Lösningsförslag till tentamen i 2G1520/1504 Operativsystem

Måndagen 2007-05-21 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

2G1504
- Studenter som tenterar 2G1504 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 STEX senast mån 11/6

Betygsgränser:
- Totalt antal poäng på tentamen är 31
- Godkänt (3) garanteras från 16 poäng
- Möjlighet till komplettering vid underkänt resultat på tentamen tillämpas inte

---

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 22/5)

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

1) a) Beskriv kortfattat vad som händer då man exekverar systemanropet `wait(PID, &status)` för en process som nyligen terminerat (dvs ingen annan process har exekverat `wait(PID,....)` ännu! (1p)
   b) Vad händer om ingen process exekverar `wait()` för en process som terminerar i ett UNIX system? (1p)

Lab1&2 central fråga:
a) Man läser av status för den terminerade processen - vilken lagras i parametern &status, processens post i processtabellen städas bort (dvs den kommer inte längre att finnas några spår av i systemet och blir inte en zombie process)
b) man riskerar att posten i sidtabellen inte (kan) tas bort och processen sysn som en sk. zombie-process

2) Om man råkar ut för ett segmentation fault i en process som exekverar på ett modernt OS - vad är det då som inträffat och på vilken nivå i datorsystemet upptäcks felet. (1p)

Man har försökt adressera minne man inte får adressera i den virtuella adressrymden eller så har man försökt göra en operation på minnet som man inte får göra som t.ex exekvera data som bara får läsas&skrivas eller att man försökt skriva till en skrivskyddad adress. Skyddet av minne sker i moderna OS med sidindelning på sidnivå och upptäcks normalt av hårdvaran i form av MMU:n.

3) Timesharing uppfanns redan i början av 1960-talet. Timesharing fick emellertid ingen större spridning förrän långt senare när datorsystemen utvecklats ytterligare. Vad var det för funktionalitet i hårdvaran som behövde utvecklas för att timesharing skulle bli användbart? (1p)

För att timesharing skall vara robust så krävs att man kan skydda de olika processernas adressrymder från varandra så att inte en process, avsiktligt eller oavsiktligt, skriver över en annan process adressrymd. Tidig hårdvara (och en del enkla processorer idag också) saknade hårdvarustöd för skydd av olika processers adressrymd

4) 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)

VFS är ett viktigt exempel på en abstraktion som finns motsvarigheter till i de flesta spridda OS. VFS lagret ger möjlighet att stödja fler än en filsystemsimplementation samtidigt. Det fungerar så att istället för att direkt accessa filen via en i-node (som skulle peka ut var på disken filen ligger i ett lokalt UFS filsystem) så accessas filen via en v-nod. v-noden pekar i sin tur ut var filen finns, t.ex om den finns i ett lokalt filsystem eller i ett distribuerat. Även om man inte har tillgång till distribuerade filsystem kan man ändå ha nytta av VFS lagret eftersom det, som sagts ovan, ger möjliget att stödja olika filsystemsimplementationer - men som ändå kommer att se ut som ett filsystem för användaren

5) En hårddisk innehållar ofta ett minne som kan användas som buffert/cache. Ge två exempel på hur det kan användas och hur det kan förbättra prestanda hos systemet! (1p)

För att få bra prestanda ur en hårddisk så krävs att den kan schemalägga diskarmsrörelserna på ett effektivt sätt - minnet kan användas för att buffra läs/skrivbegäran för att kunna omordna dessa för att få bra diskarmsschemaläggning. Bufferten kan också användas som en liten cache för att göra read-ahead vid läsningar och som en buffert för att kunna klara av tätt på varandra följande skrivningar till disken.

6) I system med virtuellminne baserat på sidindelning finns ofta en process som tar ramar från användarprocesser utan att sidfel uppstått och man explicit behöver en ledig ram. Vilket syfte fyller en sådan process? och vilken mekanism finns för att lindra effekterna om man råkar ta en ram från en process där innehållet faktiskt används (i.e. ligger i processens working-set)? (2p)

En sådan process finns i praktiskt taget alla OS som utnyttjar sidindelat virtuellminne. Dess uppgift är att se till att det alltid finns en mängd lediga ramar som kan användas för att snabbt betjäna processer som råkar ut för sidfel. Skulle man råka ut för att ta en ram från en process som använder sidan i ramen kan man i många fall minska problemen genom att implementera det som ofta kallas minor-pagefault. Detta innebär att systemet (i hård eller mjukvara) kommer ihåg vilka sidor som finns i de lediga ramarna. Om en process får ett sidfel kontrolleras först om sidan finns i någon ledig ram - i så fall behöver den inte hämtas från disken - vilket spar tid och ökar prestandan.

7) Antag att vi har ett datorsystem som använder en TLB för att snabba upp adressöversättningen, att tiden för en access i TLBn är N tidsenheter och att en access till primärminne tar M tidsenheter. Ge formler för den effektiva minnesaccesstiden givet att vi använder en en-nivås sidtabell respektive en tre nivåers sidtabell! (2p)

Antag att träffsannolikheten i TLBn är P. Då måste man vid en adressöversättning där man inte får träff i TLBn göra en respektive tre minnesaccesser om man har en en-nivås respektive en tre-nivås sidtabell. Den effektiva minnesaccesstiden blir alltså: P*N + (1-P)xM + M där x = antalet nivåer i sidtabellen

8) I framtida datorsystem kan man förvänta sig processorenheter med flera processorkärnor och större virtuella adressrymder (t.ex 128 bitar). Vilken av följande sidtabellsimplementationer är att föredra i sådana system: fler-nivåers vanlig sidtabell, inverterad sidtabell, hashad sidtabell? Varför? (1p)

Med så stora adressrymder är flernivåers sidtabeller i praktiken omöjliga att använda (jfr exempel på övningen). En inverterad sidtabell kan användas om inte primärminnet blir för stort eller om man inte delar sidor mellan processer. Bäst lämpad är trots allt en hashad sidtabell (inverterad eller vanlig) eftersom hashning är en effektiv teknik att mappa en stor (gles) mängd data på en mindre, dvs uppslagningar kommer att gå fort, och den tillåter enkelt att flera processer delar minne (sidor)

9) 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 LRU, FIFO, resp OPT! (1.5p)

a) De sidor som refereras är: 0, 3, 1, 2, 1, 0, 0, 2, 2, 3, 0, 0, 1, 4, 5, 3, 3, 5, 2 - Referenssträngen blir då (utan upprepningar) 0, 3, 1, 2, 1, 0, 2, 3, 0, 1, 4, 5, 3, 5, 2

b) Uträkningar måste redovisas: LRU 11, FIFO 11, OPT 8

10) Beskriv begreppen dynamisk laddning respektive dynamisk länkning! (2p)

Dynamisk länkning och laddning är tekniker som är vanligt använda i de flesta moderna OS för att minska den mängd kod (text) som måste hållas inladdad i en process virtuella/fysiska adressrymd. mest tydligt är detta kanske i Microsofts OS där man hittar systembibliotek mm. som så kallade dll:er - dll = dynamic linked library.

Dynamisk laddning: modulen laddas under run-tome och endast vid behov, t.ex första gången en funktion anropas. Dynamisk länkning: modulen laddas/länkas in under run-time och endast vid behov. Skillnaden mot dynamisk laddning är att om någon annan process har modulen inladdad så laddas inte en ny kopia av modulen in utan den delas om så är möjligt (dvs att den inte innehåller skrivbara delar - data).

11) Vilken systemprogramvara brukar ansvara för att säkerställa att en användare inte kan exekvera en fil med t.ex data eller text även om den har rättigheter i filsystemet (protection) som medger exekvering och hur implementeras det? (1p)

Vi bortser från textfiler som innehåller sk shell-scripts , dvs textkommandon till kommandotolken som kommandotolken kan lösa och utföra - vilket inte är exekvering i egentlig mening utan bara interpretering.

Text, data etc. som behövs för en process exekvering laddas in av en laddare. Indata till laddaren är en exekverbar fil på ett väl specifierat format. Förutom att filen innehåller text, initialiserad data mm. så beskriver formatet storleken på dessa areor. Formatet innehåller också i allmänhet någon form av identifiering, ofta i form av ett sk. magic number, av att det verkligen är en fil på ett format som kan laddas för exekvering. Om laddaren inte kan identifiera att filen är på rätt format kan den heller inte laddas för exekvering.

12) Kalle kör en dator som använder operativsystemet Free BSD. Han har noterat att när han exekverar en `exec()` i en ny process han skapat så ersätts hela den gamla processens virtuella adressrymd medan vissa datastrukturer i operativsystemet som t.ex tabellen med deskriptorer för öppna filer inte ersätts. Han planerar nu att skriva om koden för `exec()`-systemanropen så att en process efter en `exec()` får en "ren" tabell med fildeskriptorer där bara `stdin, stdout` och `stderr` är öppna. Är detta en bra ide? (1p)

Direkt relaterad till laboration 1: Free BSD är en UNIX variant (BSD = Berkley System Distribution). Att som Kalle föreslår, ge alla processer en ny filderskriptortabell efter en exec() är en dålig ide på de flesta system. I synnerhet på system som skall vara POSIX-kompatibla eftersom det omöjliggör kommunikation mellan processer mha pipes.

13) a) Beskriv kortfattat begreppen livelock respektive deadlock (baklås). (1p)
b) Vilket/vilka problem kan man räkna med att stöta på om man försöker hitta ett deadlock i ett system med hjälp av en debugger? (1.5p)

a) Både livelock och deadlock är synkroniseringsproblem. I ett livelock synkroniserar en grupp processer/trådar på så sätt att de hela tiden exekverar synkroniseringen men att de aldrig kommer framåt. I ett deadlock som omfattar en grupp processer så kan ingen av de inblandade processerna fortsätta sin exkvering eftersom alla inblandade processer väntar på att någon annan process i gruppen skall göra något för att den skall kunna fortsätta sin exekvering.

b) Deadlock är i regel beroende av hur saker och ting har schemalagts för att det skall uppstå. Ofta - men inte alltid - kan det finnas vissa schemaläggningar som inte leder till deadlock medan andra gör det. Dvs det kan vara svårt att upprepa ett deadlock. Det är i sig det första och mest grundläggande problem man kan stöta på. Andra problem som är relaterade till detta är att det nästan är säkert att den exakta schemaläggningen (timingen) ändras om man exekverar programmet i en debugger och koden som exekveras kan också i sig påverkas av att den kanske måste kompileras på ett annat sätt för att kunna köras i debuggern - vilket också påverkar schemaläggningen (timingen)

14) Antag att du exekverar följande program på en vanlig PC tillverkad omkring år 2004. Vad kan man då förvänta sig i form av exekveringstid för de två olika programmen nedan om man använder operativsystemen MS-DOS respektive Windows XP? (obs! beskriv vad som påverkar prestandan i de olika fallen!) (2p)

Program i

```
void main(int argc,char **argv)
{
  char arr[500][1000];
  int i, j;
  for(i=0; i<500; i++)
    for(j=0; j<1000; j++)
       arr[i][j] = 0;
}
```

Program ii

```
void main(int argc,char **argv)
{
  char arr[500][1000];
  int i, j;
  for(i=0; i<1000; i++)
    for(j=0; j<500; j++)
       arr[j][i] = 0;
}
```

Det enda som skiljer programmen åt är i vilken ordning man traverserar kolumner och rader i matrisen. Det rör sig alltså om en fråga om lokalitet. Eftersom den virtuella och den fysiska adressrymden är endimensionella måste en matris lagras antingen radvis eller kolumnvis. I C lagras matriser radvis (vilket man inte behöver känna till för att kunna få full poäng - däremot måste man kunna lista ut att en matris måste lagras antingen på det ena eller det andra sättet....) Det betyder att i det program där man går igenom matrisen radvis så sker minnesaccesserna till konsekutiva adresser medan man i det andra fallet sprider accesserna. Det påverkar prestandan på två nivåer: i) hårdvarumässigt genom att det senare fallet ger sämre lokalitet och därmed kan ge sämre cacheprestanda, samt ii) lokaliteten kan påverka prestanda hos ett eventuellt virtuellminnessystem implementat av OS:et genom att antalet sidfel kan påverkas.

MS-DOS: 16 bitars OS utan stöd för virtuellminne. Att OS:et är 16 bitars påverkar troligen inte prestandan i någon avgörande del i detta fall då matrisen är av typen char (8bitars data). Eftersom det inte finns stöd för virtuellminne så är den enda skillnaden i prestanda man kommer att se beroende på cache beteenden.

Windows XP: 32 bitars OS med virtuellminne i form av sidindelat minne. Programmet med sämre lokalitet kommer att generera fler sidfel omm inte hela matrisen samtidigt ryms i de ramar processen får använda (dvs om man kan ha hela datastrukturen i minnet samtidigt blir det ingen skillnad). Cachebeteendet påverkar prestanda pss som för fallet med MS-DOS - (man kan anta att just cachebeteendet är mer eller mindre oberoende av OS:et).

15) På vanliga datorsystem med en processorenhet (CPU) bör schemaläggningen alltid sträva efter att exekvera processer/trådar på CPUn för att få bästa möjliga prestanda ur systemet. På datorystem med flera processorenheter är det inte nödvändigtvis så att man alltid får bättre prestanda om man alltid försöker schemalägga processer/trådar för exekvering på varje ledig processorenhet jämfört med att man ibland låter lediga processorenheter gå tomma (idle).Förklara varför det kan vara så och ge exempel på en schemaläggningsmetod för flerprocessorsystem som kan ta sådan hänsyn (2p)

(Detta var den "svåra" frågan på denna tentan)Om man har tillämpningar som består av flera processer/trådar som samverkar i lösandet av en uppgift. vilket är troligt på fler-processor system, så kan man anta att dessa processer synkroniserar med varandra. För att synkroniseringen skall bli så effektiv som möjligt bör alla processer i tillämpningen få exekvera parallelt - dvs komma framåt i beräkningen samtidigt. Om man bara kan schemalägga en del av tillämpningen, några processer, kan dessa komma att få vänta på att synkronisera med processerna som inte får exekvera. Än värre kan det bli om synkroniseringen i tillämpningen bygger en optimistisk synkroniseringsmekanism. I en optimistisk synkronisering antar man att indata från andra processer levereras i tid och att man därför kan exekvera framåt så länge man har indata. Skulle man däremot upptäcka att man fått en leverans av indata som kommit försenat måste beräkningarna rullas tillbaka till ett korrekt tillstånd och startas om igen. Dvs om inte alla processer får exekvera riskerar man att man att hamna i en situation där man måste lägga ytterligare processorkraft på att rulla tillbaka systemet. Optimistisk synkronisering är inte ovanligt i t.ex databassammanhang och simuleringar.....

16) Bankir algoritmen kan användas för att undvika deadlock genom att hålla ett system i "säkert" tillstånd.Givet följande system med tre resurstyper, vilket är det minsta värdet på X som gör att systemet är i ett säkert tillstånd? (1p)

Process	Allokerat			Max behov			Tillgängligt		
A	1	2	3	6	9	4	2	X	2
B	0	4	2	3	7	5			
C	1	1	4	3	4	4			
D	2	2	3	5	9	9			

I synkroniseringssammanhang är deadlock det kanske allvarligaste problemet. De två grundkunskaper man bör ha inom området är dels villkoren för att deaadlock skall kunna uppstå och hur dessa kan utnyttjas för att designa system som inte kan hamna i deadlock (Coffmans villkor) samt den mekanism man kan bygga resurshanterare på i system med deadlockrisk men där man kan undvika deadlock mha resurshanteraren - Bankir algoritmen.

Systemet är, enligt bankir algoritmen, i ett säkert tillstånd om det finns en kjedja av allokeringar så att någon process kan allokera hela sitt maximala resursbehov, varefter den terminerar och dess resurser frigörs så att någon annan process kan allokera hela sitt maximala resursbehov osv. till dess att alla processer terminerat. I exemplet finns tre resurstyper <a.b.c> Med de resurser som finns tillgängliga kan bara process C få hela sitt

behov a typerna <a,c> uppfyllt. Om X=3, dvs tillgängliga resurser i systemet är <2,3,2> så kan C allokera maxbehovet, terminera och lämna tillbaka sina resurser, antalet tillgängliga resurser blir då <3,4,6>. Det räcker för att B skall kunna allokera sitt maximala behov vilket frigör resurser så antalet tillgängliga resurser blir <3,8,8> Viket räcker för att antingen A eller D skall kunna exekvera klart - dvs alla processer kan terminera. Svaret är alltså X = 3.

17) Skriv i C en "ls-"trojan som för användaren verkar exekvera hans/hennes vanliga `ls`-implementation men som också exekverar funktionen `ond_kod()` (du behöver inte ha med rätt include filer i svaret). (1p)

Övnings/labrelaterad:

```
int main(int argc, char * argv[])
{
    ond_kod();
    execv("/bin/ls", argv);
    execv("/usr/bin/ls", argv);
}
```

Den exekverbara modulen döps till "ls", obs att man då inte kan nyttja att söka enligt PATH då man då riskerar att starta programmet rekursivt.

18) Du ska köpa in mjukvara för ett client-server system där två leverantörer lämnat in offerter med prestandautvärderingar av tiden det tar att etablera kontakt mellan en klient i Stockholm och en server i Köpenhamn. Utvärderingarna har i båda fallen utförts enligt dina instruktioner på följande målsystem: *"The performence evaluation was performed on two lightly loaded SUN Ultra workstations creating a connection over the Internet between a client in Stockholm and a server in Copenhagen".*

- Leverantör A skriver i sin utvärdering: *"Timing was perfomed with the hardware supported clock with a resolution of 1 micro second. The average time to create a connection was found to be 0.164 the error in the measurements was 0.001 sec."*
- Leverantör B skriver i sin utvärdering: *"Timing was perfomed with a micro-second resolution clock. The average time for connection establishment was found to be 0.178 +/- 0.037 sec at 95% confidence level"*

Kan du med detta underlag bestämma om leverantör As eller Bs implementation är den mest effektiva? (1p)

Leverantör A anger att felet i mätningarna är detsamma som upplösningen hos den klocka man använt för att mäta tiderna. Detta verkar varken troligt, med tanke på att det rör sig om en uppkopling över Internet, eller särskilt seriöst. Leverantör B har däremot angivit ett konfidens intervall som verkar troligt. Detta täcker dessutom in den av leverantör A angivna tiden. På grundval av de data man fått kan man alltså inte säga att den ena eller den andre leverantörens implementation är effektivare än den andres. Däremot verkar leverantör B mer trovärdig vilket kanske kan påverka valet av leverantör.....

19) Hur försöker man i Microsofts operativsystem (fr.o.m. Windows 2000) prioritera interaktiva processer (1p)

Genom att ge processer som väntat på I/O som blivit klar tillfälliga prioritetshöjningar under nästa time-slice. Prioritetshöjningarna är fixa beroende på vilken I/O enhet processen väntat på. Storleken på prioritetshöjningarna är bestämd genom heuristik.

20) Vilka av följande funktioner/systemanrop har man tillgång till från moduler som exekverar inne i en LINUX kärna? Varför?
i) `printf()`
ii) `read()`
iii) `fork(`
`iv)socket()` (1p)

Ingen. Alla är antingen biblioteksfunktioner som utnyttjar systemanrop eller biblioteksfunktioner som implementarar gränssnittet för ett systemanrop. Och systemanropen är inte tillgängliga inifrån kärnan - de används bara som metoden att kunna kommunicera med kärnan från user-space.

21) I mitten av 1990-talet fanns två huvudkonkurrenter på massmarknaden för Operativsystem - Microsofts OS och fria versioner av UNIX. Ändå tog, det då relativt okända, LINUX snabbt över UNIX roll som alternativ till Microsiofts OS. Förklara varför? (1p)

När UC Berkley släppte sina BSD distributioner av UNIX fria stämdes de på upphovsrättsliga grunder av AT&T. Ingen som var vid sina sinnens fulla bruk vägade då använda BSD distributionerna av rädsla för att i sin tur inte drabbas av stämningar från AT&T. Det fria, POSIX-kompatibla alternativ som fanns att tillgå vid den tiden var LINUX som på så sätt fick en hjälp på vägen. För övrigt kan man notera att vid denna tidpunkt var BSD UNIX ett väl utvecklat och väl fungerande OS medan LINUX mer var i sin linda.....

"Vågar man inte ta nästa steg blir man stående på ett ben resten av livet"
Kinesiskt ordspråk