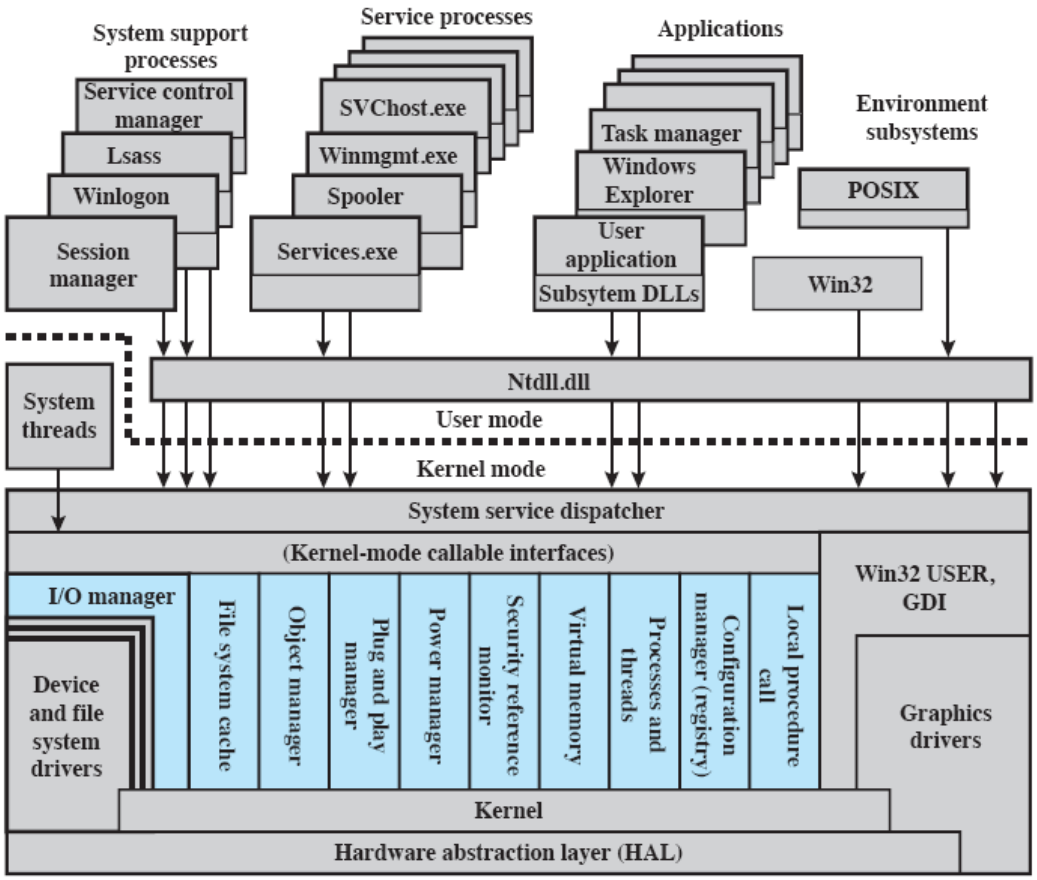
Kompendium i TDT4186 Operativsystemer Høst 2008

Oppdatert med datoer for 2009



Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

WINDOWS Vista Arkitektur

Ved:
Mads Nygård
Norvald Ryeng

Innholdsfortegnelse

1 Introduksjon til emnet

1.1 Meningen med emnet

I kurset om operativsystemer skal dere lære om operativsystemers oppbygging generelt. Vi understreker oppbygging. Dette emnet er ikke noe anvenderkurs der dere lærer å bruke forskjellige operativsystemer. Operativsystemene WINDOWS, UNIX, SOLARIS og LINUX anvendes gjennomgående som eksempler, men dette er likevel ikke ment å være noe direkte kurs i disse systemene. Dere vil kjøre de praktiske øvingene mot WINDOWS Server 2003.

De fleste av dere kommer ikke til å arbeide direkte med konstruksjon eller vedlikehold av operativsystemer. Først og fremst skal dere lære om operativsystemer fordi hvert eneste datamaskinsystem dere siden vil komme bort i, inneholder et operativsystem. Når dere skal konstruere eller implementere programsystemer, er det helt nødvendig å forstå og kunne utnytte egenskapene til operativsystemet.

Teknikker fra operativsystemer brukes også i andre flerbrukersystemer; f.eks. databasesystemer og språk med parallellitet (som Java), og de spiller en viktig rolle i reguleringssystemer; f.eks. prosesskontrollsystemer. Det skulle være åpenbart at det er viktig å beherske disse teknikkene godt, eller i alle fall å ha skjønt hva man selv ikke behersker. Tenk for eksempel på kontrollprogrammer for atomkraftverk, flytrafikk osv. Etter hvert vil slike systemer omgi oss i større og større grad, og det er noen av dere som skal lage disse!

Sist, men ikke minst, kan dette emnet tas som et "case study" i konstruksjon av programsystemer. Vi tar for oss en klasse ganske store og kompliserte systemer og ser på hvordan de kan konstrueres. Hvordan bør de brytes ned i komponenter, hvordan kan de ulike komponentene utformes, hvordan må egenskaper veies mot hverandre osv? Vi ser på hvordan kravene som stilles til systemet, innvirker på konstruksjonen, og på hvordan konstruksjonen bestemmer egenskapene til systemet.

1.2 Råd angående emnet

Selv om mye tid går med til å forstå spesifikke teknikker, forsøk å få øye på og beholde det store bildet. Å konstruere programsystemer blir nemlig hovedbeskjeftigelsen til de fleste av dere, og det er ingen bedre måte å lære det på enn å se hvordan andre (relativt kompetente folk) har gjort det.

Prøv også å få lest igjennom pensum så tidlig som mulig i semesteret, gjerne i sammenheng. Vi tror altså det er spesielt viktig å skaffe seg oversikt i dette emnet. Det blir mye lettere å forstå enkelthetene når dere skjønner sammenhengen alt skal passe inn i. Ikke la volumet skremme. Læreboka er ordrik (som så mange andre amerikanske lærebøker…), men ganske lettlest. Konsentrer dere om hovedlinjene i hvert kapittel ved første gjennomlesning, og ikke bruk for mye tid på å huske detaljer. Bokas kapitler bør nok leses i den rekkefølge de er skrevet.

De obligatoriske øvingene innebærer programmering, og dette har en tendens til å bli mye arbeid, ikke minst for de av dere som har minst programmeringserfaring. Men uansett er det ingen bedre måte å avmystifisere stoffet på enn å "gjøre det selv". De fleste av dere har også brukt programmeringsspråket Java i flere programmeringsemner allerede. Og de delene av Java som dere kanskje ikke har så stor erfaring med, er beskrevet spesielt i dette kompendiet. Således vil ikke terskelen bli så stor som mange kunne tro.

1.3 Innhold i kurset

Via NTNUs nettsider finner du bl.a. følgende informasjon om emnet:

> *TDT4186 OPERATIVSYSTEMER*
>
> Faglærer: Arvid Staupe
> Nivå: Tredjeår
> Vurdering: Skriftlig
> Karakter: Bokstav
> Omfang: 4F + 1Ø + 7S = 7.5SP
> Plassering: Høst; Uke 34-47
> Forelesning: Ti 12-14 i S3 & To 14-16 i S5
> Øving: Ti 16-17 i S5
> Eksamen: Dato 05.12.08 – Start 0900 – Lengde 4 timer – Kode D
>
> *Faglig innhold*: Emnet vil etablere definisjoner, prinsipper, rammeverk og arkitekturer for ulike typer operativsystemer. En vil diskutere målsettinger og avveiingsspørsmål, funksjoner og tjenester, samt strategier og organisering. Emnet vil vektlegge prosessor-bruk, lagertildeling, styring av inn/utenheter, samt kommunikasjon mellom og koordinering av prosesser. Viktige eksempler vil være WINDOWS XP, UNIX SVR4, SOLARIS 2X og LINUX.
>
> *Læringsmål*: Emnet skal gi studentene en forståelse for grunnleggende konsepter og tilhørende teknikker som er nødvendige for styring av og samarbeid mellom moderne datamaskiner.
>
> *Læringsformer og aktiviteter*: Forelesninger og øvinger. Ved utsatt eksamen (kontinuasjonseksamen) kan skriftlig eksamen bli endret til muntlig eksamen.
>
> *Obligatoriske aktiviteter*: 2 øvinger.
>
> *Anbefalte forkunnskapskrav*: Emnene TDT4120 Algoritmer og datastrukturer og TDT4160 Datamaskiner grunnkurs, eller tilsvarende kunnskaper.
>
> *Kursmateriell*: Oppgis ved semesterstart.

Her bør følgende bemerkes:

Som forkunnskaper angir vi bl.a. emnet TDT4120 Algoritmer og datastrukturer. Gjennom dette har man igjen avhengighet til emnet TDT4100 Objektorientert Programmering som tilsvarer det minimum av programmeringskunnskaper som trengs som inngangsbillett. Normalt har de fleste også tatt emnet TDT4140 Systemutvikling med tilhørende stort prosjektarbeid.

1.4 Forelesninger og øvinger i kurset

Når det gjelder forelesningene på mandager og fredager, vil dere treffe Mads Nygård, mens det er Norvald Ryeng, stipendiat / PhD-student, som vil stå for øvingsoppleggets auditorietimer på tirsdagene. På øvingssalen, som er rom 414 i P15 bygget, vil det være flere undervisningsassistenter / studentassistenter til stede for å hjelpe dere. Se for øvrig Kap. 2.

1.5 Pensum i emnet

Lærestoffet som vil dekke emnet er:

- W. Stallings: **"Operating Systems – Internals and Design Principles"**; Prentice-Hall, 6.utgave 2009. (WS - Boken kjøpes på bokhandelen).

  I utgangspunktet er Kap. 1-15 pensum. Dette stoffet bør man forstå forholdsvis godt. (Kap. 16-18 er ikke pensum).

- M. Nygård: **"Transparenter fra forelesningene i emnet TDT4186"**; IDI, 2008 (Direkte inkludert i heftet).

  Forelesningene oppfattes altså som en integrert del av emnets pensum.

- N. Ryeng: **"Løsninger på øvingene i emnet TDT4186"**; IDI, 2008 (Senere tilgjengelig via emnets hjemmeside).

  Øvingene oppfattes altså også som en integrert del av emnets pensum.

Vi bruker helt bevisst ordet "forstått" og ikke ordet "kunne" fordi det er viktig å forstå totaliteten og de store linjer – mer enn de mest intrikate småbitene – i dette emnet. Hvis vi ikke skulle rekke over alt som er angitt her, vil vi selvfølgelig justere pensum ut fra det.

1.6 Tidsplan for emnet

Forelesningstimene på mandager og fredager er planlagt som følger:

Uke/Dag	Innhold	Kapitler	Foiler
35 – Ti	Introduksjon / Bakgrunn	WS-1, 2	MN-A
35 – To	Introduksjon / Bakgrunn	WS-1, 2	MN-A
36 – Ti & To	Bruk av CPU	WS-3, 4	MN-B
37 – To	Forhold mellom prosesser	WS-5, 6	MN-C
38 – Ti	Forhold mellom prosesser	WS-5, 6	MN-C
39 – Ti	Forhold mellom prosesser	WS-5, 6	MN-C
39 – To	Bruk av lager	WS-7, 8	MN-D
40 – Ti & To	Bruk av lager	WS-7, 8	MN-D
	Kjøring av prosesser	WS-9, 10	MN-E
	Kjøring av prosesser	WS-9, 10	MN-E
	Bruk av I/O	WS-11, 12	MN-F
	Bruk av I/O	WS-11, 12	MN-F
	Innebygde Systemer	WS-13	WS-13
	Sikkerhet	WS-14, 15	WS-14, 15
	Oppsummering og red fxr eksamen	Alt	MN-X
	Gamle eksamensoppgaver / Spørretime		
	Eventuelt		

OBS:
Vi bruker alle de fjorten ukene 35-47. Ukene35, 36, 39, og 40 foreleses det både i tirsdagstimene og torsdagstimene, mens uke 37 foreleses det bare i torsdagstimene, og uke 38 foreleses det bare i tirsdagstimene.

Øvingstimene i auditoriet på tirsdager er planlagt som følger:

Uke	Innhold	Kommentar
39	Orientering om øvingsopplegget Introduksjon til tråder og synkronisering	
40	Presentasjon av øving P1 Presentasjon av øving P2	**P1**: Gjensidig utelukkelse **P2**: Synkronisering av tråder (Obligatorisk)
41	Diskusjon av øving P1 Diskusjon av øving P2	Innleveringsfrist – P1 & P2: **Mandag i uke 42 (dvs. 12.10)**
42	Gjennomgang av øving P1 Gjennomgang av øving P2	
43	Presentasjon av øving TA Presentasjon av øving TB	**TA**: Kap. 6 & 7 **TB**: Kap. 10 & 11
44	Gjennomgang av øving TA Gjennomgang av øving TB	
45	Presentasjon av øving P3 Presentasjon av øving P4	**P3**: Tidsstyring av prosesser (Obligatorisk) **P4**: Virtuell filhåndtering
46	Diskusjon av øving P3 Diskusjon av øving P4	Innleveringsfrist – P3 & P4: **Mandag i uke 47 (dvs. 16.11)**
47	Gjennomgang av øving P3 Gjennomgang av øving P4	

OBS:
Vi bruker altså de ni ukene 39-47. Totalt gir det ni auditorietimer.

Her er det verdt å merke seg følgende:

- **Første forelesningstime** blir da mandag 25. august, mens **første øvingstime** i auditoriet likeledes blir tirsdag 22. september. **Øvingssalen** blir da tatt i bruk en uke etter dette igjen; altså f.o.m. mandag 29. september. Endelig er **eksamenstidspunktet** lørdag 5. desember.

- Vi har både **to teoriøvinger** (TA-TB) og **fire praksisøvinger** (P1-P4). Begge teoriøvingene (TA & TB) samt to av praksisøvingene (P1 & P4) er frivillige. De to andre praksisøvingene (P2 & P3) er obligatoriske. De obligatoriske praksisøvingene må dere altså gjøre, og vi anbefaler at dere gjør de frivillige praksisøvingene også. Motivasjonen er at dette er den eneste måten å lære seg et slikt stoff på skikkelig. Det er også lurt å gjøre de frivillige teoriøvingene. Begrunnelsen er at vi på eksamen naturlig nok må stille mer denne type spørsmål.

- For hver av øvingene er det i auditorietimene inkludert en **presentasjonssesjon** samt en **gjennomgangssesjon**, mens hver av praksisøvingene også har inkludert en **diskusjonssesjon**. De obligatoriske øvingene må leveres inn senest en arbeidsdag før tilhørende gjennomgang – altså innlevering **senest mandag før gjennomgang på tirsdag**. En kan gjerne begynne å arbeide med øvingene før de blir presentert, da forelesningene som dekker tilhørende stoff ofte gis noen uker før.

1.7 Relevant informasjon

Dere finner ulik informasjon om emnet på ulike steder:

- En egen hjemmeside:

  http://www.idi.ntnu.no/emner/tdt4186

  Her ligger det allerede en god del nyttig informasjon – bl.a. tidsplaner for emnet og informasjon om øvingsopplegget. Relevante nyheter og annen informasjon vil kontinuerlig legges ut her, så sjekk denne hjemmesiden ganske ofte.

- Foreliggende kompendium

  I tillegg til dette introduksjonskapitlet (Kap. 1) finner dere også en utdypende presentasjon av øvingsopplegget (Kap. 2), en omtale av programmeringsomgivelsen samt en introduksjon til ulike programmeringsmekanismer som skal brukes (Kap. 3), en beskrivelse av teoriøvingene (Kap. 4), tekstene til praksisøvingene (Kap. 5), oppgaver til og løsninger på åtte tidligere eksamenssett (henholdsvis Kap. 6 og Kap. 7) samt kopier av lysark til forelesninger (Kap. 8).

- Utfyllende dokumentkopier

  Bakerst i dette kompendiet vil dere altså få tilgang på kopier av foreleserens lysark, mens kopier av forfatterens lysark vil dere få tilgang til via emnets hjemmeside. Etter gjennomgangen av hver av øvingsoppgavene vil dere likeledes få tilgang på kopier av løsningsforslagene.

1.8 Støttelitteratur

For de som ønsker seg utfyllende materiale angående det foreleste stoffet, vil vi anbefale følgende bøker:

- Avi Silberschatz, Peter Galvin & Greg Gagne: **"Operating System Concepts"**; John Wiley, 8.utgave 2009.

  Dette er en ny utgave av den gamle læreboka – fullgod på det mer teoretiske, men mangelfull på det mer praktiske.

- Andrew Tanenbaum & Albert Woodhull: **"Operating Systems – Design and Implementation"**; Prentice Hall, 3.utgave 2006.

  Dette er en svært interessant lærebok - etter som den inneholder komplett kildekode for et helt operativsystem, og gjennomdiskuterer det samme totalsystemet.

- Gary Nutt: **"Operating Systems – A Modern Perspective"**; Addison-Wesley, 3.utgave 2004.

  Dette er også en moderne lærebok, men enda mer ordrik og noe mer tunglest enn vår.

2 Presentasjon av øvingsopplegget

I dette kapitlet finnes en utdypende presentasjon av øvingsopplegget.

2.1 Overordnet informasjon

Hovedformålet med øvingene i emnet er å konkretisere og anvende teorien i læreboka. Dere skal få god trening i å bruke operativsystemet til å løse problemer, og innblikk i hvordan sentrale mekanismer fungerer ”på innsiden”. I vårt øvingsopplegg vil dere bruke operativsystemet Windows Server 2003, og programmeringsomgivelsen JDK versjon 1.6.0.2.

Programmeringsøvingene skal altså implementeres i Java, og kjøres på Windows 2003 terminalklienter. Det er derfor en fordel å gjøre seg kjent med både operativsystemet og programmeringsspråket. I neste kapittel i dette kompendiet gis en kort innføring i hvordan en skriver Java-programmer, og hvordan de kompileres og kjøres ved hjelp av JDK. Dette er ment som en hjelp på veien for de som måtte ha behov for å lære seg dette, evt. for de som trenger oppfriskning av gamle kunnskaper. Således forventes det at dere skal redigere, kompilere og kjøre Javakode. Det meste skulle være kjent fra før. Det er ellers anbefalt at dere benytter dere av ulike former for hjelpeinformasjon. Et eksempel på dette er API-oversikten som Sun har på sine hjemmesider. (Se http://java.sun.com/javase/6/docs/api/). I tillegg til dette har vi også inkludert i neste kapittel en kort innføring i bruk av spesielle operativsystemrelaterte mekanismer som tråder og synkronisering. Dette er ment som en slags ”kokebok” for bruk av disse mekanismene, og kan anvendes og utvides til å løse noen av de problemene som dere kommer til møte i øvingene.

Det er adgangskontroll til øvingssalen som øvingene veiledes på. Hvis du ikke har tilgang til øvingssalen, anbefales det å ta kontakt med IDIs ekspedisjon i IT Vest. I tillegg trenger du som overalt ellers brukernavn og passord på WIN-NTNU-NO-domenet. Hvis du mot formodning ikke har dette bør du ta kontakt med Orakelkontoret i Sentralbygg II.

2.2 Detaljert informasjon

Som nevnt i innledningen av dette kompendiet er øvingsopplegget delt inn i to kategorier; teoriøvinger og praktiske anvendelser. Teoriøvingene er ment som konkretisering av teorien i læreboka og skal gi trening med tanke på eksamen. Det vil si at disse vil være nyttige og er ment som eksamensrelevante. Disse øvingene er således pensum, men vil bli gitt som frivillige øvinger. Praktiske anvendelser er også ment som konkretisering av teorien i læreboka. Disse øvingene er først og fremst tenkt å gi dere trening i bruk av teorien, og skal kunne gi dere en bedre aktiv forståelse av et operativsystem. Anvendelser er nyttige som et redskap for aktive læringsprosesser. Det vil i alt bli gitt 4 praksisøvinger, hvorav to er obligatoriske.

De obligatoriske øvingene skal gjøres i grupper på 3. Den tilhørende registrering skjer via emnets hjemmeside. Dere har da muligheten til selv å danne deres egne grupper. Vi vil likevel forbeholde oss retten til omrokeringer dersom det er behov for dette. Frist for påmelding er mondag 28. september. En endelig liste over grupper vil komme på emnets hjemmeside etter at fristen er gått ut, så følg med!

Øvingene skal gjøres på IDIs maskiner. Rom 414 i P15 blir reservert for emnet (mandag-fridag 1600-2000). Her stilles ca. 50 PC-er til disposisjon. På de reserverte tidspunktene vil det være assistenter til stede for veiledning. Øvingssalen er låst og skal være låst til enhver tid. Dere må derfor bruke adgangskort (studentkort) for å få adgang til øvingssalen. Kontakt evt. IDIs ekspedisjon for mer informasjon.

Øvingene vil bli presentert i større detalj enn i dette kompendiet i øvingstimene på tirsdagene. Her vil det også bli gitt enkle tips for gjennomføringen av øvingene. Det er videre mulig allerede da å stille spørsmål. Det er videre anbefalt at dere bruker nyhetsgruppa ntnu.ime.idi.tdt4186 aktivt. Det vil ellers bli avsatt treffetid hvor dere kan komme til øvingskoordinator eller underviser med spesielle spørsmål. Informasjon om treffetider vil komme senere på hjemmesiden – http://www.idi.ntnu.no/emner/tdt4186. (NB: For at alle skal få tilfredsstillende veiledning, er det sterkt ønskelig at dere respekterer de treffetidene som blir fastsatt).

Innlevering av praksisøvinger skal gjøres i henhold til de fristene som er fastlagt (se øvingsplan). Besvarelsene leveres til studentassistent/undervisningsassistent samtidig som demonstrasjonen på øvingssal gjøres. Generelt kreves således følgende:

- Innlevering av kildekode til programmet.
- Demonstrasjon av programmet for undervisningsassistent / studentassistent.
- Innlevering av utskrift fra kjøringen.

Kravene kan variere noe fra praksisøving til praksisøving. De konkrete kravene er spesifisert i de enkelte oppgavebeskrivelsene.

3 Gjennomgang av programvaren

I dette kapitlet finnes en omtale av programmeringsomgivelsen som skal brukes, samt en introduksjon til ulike programmeringsmekanismer som vil trenges.

3.1 Programmeringsomgivelse – Sun JDK

Som i flere tidligere programmeringsemner skal dere også i dette kurset bruke Sun's Java Development Kit versjon 1.6.0.2 til å kompilere og kjøre Javakode. Dere kan skrive kildekode i den editoren dere selv måtte ønske, for eksempel Textpad, JCreator eller Emacs. De kommandoene dere vil få bruk for i JDK er:

- **Javac**: Kompilator som kompilerer javakode til såkalt bytecode; dvs. en maskinuavhengig maskinkode. Ingen lenking er nødvendig (det foretas dynamisk). Dere må oppgi hele navnet på filen - inkludert endelsen .java - for at javac skal finne den.
- **Java**: Dette programmet brukes for å kjøre applikasjoner. Dere må oppgi hvilken .class fil som skal kjøres. Dere kan også gi inn parametere til applikasjonen, for eksempel slik:
  java MyClass param1 param2 param3
  Parameterene som følger etter klassenavnet gis inn til applikasjonen som args[]-tabellen i main-metoden.

Begge disse kommandoene må kjøres fra kommandovinduet (Command Prompt) som ligger på startmenyen.

3.2 Tråder i Java

Tråder i Java er instanser av klassen Thread, og opprettes og slettes som andre objekter. Etter at man har opprettet en tråd, må man starte kjøringen av den med start()-metoden. Objektet vil da utføre en kodesekvens som en egen tråd. Det er to måter å spesifisere hvilken kode en tråd skal kjøre. For det første kan man subklasse trådklassen og redefinere run()-metoden. Denne metoden blir da kjørt av det nye trådobjektet. Den andre måten er å lage en klasse som implementerer grensesnittet Runnable og dermed metoden run(). En instans av denne klassen sendes da som innverdi til konstruktoren til trådobjektet. Run()-metoden til dette nye objektet vil da bli kjørt av den nye tråden. Når run()-metoden til en tråd avslutter/returnerer, avslutter tråden kjøringen sin. Dette er den anbefalte måten å avslutte en tråd på.

Nå vil metoder i trådklassen som kan være nyttige bli beskrevet. Trådklassen inneholder flere metoder enn de som er beskrevet her.

- **public Thread(**ThreadGroup group, Runnable target, String name**)**
  Konstruktoren til trådobjektet kan ta en eller flere av de tre parametrene som er nevnt over i den rekkefølgen som er angitt:
  - **ThreadGroup group**: Denne parameteren sier hvilken trådgruppe tråden skal knyttes til. Trådgrupper er grupper av tråder som man kan tenke seg å behandle som en enhet. Man kan kalle ulike funksjoner på et helt sett av tråder på en gang ved å bruke trådgrupper. Vanligvis har man ikke behov for å spesifisere noen trådgruppe.
  - **Runnable target**: Denne parameteren brukes hvis man ønsker å få kjørt run() metoden til et Runnable-objekt i stedet for å subklasse trådklassen.
  - **String name**: Brukes til å sette et navn på tråden. Vanligvis ikke nødvendig, men kan gjøre debugging lettere.

- **public static void sleep(**long millis**)**
  Denne metoden fører til at tråden som kjører nå, stopper utførelsen i det oppgitte antall millisekunder eller til den blir avbrutt.

- **public void interrupt()**
  Avbryter en tråd. Dette vil vekke en tråd som sover eller venter på en betingelse. (Se Synkronisering i Java).

- **public static Thread currentThread()**
  Returnerer trådobjektet til den tråden som kjører nå.

public final void suspend()
Denne funksjonen fører til at tråden den blir kalt i, stopper å kjøre. (Anbefales ikke brukt i JDK 1.6).

public final void resume()
Hvis denne blir kalt i en tråd som har blitt suspendert, vil den suspenderte tråden fortsette kjøringen. (Anbefales ikke brukt i JDK 1.6).

public synchronized void start()
Starter utførelsen av tråden funksjonen blir kalt i.

public final void stop()
Denne funksjonen avslutter en tråd ved å lage et ThreadDeath unntak. Dette unntaket bør ikke fanges. (Anbefales ikke brukt i JDK 1.6).

Deprecated metoder

Metodene suspend(), resume() og stop() er deprecated i JDK 1.6, hvilket vil si at de ikke anbefales brukt. Stop() er deprecated fordi et kall til denne metoden mens tråden holder på med å modifisere variable, kan føre til at disse blir satt i en inkonsistent tilstand. I stedet bør man la run()-metoden avslutte hvis man ønsker å avslutte en tråd. Resume() og suspend() er deprecated fordi de kan føre til vranglåser. Man har sjelden bruk for disse funksjonene, og hvis man har det kan man oppnå det samme ved å bruke wait() og notify() som er beskrevet i neste avsnitt. For en mer fullstendig beskrivelse av hva som er galt med suspend(), resume() og stop(), og hvordan man klarer seg uten dem, se http://java.sun.com/javase/6/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html.

3.3 Synkronisering i Java

Java bruker en variant av monitorkonseptet hvor alle objekter potensielt kan fungere som monitorer. Man trenger ikke å opprette monitoren eksplisitt, men indikerer i stedet hvilken kode man vil beskytte og hvilket objekts monitor som skal kontrollere tilgang til denne koden. Det er to måter å beskytte kode med en monitor på, og begge bruker ordet synchronized.

Den første måten er å deklarere en metode som synchronized. Metoder som er synchronized beskyttes av monitoren til objektet metoden kjøres på. Hvis alle metodene i en klasse er synchronized har man en klassisk monitor. Den andre måten å beskytte kode på er å lage en synkronisert blokk med kode. Dette gjøres på følgende måte:

```
synchronized(objekt) {
   // kode som skal være beskyttet
}
```

Koden inne i synchronized-blokka er beskyttet av monitoren til objektet inne i parentesene.

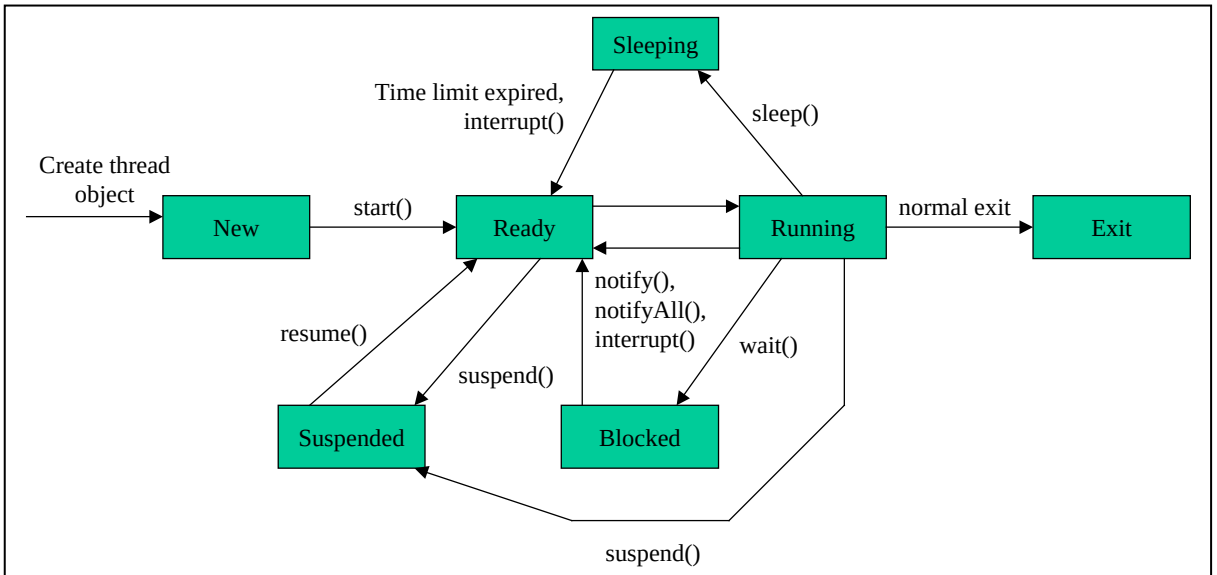
Når en tråd ønsker å kjøre kode/metoder som er synchronized, må tråden prøve å få monitoren (låsen) som beskytter koden. Hvis monitoren er ledig, får tråden låsen, og kan kjøre koden. Når tråden er ferdig med koden som er synchronized, slipper den låsen igjen slik at andre tråder får lov til å kjøre koden. Hvis en tråd prøver å kjøre kode som er synchronized, og låsen er opptatt, må tråden vente til låsen blir frigjort før den kan fortsette. På denne måten oppnår man gjensidig utelukkelse.

En tråd kan godt ha monitoren til flere objekter samtidig, og den kan også ha monitoren til det samme objektet flere ganger. Det siste skjer hvis man i en synchronized metode eller kodebit kaller en annen synchronized metode i det samme objektet. Siden objektet allerede har monitoren, vil dette gå bra, men tråden må slippe opp monitoren like mange ganger som den har låst den før andre tråder kan slippe til.

Hvis en tråd oppdager at den ikke har en ressurs som den trenger inne i en synkronisert kodebit, kan den kalle wait()-metoden til objektet som den har monitoren til. Denne funksjonen tilsvarer cwait() i læreboka, bortsett fra at det bare er én ventekø. Når en tråd kaller wait() vil tråden settes til å vente, og slipper monitoren. Hvis en tråd har lagt til en ny ressurs, kan den kalle notify()-metoden til objektet den har monitoren til. Denne funksjonen tilsvarer cnotify() i læreboka, og vil vekke en tilfeldig tråd som har

kalt wait()-metoden til objektet notify() ble kalt på. Man kan også vekke opp alle trådene som har kalt wait() på et objekt ved å kalle notifyAll() på dette objektet. Denne funksjonen tilsvarer cbroadcast() fra læreboka. En tråd vil også bli vekket opp fra wait() hvis den blir avbrutt av et interrupt()-kall. Tråder som blir vekket fra en wait() på grunn av et kall til notify(), notifyAll() eller interrupt() vil måtte konkurrere om å få tak i monitoren til objektet på nytt før de kan fortsette kjøringen sin. Dette sikrer at maksimalt én tråd til enhver tid kjører kode som er synchronized.

Funksjonene wait(), notify() og notifyAll() er implementert i Object klassen og kan dermed brukes i alle klasser, også egendefinerte. Alle Javaklasser arver fra Object enten direkte eller indirekte. En klasse som ikke arver noe som helst har en implisitt ”extends java.lang.Object” satt inn. Alle klasser som arver fra en annen klasse vil arve funksjonene fra Object gjennom den.



Figur 1: Tilstandsdiagram for tråder i Java

Figur 1 er en versjon av figur 3.9 på side 122 i læreboka som viser hvordan de ulike funksjonskallene påvirker tilstanden til en tråd i Java. Stop() er ikke vist - og avslutter en tråd uansett hvilken tilstand den er i.

Eksempel på synkronisering

Her er et enkelt og trivielt eksempel på en applikasjon med to tråder og et behov for gjensidig utelukkelse. Begge trådene skriver ting til skjermen, men kun en av dem skal kunne skrive til skjermen på en gang. Dette løses ved at en tråd må ha monitoren til Skjerm-objektet for å kunne skrive til skjerm.

Skriver.java

```
public class Skriver extends Thread
{
   // To ord som skal skrives til skjermen av denne skriveren
   private String forsteOrd, andreOrd;
   // Referanse til Skjerm-objektet
   private Skjerm skjerm;

   // Oppretter et nytt Skriver-objekt
   public Skriver(Skjerm skjerm, String forsteOrd, String andreOrd) {
      super();
      this.skjerm = skjerm;
      this.forsteOrd = forsteOrd;
      this.andreOrd = andreOrd;
   }

   // Metoden som utføres av denne Skriver-tråden
   public void run() {
      for(int i = 0; i < 5; i++) {
         // Skriv ut de to ordene
         skjerm.skrivUt(forsteOrd, andreOrd);
         // Vent noen tidels sekunder
         try {
            Thread.sleep((int)(Math.random()*700));
         } catch (InterruptedException e) {
            // Kommer hit hvis noen kalte interrupt() på denne tråden.
            // Det skjer ikke i denne applikasjonen.
         }
      }
   }

   // Metoden som blir kjørt når applikasjonen starter
   public static void main(String[] args) {
      // Opprett skjerm-objektet
      Skjerm skjerm = new Skjerm();
      // Start to skrivertråder som skriver ut ordene som ble gitt inn fra kommandolinja
      Skriver skriver1, skriver2;
      skriver1 = new Skriver(skjerm, args[0], args[1]);
      skriver1.start();
      skriver2 = new Skriver(skjerm, args[2], args[3]);
      skriver2.start();
   }
}
```

Skjerm.java

```
public class Skjerm
{
   // Siden denne funksjonen er synchronized,
   // kan bare en tråd være inne i den av gangen
   public synchronized void skrivUt(String ord1, String ord2) {
      System.out.print(ord1);
      try {
         Thread.sleep(100);
      } catch (InterruptedException e) {}
      System.out.println(ord2);
   }
}
```

Resultat av kjøringen

Her er en utskrift fra kjøring av eksempelapplikasjonen:

```
java Skriver kake spade bamse mums
kakespade
bamsemums
kakespade
bamsemums
bamsemums
kakespade
kakespade
bamsemums
kakespade
bamsemums
```

Og her er en utskrift av det samme programmet, der skrivUt-metoden ikke er synchronized:

```
java Skriver kake spade bamse mums
kakebamsespade
mums
bamsekakemums
spade
kakebamsespade
mums
kakespade
bamsemums
bamsekakemums
spade
```

4 Teoretiske øvingsoppgaver fra læreboka

I dette kapitlet identifiseres de frivillige teoriøvingene (TA-TB). Hver av disse øvingene er hentet fra læreboka.

Disse øvingene dekker henholdsvis lærebokas kapitler 6 & 7 og 10 & 11.

Lærebokas kapitler 5, 9 og 8 & 12 dekkes av praksisøvingene som er presentert i neste kapittel.

4.1 TA – Prosesser & Lager – Kap. 6 & 7

Dette dekkes av følgende oppgaver:

- Oppgave 6.6
- Oppgave 6.14
- Oppgave 6.16

- Oppgave 7.5
- Oppgave 7.7
- Oppgave 7.11

4.2 TB – Tidsstyring & Innlesing/Utskrift – Kap. 10 & 11

Dette dekkes av følgende oppgaver:

- Oppgave 10.2
- Oppgave 10.7

- Oppgave 11.6
- Oppgave 11.7
- Oppgave 11.8
- Oppgave 11.9

5 Praktiske øvingsoppgaver

I dette kapitlet finnes tekstene til de fire praksisøvingene (P1-P4). Hver av disse øvingene er omtalt ut fra følgende seks tema: Overordnet formål, detaljert beskrivelse, nyttige tips, tilgjengelig delløsning, eksakt oppgave, samt senere tilleggsløsning.

Disse øvingene dekker henholdsvis lærebokas kapitler 5 (P1), 5 (P2), 9 (P3) og 8 & 12 (P4). Øvingene P1 og P4 er frivillige, mens øvingene P2 og P3 er obligatoriske.

Lærebokas kapitler 6 & 7 og 10 & 11 dekkes av de frivillige teoriøvingene som er presentert i forrige kapittel.

5.1 P1 – Gjensidig utelukkelse – Kap. 5

Overordnet formål

Her skal du studere et tilfelle der gjensidig utelukkelse (mutual exclusion) er nødvendig for å få korrekt oppførsel i et enkelt program. Du må selv identifisere koden som har behov for gjensidig utelukkelse, den såkalte kritiske koden. Du må kunne forklare hvorfor koden er kritisk og vise hvordan manglende gjensidig utelukkelse kan føre til feil. Oppgaven omfatter problemstillinger tilsvarende lærebokas kapittel 5.

Beskrivelse av oppgaven

Du skal studere et enkelt program bestående av en bank og en rekke kurerer. Kurerene henter penger hos butikker og setter pengene inn i banken. Banken tar i mot penger i nattsafen, og når det er for mye penger i nattsafen overføres alt som ligger i nattsafen til hvelvet. I løpet av en kjøring sender banken ut en rekke kurerer, som så alle gjør en rekke innskudd i banken. Ved slutten av kjøringen viser det seg imidlertid at det noen ganger mangler penger i banken i forhold til hvor mye kurerene hevder at de har satt inn. Andre ganger er det til og med for mye penger i banken. Din oppgave er å finne ut hvorfor og løse problemet.

Tilgjengelig delløsning

Her er koden utdelt, og du skal analysere den. Du vil måtte gjøre minimalt med endringer for å få koden til å fungere korrekt.

Nyttige tips

Det kan være greit å lese informasjonen om tråder og synkronisering som er gitt i seksjon 3.2. Du bør også lese første del av kapittel 5 i boka. Gjensidig utelukkelse implementeres i Java ved hjelp av nøkkelordet synchronized.

Eksakt oppgave

- Analyser koden og finn ut hvorfor den ikke fungerer. Rett opp feilen(e).
- Sammenlign kjøretiden til den rettede løsningen og den opprinnelige koden. Forklar forskjellen.
- Vis eksempler på situasjoner der den feilaktige koden kan føre til for mye penger i banken og for lite penger i banken.
- Lever inn rettet kode og ovennevnte eksempler.

Utdelt kode

```
public class Bank {
   private int nightSafeAmount;
   private int vaultAmount;
   private Courier[] couriers;
   private long startTime;
   private int nofActiveCouriers;

   /**
    * Creates a new bank, with a number of couriers, and
    * starts all the couriers.
    */
   public Bank(int nofCouriers, int nofVisits) {
      startTime = System.currentTimeMillis();
      couriers = new Courier[nofCouriers];
      nofActiveCouriers = nofCouriers;
      for(int i = 0; i < nofCouriers; i++) {
         couriers[i] = new Courier(nofVisits, this);
         couriers[i].start();
      }
   }

   /**
    * Deposit a given amount of money. The money is put in the
    * night safe. If there is too much money in the night safe,
    * all the money is transferred to the vault.
    */
   public void depositMoney(int amount) {
      nightSafeAmount += amount;
      if(nightSafeAmount > 200) {
         // Transfer the money in the night safe to the vault:
         vaultAmount += nightSafeAmount;
         nightSafeAmount = 0;
      }
   }

   /**
    * Called by a courier when he is done with his deposits.
    */
   public void courierDone() {
      nofActiveCouriers--;
      if(nofActiveCouriers == 0) {
         System.out.println("All couriers are done.");
         // Check if the money deposited equals the money in the bank:
         int moneyInBank = nightSafeAmount + vaultAmount;
         int moneyDeposited = calculateMoneyDeposited();
         System.out.println("Money in the bank: "+moneyInBank);
         System.out.println("Total money deposited: "+moneyDeposited);
         System.out.println("Discrepancy: "+(moneyDeposited-moneyInBank));
         System.out.println("Elapsed time: "+(System.currentTimeMillis()-startTime)+
            " milliseconds.");
      }
   }
```

```
    /**
     * Sum up all the reported deposits to find out how much
     * the couriers claim that they have deposited.
     */
    public int calculateMoneyDeposited() {
        int result = 0;
        for(int i = 0; i < couriers.length; i++)
            result += couriers[i].getAmountDeposited();
        return result;
    }

    /**
     * Reads the number of couriers and the number of visits to the
     * bank for each courier from the command line parameters, and
     * starts the simulation.
     */
    public static void main(String[] args) {
        if(args.length < 2) {
            System.out.println("Usage: java Bank <number of couriers> <number of visits>");
            System.exit(0);
        }
        int nofCouriers = new Integer(args[0]).intValue();
        int nofVisits = new Integer(args[1]).intValue();
        System.out.println("Starting simulation of "+nofCouriers+" couriers with "+nofVisits+
            " deposits each.");
        Bank b = new Bank(nofCouriers, nofVisits);
      }
}

public class Courier extends Thread {
    private Bank bank;
    private int nofVisits;
    private int moneyDeposited;

    public Courier(int nofVisits, Bank bank) {
        this.nofVisits = nofVisits;
        this.bank = bank;
    }

    /**
     * Visit the bank nofVisits times, and each time
     * deposit between 50 and 149 dollars.
     * Keep records of how much we have deposited, as the
     * bank has been known to keep faulty records.
     */
    public void run () {
        for(int i = 0; i < nofVisits; i++) {
            int sum = 50+(int)(Math.random()*100);
            bank.depositMoney(sum);
            moneyDeposited += sum;
        }
        bank.courierDone();
    }

    public int getAmountDeposited() {
        return moneyDeposited;
    }
}
```

5.2 P2 – Synkronisering av tråder – Kap. 5

Overordnet formål

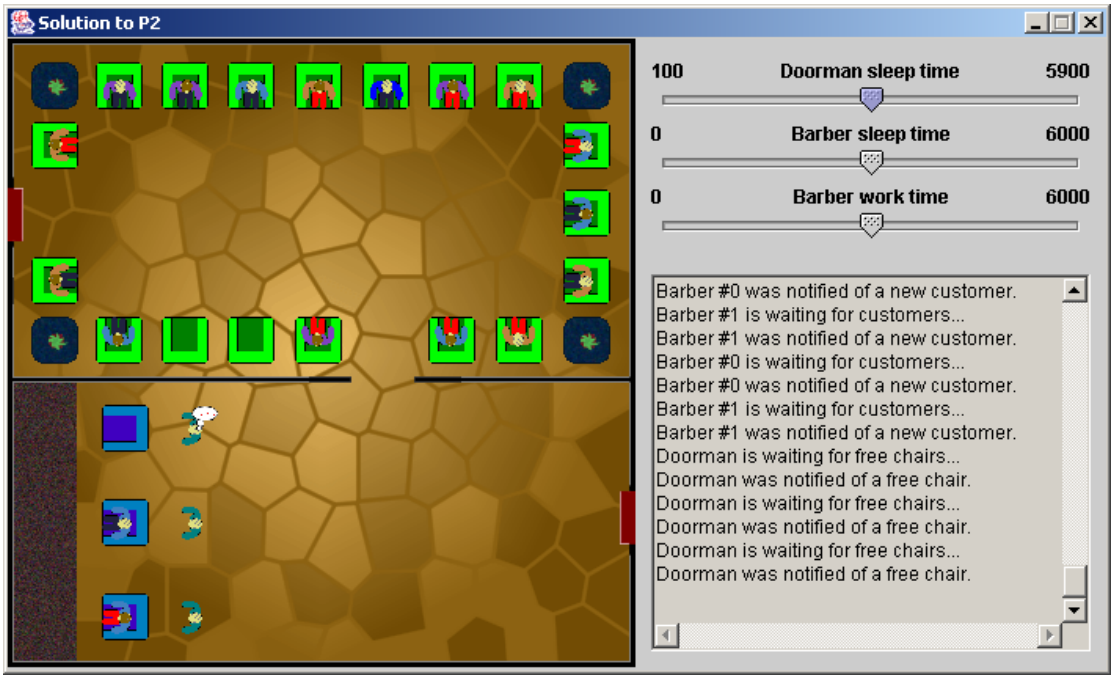
Her skal du implementere en variant av frisørsalongproblemet beskrevet i boka. Du skal bruke en produsent/konsument-modell. Produsent og konsument synkroniseres ved hjelp av metodene wait() og notify(), og gjensidig utelukkelse implementeres via Javas monitorkonsept; dvs. synchronized funksjoner. Oppgaven omfatter problemstillinger tilsvarende lærebokas kapittel 5.

Beskrivelse av oppgaven

Produsenten din, en innkaster ved en frisørsalong, utgjøres av en tråd som periodisk prøver å føye en ny person til en buffer med stoler i frisørsalongen. Med periodisk menes at produsenten sover en viss tid mellom hver gang han prøver å slippe inn en ny person.
Produsenten og konsumenten deler en sirkulær buffer med stoler. Produsenten plasserer en person i neste tilgjengelige stol, mens konsumenten alltid tar den personen som har ventet lengst.
Konsumentene dine er frisører, og frisørsalongen kan ha opp til tre frisører arbeidende på en gang. Hver frisør utgjør sin egen tråd. En frisering tar en viss tid. Mellom hver frisering dagdrømmer frisøren i tilfeldig lang tid innen gitte rammer, før han er klar til å klippe en ny kunde.

Tilgjengelig delløsning

Du vil få utdelt en GUI som gjør det lettere å visualisere hvordan synkroniseringen fungerer. Et skjermbilde fra GUI'en er vist under. Stolene øverst på bildet visualiserer den sirkulære bufferen. Nederst ses de tre frisørene. Tankeboblen betyr at frisørene dagdrømmer. Hvis frisøren ikke har tankeboble og ikke har en kunde i stolen, venter han på at det skal komme kunder. Til høyre er det et tekstområde der dere kan skrive ut ytterligere informasjon. Det er også tre slidere som stiller på hvor lenge de forskjellige aktørene sover og hvor lang tid en klipp tar. Innkasteren er ikke synlig men gjør seg gjeldende ved at det stadig kommer nye kunder.



Det er syv metoder i GUI'en som dere kan kalle på passende tidspunkter for å holde GUI'en oppdatert. Disse metodene er definert i grensesnittet Gui, og kan brukes av klassen Doorman, CustomerQueue og Barber til å oppdatere GUI'en etter hvert som ting skjer.
Les kommentarene i den utdelte koden for å se nøyaktig hva disse metodene gjør. I tillegg bør dere lese metoden startSimulation() i klassen BarbershopGui, som starter opp simuleringen. For at programmet skal kompilere og kjøre er det skissert et rammeverk til klassene klassene Doorman, Barber og CustomerQueue. Det er disse klassene dere må utvide for å få til en ferdig løsning.
Det er tre variabler i klassen Globals som angir hvor lenge de forskjellige aktørene skal sove og jobbe. Når sliderene i GUI'en skyves på, vil disse variablene endres. Referer derfor alltid til disse variablene

når dere skal avgjøre hvor lenge en produsent eller konsument skal sove. Soving kan gjøres med metodekallet Thread.sleep(). For å gi en litt mer spennende kjøring kan dere legge inn tilfeldige variasjoner i hvor lang tid aktørene sover, ved hjelp av metoden Math.random().

Nyttige tips

Her vil du måtte bruke den informasjonen om tråder og synkronisering som er gitt i seksjon 3.2.
Du bør ikke fysisk flytte kunder fra stol til stol etter som de rykker framover i køen (dette ville heller ikke være normalt i en vanlig frisørsalong). Bruk heller bufferpekere til å holde styr på begynnelsen og slutten av køen.

Når du skal bestemme hvor lenge innkasteren eller frisøren sover har du behov for å generere tilfeldige tall. En enkel måte å generere et tilfeldig tall fra min til max på er:
int r = min+(int)(Math.random()*(max-min+1));

Rammeverket til øvinga er ferdig utdelt. Dere må lage logikken. Klassene som må gjøres ferdige er: Doorman, CustomerQueue og Barber.

Eksakt oppgave

Følgende forventes som resultat av denne øvingen:

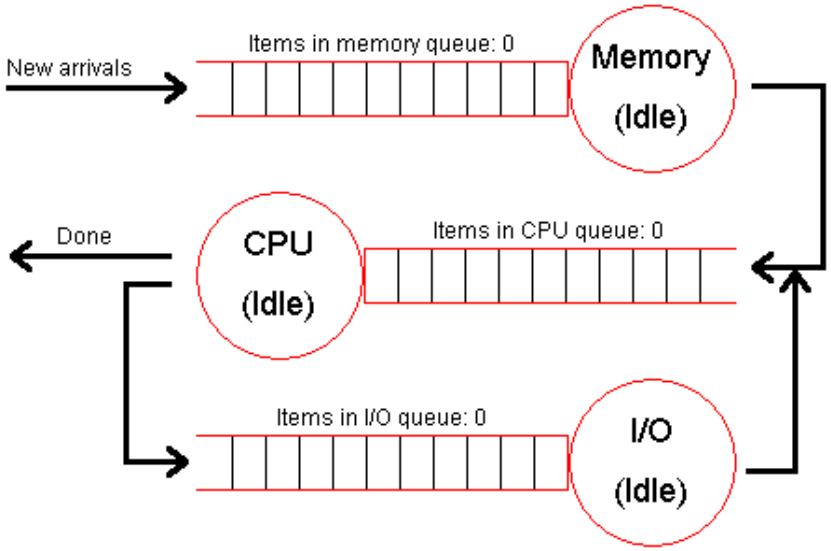
- Innlevering av kildekode til programmet
- Et ark med kommentarer som kort forklarer hvilke java-mekanismer du har brukt til å implementere Producer/Consumer-modellen.
- Demonstrasjon av programmet for undervisningsassistent / studentassistent.

5.3 P3 – Tidsstyring av prosesser – Kap. 9

Overordnet formål

Her skal dere implementere en form for tidsstyring av prosesser ved å simulere bruk av Round-Robin algoritmen i et forenklet datamaskinsystem. Oppgaven omfatter problemstillinger tilsvarende lærebokas kapittel 9.

Betrakt en enkel datamaskin med et enkelt CPU, et lite lager samt en I/O enhet. Datamaskinen behandler prosesser som utfører batchjobber med statisk minnebehov. Det finnes bare en type lager (minne), altså ingen form for virtuelt minne. Datamaskinen er modellert som et køsystem som vist i figuren under, der prosesser ankommer med jevne mellomrom og forsvinner ut av systemet når de er ferdig med jobben sin.



Det er tre køer i systemet:

- En kø for prosesser som venter på å få tildelt minne.
- En kø for prosesser som venter på prosessortid.
- En kø for prosesser som venter på å få utføre I/O.

Utnyttelsen av CPU-kraften styres med Round Robin-algoritmen. De andre køene opereres etter FIFO-prinsippet. Jobbene og deres behov genereres tilfeldig.

Detaljert beskrivelse

Programmet som skal lages, må først etterspørre en del parametere fra brukeren for å kontrollere simuleringen. Dette gjelder:

- Tilgjengelig lagerstørrelse (for eksempel 2048 KB)
- Tidskvant for Round Robin (for eksempel 500 ms)
- Gjennomsnittlig I/O operasjontid (for eksempel 225 ms)

- Total simuleringstid (for eksempel 250.000 ms)
- Gjennomsnittlig tid mellom ankomst for nye jobber (5.000 ms)

For hver ny jobb som skal kjøres, må systemet videre generere tilfeldige parameterverdier for de tre karakteristikaene:

- Prosessens lagerbehov (for eksempel 100 KB). Lagerbehov skal variere mellom 100 KB og 25% av tilgjengelig lagerstørrelse.
- Prosessens eksekveringstid (for eksempel 5.000 ms). Eksekveringstid skal variere mellom 100 ms og 10.000 ms.

- Prosessens gjennomsnittlige eksekveringstid mellom I/O-behov (for eksempel 250 ms). Gjennomsnittlig tid mellom I/O-behov skal variere mellom 1% og 25% av prosessens eksekveringstid.

Den tilfeldige tallgeneratoren må også benyttes for følgende tre andre forhold: Tid fram til neste I/O-behov hver enkelt gang (variert rundt tilsvarende gjennomsnittlige tid), tid for neste I/O-behov hver enkelt gang (variert rundt tilsvarende gjennomsnittlige tid), samt tid til neste jobb-ankomst hver enkelt gang (variert rundt tilsvarende gjennomsnittlige tid).

En nyankommet jobb plasseres i lagerkøen. Her venter prosesser på tilstrekkelig lagerplass. Køen administreres ut fra FIFO prinsippet. En gitt prosess trenger ikke å vente på at dens lagerbehov kan dekkes som en enkelt minneblokk, bare på at dens lagerbehov kan dekkes totalt sett. Virtuelt lager eller swapping brukes ikke. Med andre ord trenger lageret kun å holde rede på hvor mye minne som er ledig til enhver tid.

Når en prosess får tildelt lagerplass, flyttes den direkte over i CPU-køen. Selve minnetildelingen tar ingen tid, så minne-enheten vil til enhver tid vises som ”Idle”, men minnekøen kan være lang. I CPU-køen venter prosesser på CPU-kraft. Denne køen administreres altså ut fra Round Robin prinsippet.

Når en prosess får tildelt CPU-en, beholder den kontrollen inntil ett av tre forhold skjer: Når tidskvantet spesifisert av RR-algoritmen er oppbrukt – hvorpå prosessen flyttes tilbake i CPU-køen, når et I/O-behov oppstår – hvorpå prosessen flyttes over til en egen I/O-kø, eller når prosessen er ferdig – hvorpå prosessens lagerplass tilbakeleveres og den forlater systemet.

En prosess som står i den egne I/O-køen, flyttes derfra og over til CPU-køen igjen etter tilhørende I/O-avslutning.

Når simuleringen er avsluttet – dvs. når simuleringstiden er ute selv om det fortsatt kan være prosesser som ikke er ferdige, skal programmet rapportere følgende:

- Antall ferdige prosesser
- Antall opprettede prosesser
- Antall prosessskifter (som skyldes oppbrukt tidskvant)
- Antall utførte I/O-operasjoner
- Gjennomsnittlig gjennomstrømning (ferdige prosesser per sekund)

- Total tid CPU har brukt på å prosessere
- Total tid CPU har vært ledig
- Prosentvis hvor mye tid CPU har brukt på å prosessere (utilisasjon)
- Prosentvis hvor lenge CPU har vært ledig
- Størst forekommende samt gjennomsnittlig lengde på alle køer
- Hvor mange ganger en ferdige prosess gjennomsnittlig har blitt plassert i hver enkelt kø

- Gjennomsnittlig tid tilbrakt i systemet per ferdige prosess
- Gjennomsnittlig tid ventende på tilstrekkelig lagerplass per ferdige prosess
- Gjennomsnittlig tid ventende på CPU-kraft per ferdige prosess
- Gjennomsnittlig tid brukt i CPU per ferdige prosess
- Gjennomsnittlig tid ventende på I/O-kapasitet per ferdige prosess
- Gjennomsnittlig tid brukt i I/O per ferdige prosess

Slik rapportering krever at ulike verdier tas vare på underveis i simuleringen. Resultatene avhenger mye av hvilke parametere som brukes i simuleringen, og dette åpner opp for ulik eksperimentering for å forbedre systemet.

Nyttige tips

Denne oppgaven er et eksempel på såkalt diskret hendelsessimulering. To hovedkomponenter vil være en klokke og en hendelseskø. Klokken vil holde oversikt over forløpt tid i simuleringen, mens hendelseskøen vil holde styr på framtidige hendelser i simuleringen. Eksempler på hendelser i denne sammenhengen er: Ankomst av en ny jobb, utløp av et tidskvant, oppstart av en I/O-operasjon, avslutning av en I/O-operasjon og terminering av en eksisterende jobb. Framtidige hendelser er vanligvis sortert på tid i hendelseskøen.

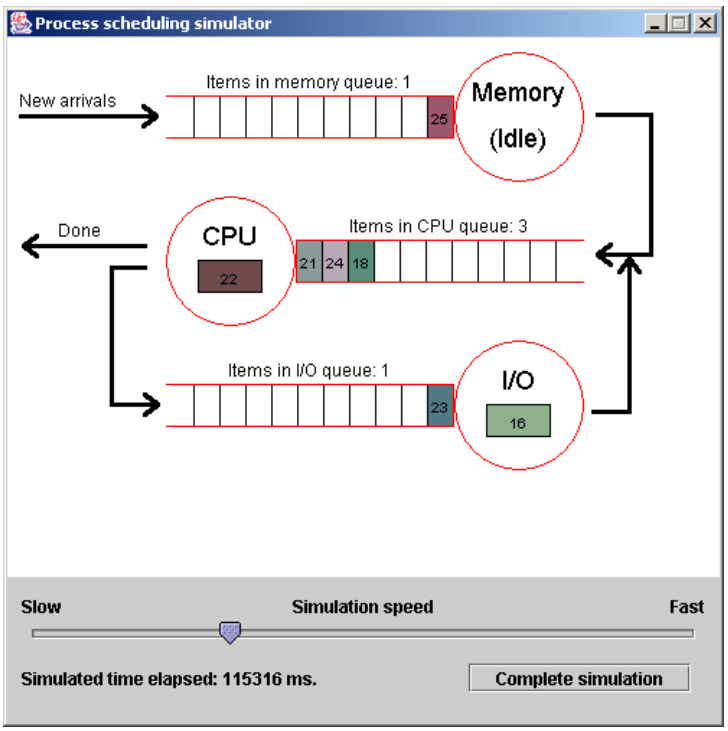
Hovedløkken i simuleringen tar hele tiden første hendelse ut av hendelseskøen, oppdaterer tiden tilsvarende den nye hendelsen og behandler hendelsen i seg selv. Således hopper tiden hele tiden framover i diskrete steg. Hvis flere hendelser skal skje på samme tid, vil de håndteres i rekkefølge uten at tiden forandrer seg. Behandlingen av en hendelse kan medføre at nye hendelser skapes og legges inn i hendelseskøen på riktig sted.

Som en liten pekepinn er kjøreutskriften fra et korrekt fungerende program vist under:

```
Please input system parameters:
Memory size (KB): 2048
Maximum uninterrupted cpu time for a process (ms): 500
Average I/O operation time (ms): 225
Simulation length (ms): 250000
Average time between process arrivals (ms): 5000
Simulating.....done.

Simulation statistics:

Number of completed processes:                                39
Number of created processes:                                  48
Number of (forced) process switches:                          207
Number of processed I/O operations:                           753
Average throughput (processes per second):                    0.156

Total CPU time spent processing:                              199213 ms
Fraction of CPU time spent processing:                        79.6852%
Total CPU time spent waiting:                                 50787 ms
Fraction of CPU time spent waiting:                           20.3148%

Largest occuring memory queue length:                         2
Average memory queue length:                                  0.13544
Largest occuring cpu queue length:                            6
Average cpu queue length:                                     1.753956
Largest occuring I/O queue length:                            6
Average I/O queue length:                                     1.121616
Average # of times a process has been placed in memory queue: 1
Average # of times a process has been placed in cpu queue:    20.333334
Average # of times a process has been placed in I/O queue:    14.256411

Average time spent in system per process:                     22772 ms
Average time spent waiting for memory per process:            657 ms
Average time spent waiting for cpu per process:               10104 ms
Average time spent processing per process:                    4465 ms
Average time spent waiting for I/O per process:               4329 ms
Average time spent in I/O per process:                        3215 ms
```

Tilgjengelig delløsning

For å gjøre arbeidsomfanget mindre, vil vi her ferdigstille utvalgte moduler av det som etterspørres. Vi har laget en GUI som viser køsystemmodellen og hvor i systemet prosessene befinner seg. Et skjermbilde fra GUI'en er vist under.



GUI'en har et par metoder som må kalles på passende tidspunkter for at simulasjonens oppførsel skal vises i GUI'en. Disse metodene er deklarert i grensesnittet Gui, som deles ut. GUI'en er implementert av de utdelte klassene SimulationGui, PicturePanel, Resource og Constants. Når det gjelder selve simuleringen er klassene Event, EventQueue og Memory ferdig implementerte, mens oppstartsklassen Simulator og klassen Process er delvis implementert. Dere må ferdigstille resten av systemet, inkludert en klasse som simulerer CPU'en og en som simulerer I/O.

For å gjøre koden mest mulig ryddig anbefaler vi å samle flest mulig av variablene som fører statistikk med kjøringen på ett sted, gjerne i en separat klasse.

Kode for, utfyllende beskrivelse av og kommentar til de utleverte modulene vil bli gjort tilgjengelig dels via hjemmesiden, dels som papirkopier og dels i øvingstimene.

Eksakt oppgave

Følgende forventes som resultat av denne øvingen:

- Innlevering av kildekode til programmet – inkludert en kommentarseksjon som klart angir hvordan du vil eksperimentere med Round Robin algoritmen for å forbedre systemet.
- Demonstrasjon av programmet for undervisningsassistent / studentassistent.
- Innlevering av utskrift fra kjøringen – inkluderende en resultatseksjon som detaljert angir effektene av Round Robin eksperimenteringen. Prøv å identifisere nøkkelparametere som har stor innvirkning på systemets totale ytelse.

5.4 P4 – Virtuell filhåndtering – Kap. 8 & 12

Overordnet formål

Her skal dere implementere en mekanisme for virtuell filhåndtering; dvs. en måte å skape et virtuelt adresserom til å aksessere en fil med. En virtuell fil består av blokker som ligger lagret på disk og caches i minne. Denne blokkdelingen er imidlertid usynlig for applikasjoner, som kun ser et sammenhengende adresserom. Dette er analogt med virtuelt minne der minnet er organisert i sider, men prosesser ser et sammenhengende adresseområde. Filhåndteringsmekanismen skal således bruke et sett med sider og en tilhørende sideombyttingsalgoritme i forbindelse med filaksessering. Oppgaven omfatter problemstillinger tilsvarende lærebokas kapitler 8 og 12.

Sett fra brukerens synspunkt vil en fil bestå av en sekvens med bytes, adressert fra 0 og oppover. Enhver lese- / skriveaksess angir første byte og antall byte involvert. Hvis tilsvarende sider ikke er tilgjengelige, genereres en sidefeil som så håndteres for å bringe inn de nødvendige sidene.

Sett fra systemets side består en fil av et fast antall sider som angis når filen skapes. Som sideombyttingsalgoritmer skal FIFO, LRU og CLOCK brukes.

Detaljert beskrivelse

Den virtuelle filhåndtereren trenger bare å administrere en åpen fil om gangen. Funksjonaliteten som skal tilbys applikasjoner er spesifisert av følgende java-grensesnitt:

```
public interface VirtualFileSystem
{
   /**
    * Initializes the virtual file system.
    * @param pageSize  The size of pages (file blocks).
    * @param nofFrames The number of page frames in the system.
    */
   public void init(int pageSize, int nofFrames);

   /**
    * Creates a new file with the given name and size.
    * @param filename   A unique name identifying the file.
    * @param nofPages   The size of the file, in pages.
    */
   public void create(String filename, int nofPages);

   /**
    * Opens a file created by the create() method.
    * @param filename   The name of the file to be opened.
    */
   public void open(String filename);

   /**
    * Reads a sequence of bytes from the open file.
    * @param address   The position of the first byte to be read, relative
    *                  to the first byte in the file.
    * @param nofBytes  The size number of bytes to be read.
    * @param data      The buffer in which to place the read bytes.
    */
   public void read(int address, int nofBytes, byte[] data);
```

```
    /**
     * Writes a sequence of bytes to the open file.
     * @param address   The position of the first byte to be written, relative
     *                  to the first byte in the file.
     * @param data      A buffer of bytes to be written to the file.
     */
    public void write(int address, byte[] data);

    /**
     * Closes the file opened by the open() method.
     */
    public void close();

    /**
     * Specifies which page replacement algorithm to be used.
     *
     * @param chooser   An object, implementing the FrameChooser interface,
     *                  specifying the page replacement algorithm.
     */
    public void setFrameChooser(FrameChooser chooser);

    /**
     * Prints the contents of the frame table in a compact but readable form
     * to System.out. It suffices to list the indexes of the pages currently
     * in the table, and the corresponding clock bits.
     */
    public void printFrameTable();
}
```

Metoden setFrameChooser gir inn et objekt som implementerer grensesnittet FrameChooser vist under:

```
/**
 * An interface specifying methods for selecting which frame in
 * a frame table should be chosen when the need to replace a
 * page arises.
 */
public interface FrameChooser
{
    /**
     * Selects a replaceable frame from a full frame table.
     *
     * @param frameTable  The frame table.
     * @return            The frame whose page can be replaced.
     */
    public Frame chooseFrame(FrameTable frameTable);

    /**
     * Returns the number of page faults processed.
     *
     * @return  The number of times the chooseFrame() method has been invoked.
     */
    public int getNofPageFaultsProcessed();

    /**
     * Returns a string describing the algorithm used by this FrameChooser.
     *
     * @return  The name of the page replacement algorithm used.
     */
    public String getAlgorithm();
}
```

Implementasjoner av dette grensesnittet, samt klassen FrameTable, må dere lage selv. Dere må også lage en implementasjon av grensesnittet VirtualFileSystem. For å ta seg av selve skrivingen til og fra disk kan dere benytte dere av den utdelte klassen DiskManager.

Nyttige tips

Konkrete tips til hva de forskjellige metodene i implementasjonen av VirtualFileSystem må gjøre er listet opp under:

- init(): Denne metoden tar inn to parametere som spesifiserer hvor mange siderammer filsystemet har og hvor stor (antall bytes) hver side er. Sidestørrelsen må være lik blokkstørrelsen brukt i DiskManager. Konstruktoren bør opprette en rammetabell med siderammer, og en standard FrameChooser bør opprettes i tilfelle applikasjonen ikke spesifiserer en algoritme ved et senere kall til setFrameChooser().

- create(): Denne metoden kan benytte seg av create-metoden i klassen DiskManager.

- open(): Dere må opprette en sidetabell for filen. Størrelsen på denne avhenger av antall sider i filen. Initielt opptar filen ingen siderammer. Tilsvarende sider hentes etterhvert inn i siderammer når det oppstår sidefeil i.f.m. aksessering av filen. Hvis alle siderammene er opptatte når det oppstår en sidefeil må FrameChooser-objektet brukes til å velge en side som kan erstattes. Siden som erstattes må skrives ut til disk hvis den er endret.

- read(): Adresseområdet som skal leses må oversettes til sideaksessoperasjoner. For hver side som skal aksesseres må det sjekkes om siden ligger i rammetabellen. Hvis den ikke gjør det, oppstår det en sidefeil som må behandles før leseoperasjonen kan fortsette. Sideaksessoperasjoner kan utføres ved hjelp av metoder i DiskManager.

- write(): Skriving til fil fungerer veldig likt som lesing fra fil, og kan også generere sidefeil. Her bør dere utnytte de muligheter til gjenbruk som finnes. Modifiserte sider i rammetabellen skal ikke skrives ut til disk før filen lukkes, eller sidene blir erstattet.

- close(): Når en fil lukkes må sider i rammetabellen som er endret skrives til disk. Alle siderammene frigjøres, og sidetabellen slettes.

Hver sideramme skal altså inneholde et bit som angir om tilhørende side er endret eller ikke siden den ble innlest. Dette vil avgjøre om den må skrives tilbake til disk når den blir ombyttet med en annen side. Hver sideramme bør også inneholde et bit (eller noe lignende) som angir om tilhørende side er aksessert ”siden sist”.

Dere trenger åpenbart klasser for sider og siderammer, og implementasjoner av FrameChooser-grensesnittet som implementerer de forskjellige sideerstattingsalgoritmene. Det kan også være nyttig å lage egne klasser med tilhørende metoder for sidetabellen og rammetabellen.

Hvis dere forsøker å kompilere den utdelte koden, vil dere få en rekke ”cannot resolve symbol”-feilmeldinger. Disse indikerer noen klasser som må implementeres.

Tilgjengelig delløsning

Deler av løsningen er ferdig utdelt. Dette omfatter grensesnittene som skal implementeres, en klasse som tester ut filsystemet, og DiskManager-klassen. Kode for, utfyllende beskrivelse av og kommentar til disse modulene vil igjen bli gjort tilgjengelig dels via hjemmesiden, dels som papirkopier og dels i øvingstimene.

Eksakt oppgave

Følgende forventes som resultat av denne øvingen:

- Innlevering av kildekode til programmet
- Overordnet beskrivelse av hvordan programmet er bygd opp og fungerer
- Demonstrasjon av programmet for undervisningsassistent / studentassistent
- Innlevering av utskrift fra kjøringen, med diskusjon av de forskjellige sideombyttingsalgoritmenes fordeler og ulemper.

Senere tilleggsløsning

Her vil det ikke bli behandlet noen utvidelser av den oppgitte problemstillingen. Men i våremnet TDT4190 Distribuerte Systemer vil det bli gitt en oppgave på å lage en distribuert versjon av dette filsystemet.

6 Oppgaver – gamle eksamenssett

I dette kapitlet finnes oppgavene for åtte gamle eksamenssett i emnet. Tilsvarende løsninger finnes i neste kapittel. Høst 2006 og Sommer 2007 er ikke inkludert fordi emnet da ble gitt av andre.

De inkluderte eksamenssettene gjelder:

Høst 2003
Sommer 2004
Høst 2004
Sommer 2005
Høst 2005
Sommer 2006
Høst 2007
Sommer 2008

Emnet – inkludert en datamaskinarkitektur del istedenfor en distribuert system del – hadde betegnelsen TDT4155 f.o.m. høst 2003 t.o.m. sommer 2004.

6.1 Høst 2003

6.2 Sommer 2004

6.3 Høst 2004

6.4 Sommer 2005

6.5 Høst 2005

6.6 Sommer 2006

6.7 Høst 2007

6.8 Sommer 2008

7 Løsninger - gamle eksamenssett

I dette kapitlet finnes løsningene for åtte gamle eksamenssett i emnet. Tilsvarende oppgaver finnes i forrige kapittel. Høst 2006 og Sommer 2007 er ikke inkludert fordi emnet da ble gitt av andre.

De inkluderte eksamenssettene gjelder:

Høst 2003
Sommer 2004
Høst 2004
Sommer 2005
Høst 2005
Sommer 2006
Høst 2007
Sommer 2008

Emnet – inkludert en datamaskinarkitektur del istedenfor en distribuert system del – hadde betegnelsen TDT4155 f.o.m. høst 2003 t.o.m. sommer 2004.

7.1 Høst 2003

7.2 Sommer 2004

7.3 Høst 2004

7.4 Sommer 2005

7.5 Høst 2005

7.6 Sommer 2006

7.7 Høst 2007

7.8 Sommer 2008

8 Lysark - kopisett

I dette kapitlet finnes kopier av seks sett med lysark for lærebokens Kap. 1-12. Disse er laget av forleseren i emnet.

De gjelder altså:

- Introduksjon / Bakgrunn (Kap. 1-2)
- Bruk av CPU (Kap. 3-4)
- Forhold mellom prosesser (Kap. 5-6)
- Bruk av lager (Kap. 7-8)
- Kjøring av prosesser (Kap. 9-10)
- Bruk av I/O (Kap. 11-12)

På emnets hjemmeside finnes kopier av tre sett med lysark for lærebokens Kap. 13-15. Disse er laget for forfatteren av boken.

De gjelder altså:

- Innebygde systemer (Kap. 13)
- Sikkerhet - Trusler (Kap. 14)
- Sikkerhet - Teknikker (Kap. 15)

8.1 Introduksjon / Bakgrunn

8.2 Bruk av CPU

8.3 Forhold mellom prosesser

8.4 Bruk av lager

8.5 Kjøring av prosesser

8.6 Bruk av I/O