Tentamen i 2G1504 (fd 2G1113) Operativsystem
lördag 2004-03-13 kl 1000-1400

Institutionen för Mikroeletronik och Informationsteknik

Examinator: Robert Rönngren

Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.

Ange på omslaget vilken kurs du tenterar (2G1113 eller 2G1504) 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 5/4
-

Betygsgränser:
- Totalt antal poäng på tentamen är 31
- Godkänt (3) garanteras från 16 poäng

Lösningsförslag: anslås på kursens webbsida efter tentamen

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

1) Vilka tre grundkrav måste man kunna ställa på en prestandautvärdering? (1p)

2) Vad ligger på stacken vid programstart i ett UNIX system? (1.5p)

3) Ange tre huvudskäl till att man inför processer förutom att man kanske vill tillåta flera samtidiga användare! (1.5p)

4) I de flesta OS med schemaläggning baserad på prioriteter finns en övre gräns för hur hög prioritet en vanlig användare kan sätta på en process. Ändå kan samma process tidvis exekvera med ännu högre prioritet. Förklara hur det kan gå till och varför det är så! (1p)

5) a) Vid vilka tidpunkter kan relativadresser bindas till absolut adresser? (1p)
   b) Vilka av dessa adressbindningstillfällen är praktiskt användbara i virtuellminnessystem? (1p)

6) Kalle skall implementera sidindelat virtuellminne på en arkitektur som saknar TLB. Han funderar på om han skall välja en en-nivåers eller två-nivåers sidtabell. Vad bör han välja? (1p)

7) Varför måste man normalt öppna en fil innan man läser/skriver den och stänga den efteråt? (1p)

8) Följande anger en relativ sökväg i ett AFS filsystem `~kalle/toughCourse/lab1/sol.c`
   Användaren pelle har följande rättigheter

dir/fil	**rättigheter**
`~kalle`	`l`
`~kalle/toughCourse`	`none`
`~kalle/toughCourse/lab1`	`rl`
`sol.c`	`-rw-r--r--`

   Kan pelle läsa filen `sol.c`? Varför/varför inte? Förklara ditt svar!
   (det räcker inte att bara säga att han har tillräckliga eller inte tillräckliga rättigheter) (1p)

9) Under vilka förutsättningar kan ett multitrådat program exekvera snabbare än ett singeltrådat på en en-processors maskin? (1p)

10) Många OS använder “copy-on-write” för adressrymden i samband med `fork()`. Vad innebär det och vad hoppas man vinna på det? (1p)

11) a) Vilken typ av semantik med avsende på hur många gånger en viss begäran utförs kan man normalt åstadkomma i det generella fallet för begäran hos en server i ett distribuerat system? (1p)
b) Vilken semantik passar bäst för en idempotent begäran? (1p)

12) Kalle skriver på en distribuerad client-server lösning implementerad mha RPC där han planerar skicka svaret från servern i form av en länkad lista med resultat. Vilket är det största problemet han måste lösa? (1p)

13) Beskriv hur man någorlunda enkelt kan implementera sekvensiell semantik för filer i ett distribuerat filsystem! (2p)

14) Förklara begreppet synkronisering! (1p)

15) Kalle skall skriva en resurshanterare för ett system med två resurstyper. En process försöker alltid allokera en resurs av varje typ, vilket behövs för dess fortsatta exekvering. Kalle har hittat på ett protokoll som skall undvika synkroniseringsproblem enligt förljande:

Tag en resurs av typ A
Försök tag en resurs av typ B, men vänta inte längre än X tidsenheter på att en enhet skall bli ledig
Om man inte fått bägge resurserna släpp resurs av typ A och
  Tag en resurs av typ B
  Försök tag en resurs av typ A, men vänta inte längre än X tidsenheter på att en enhet skall bli ledig
  Om man inte fått bägge resurserna släpp resurs av typ B och börja om från början

Har den här modellen några problem? (1p)

16) Förklara varför det skulle kunna vara sämre ur säkerhetssynpunkt att inte behöva ge hela programnamnet med extensions när man startar ett program som t.ex i MS-DOS/WINDOWS än att behöva ge hela programnamnet som i t.ex UNIX! (1p)

17) Beskriv kortfattat de olika lagren som bör finnas i ett (generiskt) skalskyddat datorsystem! (2p)

18) Vissa operativsystem med stöd för multimediaapplikationer tillåter att kärnan och systemavbrott avbryts.
a) Förklara varför? (1p)
b) Vilka huvudsakliga sätt/modeller finns att implementera den här funktionaliteten? (1p)

19) Om en högprioritetsprocess skickar en fråga till en process med låg prioritet och väntar på svar från lågprioritetsprocessen kan man råka ut för att den högprioriterade processen får vänta även på andra processer med prioriteter mellan hög och låg innan den får svaret. Det är inte eftersträvansvärt. Skissa på en enkel lösning för att råda bot på problemet! (1p)

20) WINDOWS2000 (NT) har en intressant säkerhetsdetalj i sin sidutbytesalgoritm som särskiljer den från många andra operativsystems som vanlig UNIX mfl. Vilken är det och vilket problem löser den? (1.5p)

21) a) Beskriv schemaläggningen av “vanliga timesharing” processer i LINUX! (2.5p)
b) Vilka egenskaper har schemaläggningsmetoden? (1p)

22) Antag följande: `sizeof(int) == 4, sizeof(void *) == 8`

```
typedef struct {
  void *next, *prev;
  int number;
} Item;

Item vec[10];
```

och att `vec == 1248` (decimalt) Vad har då vec+1 för värde (decimalt)? (1p)

Lösningar till tentamen i 2G1504 (fd 2G1113) Operativsystem

lördag 2004-03-13 kl 1000-1400

Institutionen för Mikroeletronik och Informationsteknik

1) Att kunna genomföra pålitliga prestandautvärderingar hör till grundkunskaper för en civilingenjör. De grundläggande kraaven på en sådan är att man:
   mätt rätt saker (tyvärr är det inte ovanligt att man mäter något annat än det man tror sig mäta)
   har tillräcklig (och angiven) mätnoggranhet
   beskrivit och utfört utvärderingen så att den går att reproducera

2) Argumenträknare, vektor med pekare till kommandoradsargumenten, vektor med pekare till environmentvariabler, textsträngar med kommandoradsargumenten, textsträngar med environmentvariabler

3) Många problem löses naturligt av processer
   Bättre feltolerans/isolering
   Kan leda till bättre resursutnyttjande till följd av överlapp mellan t.ex I/O och beräkning

4) Användar proceser kan övergå till kernel-mode vid t.ex systemanrop. Då kan de behöva ha högre prioritet för att t.ex på ett effektivt sätt kunna hantera en uppstartad I/O operation.

5) a) Compile-time, load-time, execution-time
   b) I praktiken endast execution-time eftersom man inte på förhand vet vilket minne/ramar som kommer att finnas ledigt att ladAntag att minnesacesstiden är T

6) Antag att minnesaccesstiden är T. Med en-nivås sidtabell blir accesstiden minst 2T och med två-nivåers sidtabell blir accesstiden minst 3T. Alltså bör han välja en en-nivås tabell om den får plats i minnet.

7) Öppna: OS:et kontrollerar accessrättigheter. Det bokför också position i filen och hur många som öppnat den. Stänga: Bokföring - räkna ned antalet som har filen öppen och vid behov skriv tillbaka innehåll i buffert till disk

8) Eftersom pelle inte får lista, dvs läsa innehållet i katalogen toughCourse kan han heller inte läsa v-noden som används för att lagra information om var katalogen lab1 finns och därmed kan han inte komma åt att läsa den katalogen och hittar följdaktligen inte heller diskblocken för filen sol.c

9) Förutsätter trådar på kärnnivå där en annan tråd kan schemaläggas medan någon tråd väntar på I/O

10) Efter en fork skrivskyddas både förälder och barnprocessens adressrymd. Då endera parten försöker skriva en sida kopieras den (om den skall vara skrivbar). Eftersom barnprocessen ofta gör exec() snart efter att den skapats hoppas man på så sätt kunna spara tid på att inte från början kopiera hela adressrymden.

11) a) at least once eller at most once
   b) eftersom en idempotent begäran inte har några sidoeffekter passar at least once semantik bäst

12) Om vi antar att han inte har distribuerat delat minne, vilket verkar otroligt då han använder RPC, så måste han komma på ett sätt att packa ned den länkade listan så den kan värdeöverföras till klienten eftersom server och klient har olika adressrymder

13) Man har en server som delar ut "tokens"/biljetter av två oloka typer som ger rättighet att läsa respektive skriva filen. Om en process vill skriva filen så måste servern se till att få till baka alla utdelade lästoken innan den kan dela ut ett skrivtoken. Vill någon läsa filen så får det inte finnas något skrivtoken utdelat. Man kan dela ut flera lästoken samtidigt men bara ett skrivtoken.

14) När två eller flera processer kommunicarar för att bestämma i vilken ordning saker och ting får utföras

15) Kan leda till livelock om två processer omväxlande tar en resusrs av typen A resp B och sedan tvingas släppa dem och prova med resurser av typ B resp A osv.

16) Vissa OS letar efter filer med flera olika extensions, t.ex i MS-DOS/WINDOWS letar man först efter ett program med extension .com och sedan med extension .exe vilket få användare känner till och vilket öppnar möjligheter att installera trojanska hästar.

17) Utifrån och in:
   Intrångsdetektion - försök upptäcka försök till intrång
   Brandvägg - som bara släpper igenom viss trafik
   Autenticerad inloggning - bara “godkända” användare skall få komma åt systemet

Kryptering av känslig data och backuper
Fysiskt åtkomstskydd - vem som helst skall inte kunna komma åt datorn och t.ex konsolterminal, hårdvaru reset etc
Säkert OS med accesskontroll etc.

18) För att minska dispatch latency när en mjuk realtidsprocess behöver få köra. Man kan antingen avbryta kärnan i vissa punkter sk. preemption points eller så kan kärnan avbrytas var som helst. I bägge fall gäller att den avbrytande processen inte får modifiera datastrukturer som den avbrutna arbetade med.

19) Den låpgprioriterade processen får jobba med (ärva) samma prioritet som den högprioriterade medan den behandlar den senares förfrågan

20) Man kan säkerställa att en process som "begär" en ny ram får en nollställd ram - dvs man kan förhindra att data oavsiktligt sprids via primärminnet mellan processer

21) a) Varje process har en grundprioritet + nice i intervallet 0-40. Varje process har et individuellt tidskvanta som beräknas om då alla processer är blockerade eller har slut på sitt tidskvanta enligt: kvanta = kvanta/2 + prioritet. Processeer körs till dess deras tidskvanta tagit slut eller att de blockeras av t.ex I/O väntan Varje gång en schemaläggning görs beräknas processernas “godness” som används som prioritet vid schemaläggningen enligt: if(quantum > 0) godness = quantum + prioritet else godness = 0

    b) prioriterar jobb som utnyttjar mycket I/O, CPU bundna proceser får CPU tid proportionellt mot sin grundprioritet

22) Om inte alignment till 8 bytes krävs för void * så 4+8+8+1248 = 1268, annars 8+8+8+1272