ROYAL INSTITUTE
OF TECHNOLOGY

Lösningsförslag till tentamen i IS1350 Operativsystem

Måndagen 2015-03-16 kl 09:00-13:00

Examinator: IS1350 Jim Dowling
Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.
Ange vilken kurs du tenterar och vilken termin du registrerades ffg. på 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 åtminstone en kortfattad motivering för att ge poäng

Betygsgränser:

- Godkänt (E) garanteras från 50% av den maximala poängen på respektive kurs

Frågor:

(1) Vad är en Memory Management Unit, MMU, och varför behövs den? (1p)
Lösning

En MMU är en minneshanteringsenhet som används för att översätta virtuella adresser till fysiska adresser. Den kan innehåller en Translation Lookaside Buffer och brukar implementeras som en del av en CPU.

(2) Vad är en process? Vad används heap, data respektive bss sektionerna som man hittar i den virtuella adressrymden för en process till? (1p)
Lösning

En process är ett program under exekvering. Heap: ett minnesområde (memory pool) som

används för dynamisk allokeringar (malloc() och free()) - den är läsbar och skrivbar. Data: initialiserade globala and statiska lokala variabler. BSS (Block Started by Symbol): oinitialiserade globala and statiska lokala variabler

(3) Beskriv den tre tillstånds-processmodellen som används av CPU-schemaläggaren i Unix. Beskriv vilka tillståndsövergångar är tillåtna och beskriv en händelse som kan orsaka en tillståndsövergång. (1p)

Lösning

Blocked, Ready, Running. En process kan gå från running till blocked när I/O sker, från blocked till ready när I/O:n är klar, från ready till running när processen är schemalagd och från running till ready om processens tidsdel (time-slice) tar slut. Otillåtna tillståndsövergånger är: blocked-¿running, ready-¿blocked.

(4) Förklara skillnaden mellan cachning och buffring och ge några konkreta exempel från Unix. (1p)

Lösning

Cachning används för att snabba på läsningen av data genom att hantera en kopia av data. Det finns L1, L2 cachar till primärminne, TLB är en cache till sidtabellen. Buffring är ett minnesområde där data (inte en kopia av data) sparas temporärt under tiden data flyttas mellan ett ställe till ett annat. Buffring behövs för att effektiversa användningen att dataöverföringsresurser - t.ex., att klumpa ihop data för att skicka större nätverkspaket eller att överföra större mängder av data till hårddiskan åt gången.

(5) Vad betyder det om man säger att en process är I/O bunden? Vad betyder det om man säger att en process är CPU bunden? (2p)

Lösning

En process som är I/O bunden (I/O-bound) har som primär flaskhalsan skrivningar eller läsningar till/från I/O enheter. En process som är CPU bunden (CPU-bound) har som primär flaskhalsan tillgång till CPU cyklar. Om man kan tar bort flaskhalsan, kan man minska körtiden av processen.

(6) Förklara skillnaden mellan preemptiv schemaläggning och ickepreemptiv schemaläggning. (1p)
Behöver antingen preemptiv eller ickepreemptiv schemaläggning hårdvarustöd? Om ja, vilken typ av hårdvarustöd? (1p)

Lösning

Vid non-preemptive schemaläggning exekverar alla startade funktioner färdigt, iaf tills de överlämnar kontrollen till kärnan. Vid preemtive schemaläggning kan kärnan avbryta exekveringen av processer för att schemalägga en annan körfärdig process (t.ex., för att den process som kör har använd sin tidsdel eller för att en annan högre prioritet process är körfärdig). Ja, preemptiv schemaläggning behöver en CPU klocka för att genera timer-interrupts. Preemptiv schemaläggning

(7) Två processer, X och Y, har följande sekvensiella exekveringar (sequential execution patterns):

- X: [cpu 6 ms; I/O 5 ms; cpu 3 ms; I/O 4 ms; cpu 6 ms]

- Y: [cpu 2 ms; I/O 2 ms; cpu 4 ms; I/O 2 ms; cpu 2 ms]

Antag att I/O operationerna inte interfererar med varandra och att I/O operationerna är blockerande och schemaläggaren är preemptiv med en tidsdel (time-slice) av 10ms. Antag att X schemläggs först och därefter schemläggs processerna en efter den andra i Round-Robin-ordning (processer har ingen prioritet). Beräkna den totala tid för att exekvera både X och Y. (2p)
Räkna om den totala tid som behövs för att exekvera både X och Y om vi sänker tidsdelen till 3ms. (1p)

Lösning

27 ms (10 ms tidsdel) 26 ms (3 ms tidsdel)
Se slutet för en tabell med beräkningar.

(8) Beskriv två fördelar av NTFS jämfört med FAT filsystem. (1p)

I Unix File System, givet ett filnamn, beskriv hur fil-metadata används för att läsa upp den första diskblocken som tillhör filen. (2p)

Lösning

NTFS kan ha större filer, längre filnamn, och NTFS har stöd för journalling.

När en process läser första block till ett fil, först måste filsystemet resolva all inodes i sökvägen till filen. Filsystemet underhåller en datastruktur med alla inodes och innan filen läsas måste filsystemet kontrollera behörigheten av användaren att läsa filen. I inoden till filen finns en pekare till en "disk block location", där innehållet av filen sparas i data blockar.

(9) Hur många sidfel får man om man har 4 ramar, ren demand paging och följande referensssträng: 5 2 1 7 1 3 4 2 6 1 7 5 6 4
För algoritmerna OPT, LRU och FIFO. (2p)

Lösning

8, 12 och 12.

(10) Ett av Coffmans villkor för deadlock brukar lyftas fram som det enklaste att bryta.
a) Vilket villkor är det?
b) Förklara/argumentera för varför det ofta är det enklaste att bryta! (2p)

Lösning

a) Cirkulär väntan brukar vara enklast att bryta i de flesta fall

b) Att bryta Cirkulär väntan kan man enkelt göra om alla accessar kritiska sektioner/delade resurser i samma ordning. Att bryta Coffmans övriga villkor är svårare: msesidig uteslutning skulle kräva att alla resurser i systemet skall kunna delas, dvs att de inte får ägas/hanteras exklusivt av någon - det är ganska uppenbart att det inte fungerar för de flesta typer av resurser som kan uppdateras/modifieras. Hålla och vänta om man ska kunna bryta det villkoret måste man kunna ta alla resurser man behöver på en gång - vilka resurser man behöver kan vara svårt att förutse, man kan tvingas släppa resurser man redan har och kanske arbetar med när man vill ha en ny och att bygga en resurshanterare som tillåter att man i en operation kan allokera många olika typer av resurser kan vara svårt. Ingen preemption kräver att man kan ta resurser från någon vilket ger två svårigheter - vem skall man ta resurser från och hur sparar man tillståndet för den som man tar resursen från och för själva resursen så att det kan återställas. Slutsatsen är att cirkulär väntan generellt sett är enklast att bryta.

(11) Förklara varför den publika nyckeln (public key) i asymmetrisk kryptering kan på ett säkert sätt delas över okrypterade länkar. Med andra ord, förklara hur asymmetrisk krypteringen fungerar. (1p)
Vad är en digital certifikat och hur byggs den med hjälp av asymmetrisk kryptering? (1p)
Lösning

Envägsfunktioner är förutsättningen för asymmetrisk kryptering. I asymmetrisk kryptering har avsändare och mottagare har olika krypteringsnyckel - avsändare brukar har public nyckeln till mottagare som har en privat nyckel. Om man kryptera med public nyckeln måste man dekryptera med privat nyckeln. Bara mottagaren har privat nyckeln. Det går inte att använda den publika nyckeln för att dekryptera ett krypterat meddelande

Ett digitalt certifikat är en datafil som består av kryptonycklar (public och privat nycklar). Certifikatet kan användas för kryptering och digitala signaturer. Certifikat utfärdas av en certifikatauktoritet. Ett digitalt certifikat kan använts för att signera ett dokument genom att räknar ut en kontrollsumma med hjälp av dokumentet och krypterar den med sin privata nyckel. Mottagaren dekrypterar kontrollsumman med avsändarens publika nyckel, beräknar dokumentets kontrollsumma och jämför den beräknade kontrollsumman med kontrollsumman som dekrypterades med avsändarens publika nyckel. Om kontrollsummorna stämmer överens har dokumentet signerats med avsändarens privata nyckel och meddelandet har inte förändrats på vägen över nätet.

(12) Förklara vad ett systemanrop är, hur ett systemanrop genomförs och vad som händer i OS:et vid ett systemanrop! (1p)
De flesta operativsystem har en lång livslängd. Förklara varför Windows 32 subsystem stödjer så många olika API:er och vilka problem som introduceras för underhåll av operativsystem. (1p)
Lösning

a) Systemanropen är det gränssnitt/de funktioner som operativsystemet gör tillgängligt för användarprogram att anropa för att få tjänster utförda av operativsystemet. (dvs . de är operativsystemets API). När ett användarprogram utför ett systemanrop så anropas normalt en biblioteksfunktion i C som utför själva systemanropet. Dessa biblioteksfunktioner är det som standardiseras i t.ex POSIX standarden. I biblioteksfunktionen läggs parametrarna på de ställen där de skall ligga för att os:et skall kunna hitta dem, vilket ofta är i register. Man laddar också in systemanropsnumret i ett register och sedan TRAPas till kärnan. Det som händer då är att processen/tråden som utför systemanropet byter stack till en stack som används när den exekverar i kärnan och den fortsätter att exekvera i kernelmode och hanterar systemanropet. I kärnan kontrolleras först varför man trappade till kärnan. När man konstaterat att det var pga ett systemanrop så kontrolleras vilket systemanrop det var. Var det ett systemanrop som implementeras i själva kärnan anropas den funktionen direkt. Var det ett systemanrop somimplementeras av en drivrutin slår kärnan upp den drivrutinen i en drivrutinstabell och anropar rätt funktion i drivrutinen. Antingen returnerar man direkt eller om det är ett blockerande systemanrop, som vid t.ex. blockerande I/O, så kan processen blockeras och återupptas då ett avbrott kommer som talar om att operationen är klar.

b) Windows 32 subsystem stödjer så många olika API:er för bakåtkompatabilit. Dvs, gamla program som skrev mot gamla API:er fortsätter att funka med nya versioner av Windows. Den trade-off här är att man måste underhåller en mer komplicerade operativsystem med mer funktioner som ska testas.

(13) Vad är Window Hardware Abstraction Layer (HAL) och hur hjälper den att förbättra portabilitet av Windows? (1p)
I virtual memory management subsystem för Windows NT det finns en balance set manager som startar en working set manager om det inte finns tillräckligt många lediga ramar (frames) i systemet. Varför? Vad vill den working set manager åstadkomma? (1p)
Lösning

HAL är ett abstraktionslager som gör underlätta portabilitet av OS:et till nya hårdvara. HAL gör att windows stödjer olika hårdvara platformer utan att behöva helt olika versioner av OS:et. Alla componenter i Windows 2000 kärnan accessar hårdvaran genom HAL. På en PC, HAL kan ses som en drivrutin till moderkortet som tilllåter instruktioner från ett högre nivå API men förhindrar direkt tillgång till hårdvaran.

(14) Varför är det snabbare att göra en context-switch (växlar) mellan kärntrådar än att göra en context-switch mellan processer? (1p)
Lösning

När en context switch händer måste kärnan spara tillståndet av processen eller tråden så att den kan återupptas vid ett senare tidspunkt. En context switch har större kostnader för processer än för kärntrådar. Dvs, det är snabbare att schemalägga kärntrådar än processer. Kärntrådar har sina egna program counter, stack och registrar. Men en process har mycket mer information som ska sparas - allt från sidtabller, TLB entries, osv.

(15) Unix operativsystem använder copy-on-write för att implementera *fork*() systemanropet när en ny process skapas. Vilka delar av en process adressrymd måste kopieras omedelbart och vilka delar kan försenas och kopieras först när det sker en skrivning? (1p)
Lösning

Copy-on-write är en optimeringsstrategi som används i operativsystemet när kärnan skapar en ny process med hjälp av fork. När fork anropas och en barn process skapas av en föreldra process, delar barnet adressrymden av föreldra processen, men sidorna som delas av föreldar och barnet markeras som "read-only". Bara när sidorna kommer att ändras skapar OS:et en kopia av sidorna. I moderna UNIX OSes är alla sidor markerade som copy-on-write, inklusive heap, stack, och BSS.

(16) Hur kan Unix systemanropet *pipe* används för att synkronisera kommunikation mellan två processer? Vilka parametrar tar systemanropet *pipe*? (1p)
Lösning

(17) Linux oftas kallas för en monolitisk kärna (monolithic kernel). Vad är en monolitisk kärna? Hur kan man slimma ner storleken av Linux kärnan för att sänka minnesanvändningen? (1p)
Lösning

I en monolitisk kärna, de flesta operativsystem tjänster och drivrutiner kör som en del av kärnan och inte som processer på user-nivå. Motsatsen till monolitiska kärnor är mikrokärnor, Du kan slimma ner storlekan av Linux kärnan genom att kompilera om kärnan utan att kompilera in de drivrutiner som din hårdvara egentligen behöver.

(18) I Linux kärnan moduler kan laddas in och laddas ur vid körtiden. Hur minskar kärnan risken att extern källkod kan orsakar problem inom kärnan? Dvs, vilket stöd finns det för isolering (isolation) av laddbara moduler? (1p)

Lösning

En Linux modul har en väldefinerad API som används av kmod, kernel module daemon, som exekverar modprobe för att ladda in modulen. Det finns inga direkt stöd för isoleringen av laddbara moduler i Linux. De kör i kärnan och kan krascha kärnan. Däremot försöker Linux minska risken att en dåligt skriven modul kan krascha kärnan genom att använda väldefinerad API och genom kärnans egna implementationen av syscalls som malloc (kmalloc()). En drivrutin är ett exempel på en modul i Linux.

(19) Vad är buffer overflow och vilka metoder använder moderna C kompilatorer för att förhindra buffer overflow? (1p)

Lösning

Buffer overflow händer när en process försöker spara mer data i en buffert än vad som får plats. Ofta skrivs då efterföljande minnesområde över. Detta kan leda till krasch eller säkerhetshål. T.ex., man kan skriva över returadressen till en funktion med adressen av en annan funktion.

(20) Magnetiska hårddiskar kan förbättra sin read throughput (antal byter som läsas per sekund) genom att byta från en first-come, first-served (FCFS) policy till en förbättrad policy. Beskriv en policy som borde öka read throughput under en workload som består av många concurrent (jämlöpande) processer som läser filer slumpmässigt från disken. (1p)

Lösning

Några exempel av policy som kontrollerar ordningen av läsningar och skrivningar till och från hårddiskar är Shortest seek-time first, SCAN, C-SCAN, LOOK, C-LOOK.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	19	20	21	22	23	24	25	26
X	R	R	R	R	R	R	B	B	**B**	**B**	B	I	I	I	R	R	R	B	B	**B**	**B**	R	R	R	R	R	R
Y	I	I	I	I	I	I	R	R	**B**	**B**	R	R	R	R	B	B	B	R	R								

R=Running
I= Ready (Idle)
B=Blocked (Sleeping)

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	19	20	21	22	23	24	25
X	R	R	R	I	I	R	R	R	B	B	B	B	**B**	R	R	R	B	B	**B**	**B**	R	R	R	R	R	R
Y	I	I	I	R	R	B	B	I	R	R	R	R	**B**	B	I	I	R	R								

Figure 1: Preemptive Scheduling