsystem

Tisdag 2005-08-30 kl 0900-1300

Institutionen för Mikroeletronik och Informationsteknik

Examinator: Robert Rönngren

1) I POSIX standarden har man valt att standardisera biblioteksfunktionerna och inte själva systemanropen. Varför? (1p)

Själva implementationen av systemanropen är beroende av den arkitektur (främst processor) man kör på och måste i regel ofta göras helt eller delvis i assembler. Skillnaderna blir då så stora mellan olika system att det inte är meningsfullt att standardisera på den här nivån. På C-biblioteksfunktionsnivå är det däremot enkelt att standardisrea gränssnitten och funktionaliteten.

2) Enklaste lösningen på hur man implementerar biblioteksfunktionerna för malloc(), realloc(), free() och calloc() bygger på att man lägger en header som talar om storlek på det allokerade blocket och annan kontrollinformation precis före den adress till det allokerade blocket som lämnas ut av malloc(), realloc() ... En sådan lösning är känslig för fel till följd av att en användare adresserar sig utanför den allokerade minnesarean. Beskriv en enklel lösning för att bygga in en mekanism för att upptäcka sådana felfall! (1.5p)

Enklaste lösningen bygger på att man fösrts och sist i headern för minneblocket lägger in data (ett bitmönster) i form av ett "magic number", dvs ett speciellt heltal. Skrivs detta sönder kan man genom att upptäcka det i kontroller som görs vid allokering/avallokering av blocken. Vidare kan man kontrollera att adressen till ett block som lämnas tillbaka ligger inom gränserna för det område av minnesrymden som kontrolleras av minneshanteringsfunktionerna. Dvs man kan hålla reda på t.ex var man startade allokera första minneblocket och var det sista slutade.

3) Vissa ser ett OS som en virtuell maskin - vad är det man vill att OS:et skall uppnå om man har det synsättet? (1.5p)

Det ger flera fördelar som: i) man kan dölja detaljer i den verkliga maskinen och drmed få ett bätrre användargränssnitt; ii) man kan öka portabiliteten frö programvara; iii) man kan nå bättre/enklare resusrsdelning om varje användare tror han/hon har en egen (virtuell) maskin; iv) man kan tillhanda virtuella resurser (jfr. virtuellminne) osv.

4) Ge exempel på vad som kan sägas vara policy respektive mekanism i en prioritetsschemaläggning! (1p)

Policy är hur man väljer (och ev förändrar) prioriteten hos processen. I UNIX implementeras t.ex en policy som sänker prioriteten för processer som använder mycket CPU tid. Mekanismen är hur man implementerar själva prioritetsschemaläggaren. I UNIX är den t.ex implementerad som ett antal köer, en för varje prioritet, som betjänas med round-robin.

5) Beskriv korfattat stegen som ett C-program bestående av flera källkodsmoduler går genom från källkod till dess att det finns i minnet som en process! (2p)

i) Pre-processor som gör textsubstitutioner; ii) C-kompilering som kontrollerar syntax och semantik och oftast genererar assembler som ett första steg, assemblern går sedan genom en assemblerare som genererar en objektfil som innehåller enbart relativa adresser; iii) länkaren tar en eller flera objektfiler och länkar ihop dessa, dvs ser till att de olika filernas respektive relativa adresser, som för varje fil är relativa den egna filstarten, översätts till att bli relativa starten på den ihoplänkade exekverbara filen; iv) laddaren laddar slutligen in valda delar av den exekverbara filen i det fysiska minnet på lediga ramar; v) sista steget är att OS:et skapar en post i processtabellen och när procesorm är ledig startar exekveringen av den första instruktionen som finns i minnet.

6) Ge en formel för den effektiva minnesaccesstiden i ett sidindelat system med två-nivåers sidtabell och TLB! (1p)

M = minnesaccesstid, s = söktid i TLP, P = sannolikhet för träff i TLB

effektiv minnesaccesstid = P*s + (1-P)*(s+2M) + M

7) Vad är en RAID disk och beskriv fördelarna med att använda RAID diskar istället för en vanlig disk. (1.5p)

Redundant array of inexpensive/independent disks. I en RAID kan data delas upp och skrivas till /läsas från flera diskar för att förbättra läs/skrivprestanda. Samma data kan också skrivas till flera diskar (ev. med felrättande

koder) för att minska sannolikheten att data förloras om en disk kraschar, dvs RAID kan också användas för att öka tillförlitligheten.

8) Beskriv vilka de minimala rättigheterna en användare pelle i ett AFS filsystem måste ha i varje katalog efter sökvägen: `~kalle/toughCourse/lab1/sol.c` för att pelle skall kunna accessa filen sol1.c (Obs! för mycket/för lite rättigheter ger poängavdrag pps. som avsaknaden av motivering) (1p)

Man behöver list-rätigheter i alla kataloger ner till och med lab1, i lab1 behöver man också de rättigheter som gör att man t.ex kan läsa (r) och/eller skriva (w) filen.

9) Kalle hävdar att det inte finns någon avgörande skillnad mellan att implementera ett feltolerant system mha flera processer eller flera trådar. Har han rätt eller fel? (1p)

I normalfallet har han fel! I de flesta system kommer ett allarligt fel i en tråd att påverka hela processen, t.ex så att den dödas, och därmed påverka alla trådarna i processen. Processer är i sig mer oberoende av varandra och på så sätt mer feltoleranta, dvs. ett fel i en process påverkar oftast inte andra processer.

10) Ge ett exempel på när man skulle vilja använda, och ofta använder, en "copy-on-write" semantik i moderna OS med virtuellminneshantering! Vad vinner man på det? (1p)

När man skapar en ny process genom att kopiera en redan exixterande provcess (jfr fork()) så kan man spara tid om man slipper kopiera hela adressrymden och endast kopiera ev. sidor som modifieras. Eftersom de flesta nyskapade processer byter ut sin adressrymd genom ett anrop till exec() är detta en rimlig optimering.

11) Beskriv stegen i ett Remote Procedure Call! (2p)

PÅ klientsidan anropas en stub-procedur some ser ut som et lokalt funktionsanrop. Stub-proceduren packar ihop (marshalling) parametrarana på ett nätverks och datorarkitetur obeorende format och skickas sedan till servern (oftast över ett nät). På serversidan packas meddelandet upp och man konstaterar till vilken lokal funktion det var ämnat. Den funktionen anropas med rätt parametrar. Resultatet från funktionen på serversidan packas i sin tur genom marshallong och returneras till klienten. Där tas det emot och packa upp och man ser till att returnara resultatet pss som om stub-proceduren varit lokal.

12) a) Vad innebär begreppet sessionssemantik i distribuerade filsystem? (1p)

En process som öppnar en fil får en lokal kopia som används under sessionen, ändringar blir inte synliga för andra användare förrän sessionen är över och filen stängs

b) Ge ett exempel på ett vanligt förekommande distribuerat filsystem som använder sessionssemantik! (0.5p)

AFS använder en något modifierad version av sessionssemantik.

13) Vad innebär filreplikering i ett distribuerat filsystem? Vad är det bra för och när kan man få problem? (1.5p)

Filreplikering innebär att man skapar flera kopior av en fil så att fler än en server kan ha en kopia lokalt. Det kan öka prestanda om en klient kan ladda hem data från närmaste tillgängliga server och det fkan förbättra tillgängligheten genom att man inte är beroende av bara en kopia/server. Däremot kan man få konsistensproblem om filen uppdateras.

14) Beskriv vad ett race-condition är! (1p)

Ett race-condition uppstår när flera processer accessar samma resurs (och ev. modifierar den) på ett osynkroniserat sätt. resultatet kan då bli beroende på i vilken ordning processerna accessat resursen.

15) Beskriv ett enkelt sätt att organisera resurshanteringen i ett system med ett ändligt antal resurstyper så att man garanterat kan undvika deadlock. För enkelhets skull kan antar vi att det bara finns en resurs av varje typ. Visa varför metoden undviker deadlock. (1.5p)

Man tilldelar varje resusr/resusrtyp ett unikt ordningsnummer. Man får bara allokera resuresr i strikt stigande resurstnummerordning. Dvs har man redan en resurs av typen 2 får man inte allokera fler av den utan att först släppa alla resusresr av tyepn 2, däremot kan man allokera nya resurser (alla på en gång) av t.ex typ 3. Man kan relativt enkelt bevisa att man på så sätt undviker att processer hamnar i cirkulär väntan, dvs man bryter ett av Coffmans vilkor. (Beviset kan tex utgöras av ett enkelt resonemang)

16) Hur skapar man enklast ett kompanjonvirus under MS-DOS/WINDOWS? (1p)

Eftersom man normalt letare efter ett filnamn med extension .com före extension .exe när man skall exekvera ett program i MS-DOS/Windows och de flesta program har extension .exe lägger man helt enkelt till ett program med samma namn men extension .com som kompanjonvirus.

17) Antag att man utgår från ett system som man säkert vet inte innehåller några filer infekterade med virus eller trojaner. Beskriv hur man skulle kunna gå till väga för att med stor säkerhet kunna upptäcka om någon fil blir infekterad, även av virus som är så nya att man inte känner igen deras signaturer. (1.5p)

Man kan skapa en typ av digital signatur för varje program genom att t.ex beräkna en checksumma eller ett hashvärde för varje virusfri fil som man sedan krypterar med en metod/nyckel som ett ev. virus inte kan känna till och sparar dessa krypterade signmaturer på disk. Vid varje programstart beräknas signaturen och jämförs med dn krypterade på disken. Är dessa inte samma har den exekverbara filen modifierats och kan alltså vara virusinfekterad varför man inte startar programmet.

18) Beskriv kortfattat hur man kan "länka" in en ny drivrutin i en OS kärna utan att man utför någon egentlig länkning (jfr LINUX) (1.5p)

Jfr föreläsningsanteckningarna (bilden där är bättre än vad jag enkelt kan åstadkomma på rimlig tid)

19) a) Kalle hävdar att det är Rate Monotonic Scheduling i en avgörande aspekt inte är lika bra som Shortest Job First som schemaläggningsmetod för hårda realtidsystem med en processor. Vad avser han? (1p)

RMS kan bara garantera att laster upp till ~ 0.69 kan klaras, medan SJF kan klara al alaster upp till 1 på ett singelprocessor system.

b) Förklara varför RMS ändå ofta väljs före SJF i många system! (1p)
RMS arbetar med fixa prioriteter vilket gör det enklare att skapa en enkel och effektiv schemaläggare.

20) Hur försöker man i WINDOWS2000 schemaläggning prioritera interaktiva processer? (1p)

Genom att, mha heuristik, temporärt höja processers prioritet då de väntat färdigt på olika I/O enheter.

21) Jämför risken för svältning i UNIX med vanlig "standard" schemaläggning och i LINUX (2p)

I UNIX har man normalt fixa prioriteter, dock kan en process som använder hela sin time-slice få sänkt prioritet. Detta reducerar risken för svältning men eliminerar den inte. Har man t.ex en grupp processer som alla är interaktiva kommer dessa inte få sänkt prioritet, men de kan ändå använda så mycket CPU tid att de effektivt stänger ute lägre prioriterade processer.

I LINUX körs däremot processpopulatione till dess att alla processer är blockerade (väntar på avbrott) eller dess quantum, som säger hur mycket CPU-tid de får förbruka, har tagit slut. När inga proceser finns att schemalägga beräknas quantum om för varje process. Det här betyder i praktiken att alla processer som vill exekvera förr eller senare också får exekvera (om man inte har en oändlig ström av nya högprioriterade processer som kommer till), dvs man undviker i praktiken svältning.

22) Antag att `sizeof(double) == sizeof(void *) == 8`. Vad kan man då säga om pek+2379 på en arkitektur som kräver alignment om man har följande deklarationer (förklara också varför): (1p)

```
typedef struct {
   union s {
     double x;
     struct {
       int i;
       char text[91];
       float text;
       void * prev, * next;
     } head;
   }
} MyType;

MyType *pek malloc(HUGENUMBER*sizeof(MyType));
```

Om vi antar att double kräver flest bytes för att lagras vet vi att (pek+2379)%8 = 0, dvs att adressen är en jämn multipel av 8. Detta peror på att vi inte skall få alignmentfel när man försöker accessa det första som kan ligga på den adress som pekas ut av pek som kan vara en double (jfr lab3)

23) Vad är grundfilosofin bakom ett OS designat på en sk. “exokärna”? (1p)’

Det bygger på en end-to-end argumentation som säger att allt som inte behöver göras i kärnan utan skulle kunna göras av användaren (i t.ex ett bibliotek) inte skall ligga i kärnan. Allt som då behöver skötas av kärnan är säker resursallokering.

"Förmågan att idag tänka annorlunda än igår skiljer den vise från den envise"
John Steinbeck