Om eksamensoppgavene

Emnekode og -navn: ITF22519 Innføring i operativsystemer
Dato og tid: 20.12.2022, varighet 4 timer
Fagansvarlige: Nga Dinh og Jan Høiberg
Tillatte hjelpemidler: Bruk av ikke-kommuniserende kalkulator er tillatt
Sensurfrist: 10.01.2023**,** resultater blir publisert i Studentweb.

Oppgavesettet består av i alt tolv oppgaver i denne rekkefølgen:

- Åtte oppgaver med spørsmål fra ulike deler av pensum. Hver av disse åtte oppgavene inneholder fem deloppgaver. De fleste deloppgavene er flervalgsoppgaver, der det skal velges ett riktig svar fra flere svaralternativer. I noen deloppgaver skal det angis et tall som svar. Det gis ett poeng for hvert riktige svar og null poeng for et galt svar.
- En oppgave der det skal gis tallsvar.
- En oppgave der det skal gis tekstlige svar.
- To programmeringsoppgaver.

Hver av de fire siste oppgavene vektlegges med 10% i karaktersettingen av en eksamensbesvarelse. De åtte flervalgsoppgavene teller tilsammen 60%, fordelt likt på alle deloppgaver.

1 Oppgave 1: Generelt om operativsystemer

Deloppgave A
Nedenfor er det gitt fire beskrivelser av hva et operativsystem (OS) er. Hvilke(n) av dem er riktig(e)?

- Et OS er en samling med programvare som håndterer styringen av datamaskinen.
- Et OS er et mellomlag med programvare som skjuler kompleksiteten og dynamikken i maskinen for brukere og applikasjoner.
- Et OS tilbyr abstraksjon vekk fra detaljer om maskinvare og systemprogramvare.
- Alle de tre beskrivelsene ovenfor er riktige.

Deloppgave B
Nedenfor er det listet opp fire ulike typer permanente lagringsmedia:

A. Magnetisk, roterende harddisk
B. Solid state drive (Flash-disk)
C. Optisk disk
D. Hullkort

Sett dem i riktig rekkefølge sortert etter hastighet på innlesing av data (raskest først, tregest sist):

- A B C D
- B A C D
- B D C A
- C B D A

Deloppgave C
Bare én av disse fire typene programvare er en del av selve operativsystemet. Hvilken?

- Shell
- Filsystem
- Kompilator
- Brukertrådbibliotek

Deloppgave D
Hva kalles denne metoden for å dele f.eks. en printer mellom flere brukere/prosesser: Filer som skal skrives ut legges først i en spesiell katalog, for senere å bli skrevet ut av en egen daemon-prosess som håndterer printeren og printerkøen?

- Race conditioning
- Multiprogramming
- Caching
- Spooling

Deloppgave E

Alderen på et operativsystem kan defineres som antall år siden første versjon av systemet ble gjort tilgjengelig og kunne installeres på en datamaskin. Nedenfor er det listet opp navnet på fire ulike operativsystemer:

A. Linux
B. Unix
C. Microsoft Windows
D. Multics

Sett disse operativsystemene i riktig rekkfølge, sortert stigende på alder (nyeste først, eldste sist).

- A B C D
- A C B D
- C A D B
- D C B A

Maks poeng: 7.5

2 Oppgave 2: Programmering i C

Deloppgave A

Nedenfor er det gitt fire utsagn om programmeringsspråket C. Bare ett av dem er riktig. Hvilket?

- Programmer skrevet i C vil vanligvis kjøre langsommere enn tilsvarende program skrevet i Java,
- I et C-program må all koden ligge på en og samme fil.
- C har et lavere abstraksjonsnivå og er nærmere maskinvaren enn Java.
- C bør ikke brukes til å programmere operativsystemer.

Deloppgave B

Nedenfor er det gitt fire utsagn om bruk av pekervariable i C-programmer. Bare ett av dem er ikke riktig. Hvilket?

- To eller flere pekervariable kan ikke legges sammen (adderes) eller trekkes fra hverandre (subtraheres).
- Pekervariable lagrer virtuelle minneadresser.
- Pekervariable gir mer effektiv parameteroverføring av store datamengder til funksjoner.
- Pekervariable må brukes for å kunne endre verdien på variablene som brukes som parametre når vi kaller på en funksjon.

Deloppgave C

Følgende C-program er gitt:

```
#include <stdio.h>
int main()
{
  int *p;
  int a = 0;
  int b = 1;
  p = &b;
  a = *p;
  printf("%d\n", ++a);
}
```

Hva skrives ut når programmet ovenfor kjøres?

- 0
- 1
- 2
- 3

Deloppgave D
Følgende C-program er gitt:

```
#include <stdio.h>

void bytt(int x, int y)
{
   int tmp = x;
   x = y;
   y = tmp;
}

int main()
{
   int a = 0, b=1;
   bytt(a, b);
   printf("a=%d, b=%d\n", a, b);
}
```

Hva skrives ut når programmet ovenfor kjøres?

- a=0, b=1
- a=1, b=0
- a=0, b=0
- a=1, b=1

Deloppgave E
Følgende C-program er gitt:

```
#include <stdio.h>
int main()
{
   int A[] = {0, 1, 2, 3};
   int *p = &A[3];
   while (p != A)
   {
      printf("%d ", *p);
      p--;
   }
   printf("\n");
}
```

Hva skrives ut når programmet ovenfor kjøres?

- 1 2 3
- 3 2 1 0
- 3 2 1
- 0 1 2 3

Maks poeng: 7.5

3 Oppgave 3: Prosesser og tråder

Deloppgave A

Prosesser og tråder kan være i en av tre mulige tilstander: Running (kjører i CPU), Ready (ligger i prosesskøen og venter på å få kjøre) eller Blocked (venter på at en hendelse skal skje, oftest at I/O skal bli ferdig), Hvilken av følgende overganger fra én tilstand til en annen er ikke mulig å få til?

- Blocked → Ready
- Ready → Blocked
- Running → Ready
- Ready → Running

Deloppgave B

Hvilket systemkall i Linux kan brukes i C-programmer til å lage en ny prosess som blir en kopi av den kallende prosessen, men som ikke vil dele minneområde med forelderprosessen?

- fork
- new
- clone
- malloc

Deloppgave C

Hvilket systemkall i Linux kan brukes i C-programmer til å lage en ny kjernetråd (også kalt "lettvektsprosess" i Linux) som vil dele minneområde med den kallende prosessen?

- fork
- new
- clone
- malloc

Deloppgave D

En webserver kjøres som én prosess med flere aktive tråder. En av trådene, "master"-tråden, har ansvaret for å ta i mot requests. En request er en forespørsel/henvendelse som kommer fra en webklient på nettet, om å få se innholdet i en webside. Hver gang det kommer en request, starter master-tråden en ny "worker"-tråd. Worker-trådene skal lese innholdet av en webside fra disk, og deretter returnere dette innholdet til webklienten som har bedt om å få hente websiden. For å lese fra disk brukes en blokkerende read-operasjon.

Hvilken implementasjon av tråder vil være minst effektiv og bør ikke brukes i en slik webserver?

- Kjernetråder
- Brukertråder, der alle trådene kjøres av et run-time system i user mode
- POSIX-tråder
- "Lettvektsprosesser" i Linux

Deloppgave E

En datamaskin har én CPU som bare har én kjerne. Alle prosessene som kjører på maskinen bruker 50% av tiden (fra prosessen opprettes til den termineres) til å vente på I/O. I/O skjer på helt tilfeldige tidspunkter. Hvor mange prosesser må minst være aktive på maskinen samtidig for at CPU-utnyttelsen skal bli bedre enn 95%?

← Skriv svaret ditt (et heltall) her

Maks poeng: 7.5

4 Oppgave 4: Interprosesskommunikasjon

Deloppgave A

Låsvariable ("locks") brukes for å unngå race conditions. Hvis en tråd A forsøker å endre/få tilgang til en låsvariabel som holdes av en annen tråd B, må tråd A vente til B frigir låsvariabelen. Hvilket av utsagnene nedenfor er riktig for en låsvariabel som er implementert som en "spin lock"?

- Trådene kan bare gjøre atomære/udelelige operasjoner på en "spin lock".
- Med en "spin lock" vil trådene A og B alltid kjøre sin kritiske kode annenhver gang.
- Trådene blokkeres ikke, men kjører i en løkke inntil "spin lock" frigis.
- En "spin lock" kan alltid implementeres mer effektivt enn semaforer og mutexer.

Deloppgave B

Hva var navnet på informatikk-forskeren som introduserte bruken av semaforer i operativsystemer for å unngå race conditions?

- Edsger W. Dijkstra
- Alan Turing
- Kristen Nygaard
- Ole Johan Dahl

Deloppgave C

Følgende metode brukes for å prøve å unngå race conditions mellom kjørende tråder:

- Alle trådene deler én låsvariabel *L* (en vanlig heltallsvariabel) som initielt settes lik 0
- Når en tråd vil begynne å eksekvere kode i sitt eget kritiske område:
  - Hvis *L* er lik 1, vent her til *L* blir lik 0
  - Sett *L* lik 1 og fortsett inn i kritisk område
  - Sett *L* tilbake til 0 når tråden er ferdig med kritisk område

Hvorfor vil ikke denne metoden fungere?

- Operasjonene på *L* er ikke atomære/udelelige.
- Vi får en race condition på selve låsvariabelen.
- Metoden kan ikke garantere at bare én tråd om gangen eksekverer kode i kritisk område.
- Alle alternativene ovenfor er årsaker til at metoden ikke fungerer.

Deloppgave D

Hvilket av utsagnene nedenfor om race conditions er ikke sant?

- Kan gjøre programvare uforutsigbar og gi feil resultater
- Kan gjøre debugging svært vanskelig
- Oppstår oftest fordi OS bruker timesharing, med en scheduler som stopper og starter prosesser for at de skal kunne dele på CPU(ene)
- Kan forebygges ved å øke kapasiteten på RAM

Deloppgave E

Følgende C-program som bruker biblioteket pthreads for trådhåndtering er gitt:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

pthread_cond_t cond;
pthread_mutex_t lock;

int x = 0;

void *start(void *dummy)
{
   pthread_mutex_lock(&lock);
   if (x == 0)
   {
      x = 1;
      printf("red ");
      pthread_cond_wait(&cond, &lock);
      printf("blue ");
   }
   else
   {
      printf("green ");
      pthread_cond_signal(&cond);
   }
   pthread_mutex_unlock(&lock);
}

int main()
{
   pthread_t t1, t2;
   pthread_mutex_init(&lock, NULL);
   pthread_cond_init(&cond, NULL);
   pthread_create(&t1, NULL, start, NULL);
   sleep(1);
   pthread_create(&t2, NULL, start, NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   printf("\n");
}
```

Hva skrives ut når C-programmet ovenfor kjøres?

- red green blue
- red blue green
- green blue red
- De tre ordene "red", "blue" og "green" i en eller annen tilfeldig rekkefølge

---

Maks poeng: 7.5

5 Oppgave 5: Scheduling

Deloppgave A
For å få et effektivt system med god utnyttelse av både CPU og I/O-enheter, bør en scheduler prioritere:

- CPU-bundete prosesser
- Prosesser som krever mye RAM
- I/O-bundete prosesser
- Alle de tre typene prosesser angitt i alternativene ovenfor

Deloppgave B
Turnaround time i scheduling av prosesser er definert som:

- Den totale tiden fra første gang prosessen ble lagt inn i readykøen til den er ferdig og terminerer
- Den totale tiden som prosessen har ligget i readykøen
- Den totale tiden som bruker har ventet på respons fra prosessen
- Den totale tiden som prosessen har ventet på I/O

Deloppgave C
Ventetid (wait time) i scheduling av prosesser er definert som:

- Den totale tiden fra første gang prosessen ble lagt inn i readykøen til den er ferdig og terminerer
- Den totale tiden som prosessen har ligget i readykøen
- Den totale tiden som bruker har ventet på respons fra prosessen
- Den totale tiden som prosessen har ventet på I/O

Deloppgave D
Hvilken av følgende hendelser vil kunne medføre at operativsystemet gjør scheduling?

- Kjørende prosess oppretter selv en ny prosess
- Kjørende prosess terminerer
- Kjørende prosess blokkerer, f.eks. for I/O
- Kjørende prosess stoppes av et klokke-avbrudd
- Alle de fire hendelsene ovenfor vil kunne forårsake scheduling.

Deloppgave E

I et batch-system (som bare har én prosessor med én kjerne) er det 5 prosesser, A-E, som venter på å få kjøre. Kjøretiden i minutter som hver prosess krever er: A:3 B:2 C:5 D:8 E:6. I hvilken rekkefølge må prosessene kjøres for å få lavest mulig gjennomsnittlig turnaround time?

- B C E D A
- A B C D E
- E D C B A
- B A C E D

Maks poeng: 7.5

6 Oppgave 6: Minnehåndtering

Deloppgave A
I et system med virtuelt minne og paging, hva slags minneadresser brukes internt i CPU?

- Virtuelle adresser
- Fysiske adresser
- Absolutte adresser
- Base- og limit-registre

Deloppgave B
Operativsystemet må lagre og vedlikeholde én pagetabell for hver:

- CPU
- Tråd
- Prosess
- CPU-kjerne

Deloppgave C
Hvilke(t) teknisk(e) problem(er) som er forbundet med sammenhengende fysiske minneområder løses når vi bruker virtuelt minne med paging?

- Svært ressurskrevende prosesser (bloatware) trenger ikke å ha hele minneområdet sitt i RAM når vi bruker virtuell hukommelse med paging.
- Vi slipper å swappe store sammenhengende minneområder til/fra disk.
- Vi unngår å få fragmentert/"hullete" minne.
- Alle de tre problemene ovenfor løses når vi bruker virtuell hukommelse og paging.

Deloppgave D
En prosess bruker 4 pages med minne. Algoritmen "NRU – Not Recently Used" brukes for velge ut hvilken page som skal lagres på disk ved en page fault. Verdiene av R-bit ("Referenced") og M-bit ("Modified") for hver page er:

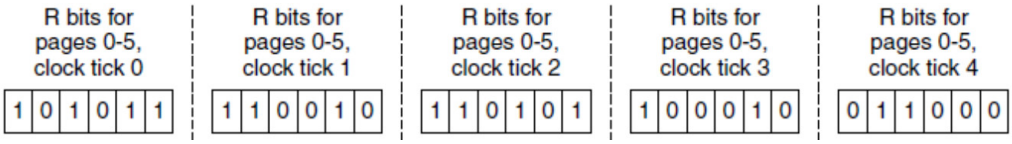
Page	R-bit	M-bit
0	0	1
1	1	1
2	0	0
3	1	0

Hvilken page vil bli valgt av NRU-algoritmen?

- Page 0
- Page 1
- Page 2
- Page 3

Deloppgave E

Aging-algoritmen skal brukes for å gjøre page replacement. Det brukes en 5-bits "teller" i algoritmen, dvs. at verdien av R-bit lagres for de 5 siste klokketikkene for hver page. Pagetabellen har 6 pages, nummerert fra 0 til 5. Det skal nå gjøres en page replacement, og de lagrede verdiene for R-bit ser slik ut:



Hvilken page vil bli valgt?

- Page 0
- Page 1
- Page 2
- Page 3
- Page 4
- Page 5

Maks poeng: 7.5

7 Oppgave 7: Deadlocks

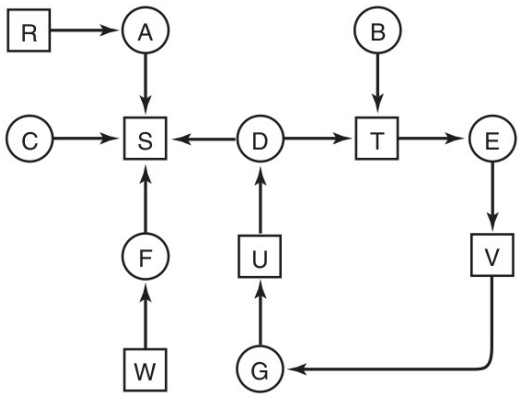
Deloppgave A
Hvilken av disse fire algoritmene/strategiene for deadlockhåndtering er raskest og krever minst systemressurser?

- Deadlock prevention
- Ostrich algorithm
- Deadlock avoidance
- Deadlock detection and recovery

Deloppgave B
Hva er årsaken(e) til at standardversjonen av både Windows og Linux bruker "strutsealgoritmen" for deadlocks?

- Deadlocks forekommer svært sjeldent.
- Det er vanskelig å få automatisk deadlockhåndtering til å fungere godt.
- Deadlockhåndtering kan være ressurskrevende og gjøre systemet tregere.
- Alle de tre alternativene ovenfor er årsaker til at "strutsealgoritmen" brukes.

Nedenfor er det vist en ressursallokeringsgraf for et system med 7 prosesser A - G (tegnet som sirkler) og 6 ulike ressurser R - W (tegnet som firkanter). Det er bare én forekomst av hver enkelt ressurs. Denne grafen skal brukes i deloppgavene C og D.



Deloppgave C
Hvilket av følgende utsagn om prosesser og ressurser i grafen ovenfor er ikke sant?

- Prosess A holder R og venter på S
- Det finnes en ressurs i grafen som fire av prosessene ønsker å reservere
- Alle prosessene holder minst én ressurs
- Alle prosessene venter på minst én ressurs

Deloppgave D

Det er en deadlock i systemet som er modellert med ressursgrafen gitt ovenfor. Hvilke prosesser er i deadlock?

- A, C, D og F
- A og B
- D, E og G
- Alle prosessene er i deadlock

Deloppgave E

Et system har 4 prosesser P1 - P4, og 5 ulike typer ressurser A-E som har et ulikt antall forekomster. Antallet forekomster av hver ressurs som prosessene holder, og antallet forekomster som de trenger, er gitt i de to tabellene til venstre nedenfor. Det er også angitt hvor mange forekomster av hver ressurs som fortsatt er tilgjengelig i systemet, og hvor mange forekomster som totalt finnes.

Holder	*A*	*B*	*C*	*D*	*E*
P1	0	1	1	1	2
P2	0	1	0	1	0
P3	0	0	0	0	1
P4	2	1	0	0	0

Trenger	*A*	*B*	*C*	*D*	*E*
P1	1	1	0	2	1
P2	0	1	0	2	1
P3	0	2	0	3	1
P4	0	2	1	1	0

Finnes	*A*	*B*	*C*	*D*	*E*
	2	4	1	4	4

Tilgjengelig	*A*	*B*	*C*	*D*	*E*
	0	1	0	2	1

Du skal bruke Banker's-algoritmen på dette systemet for å finne ut om det er i en sikker eller usikker tilstand. Når du gjør dette, vil du se at det er en eller flere prosesser som ikke kan få kjøre og derfor er i deadlock.

Hvilke av prosessene P1-P5 er i deadlock?

- P3
- P1 og P4
- P2 og P5
- P2, P3 og P5

Maks poeng: 7.5

8 Oppgave 8: Filsystemer

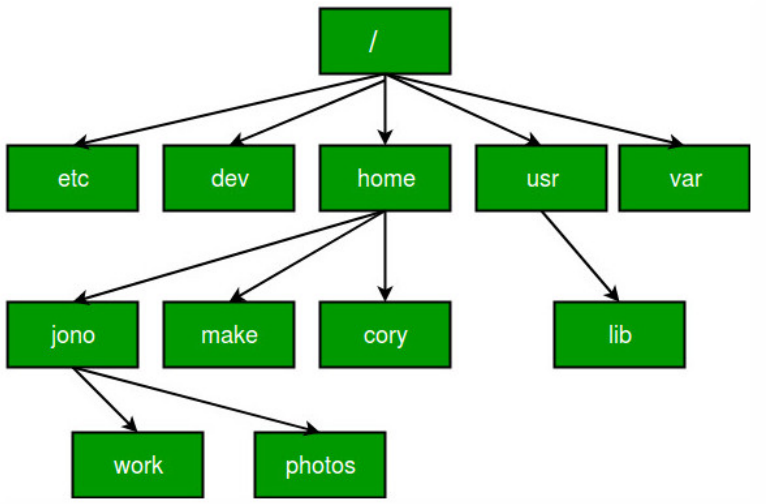
Deloppgave A
Hva er oppgaven(e) som et filsystem skal utføre i OS?

- Håndtere filenes innhold og metadata
- Strukturere lagringsplassen som er tilgjengelig
- Gi god pålitelighet, effektivitet og datasikkerhet for eksterne lagringsmedia
- Tilby brukerne verktøy for å håndtere filer
- Muliggjøre deling av data
- Alle oppgavene ovenfor skal utføres av et filsystem

Deloppgave B
Nedenfor er det gitt beskrivelser av hva en fil kan være. Hvilke(n) av beskrivelsene er riktige for filer i Linux?

- En fil er en samling med data/bytes som logisk hører sammen
- En fil er enten en kilde som data kan leses fra, eller et medium som data kan skrives til
- En fil representerer en fysisk enhet som kan sende eller ta i mot data
- Alle definisjonene ovenfor beskriver filer i Linux

Deloppgave C
Figuren nedenfor viser et katalogtre i Linux, der alle de grønne "boksene" innholder navnet på én katalog.



Anta at stående katalog er `jono`. Hvilke(n) av shell-kommandoene nedenfor vil skrive ut en liste med navnet på filene i katalogen `lib`?

- `ls ../../usr/lib`
- `ls usr/lib`
- `ls home/usr/lib`
- `ls ../home/usr/lib`
- Alle kommandoene ovenfor vil liste ut filene i `lib`

Deloppgave D

En tekstfil som har filnavnet `minfil.txt` inneholder bare denne ene linjen med tekst:

```
Linux Windows for winners loosers
```

Følgende C-program bruker Linux systemkall til å åpne og lese fra denne filen:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define N 10

int main()
{
   int i, fd;
   char S1[N], S2[N], S3[N];

   for (i = 0; i < N; i++)
      S1[i] = S2[i] = S3[i] = '\0';

   fd = open("minfil.txt", O_RDONLY);

   read(fd, S1, 6);
   lseek(fd, 8, SEEK_CUR);
   read(fd, S2, 4);
   lseek(fd, -16, SEEK_END);
   read(fd, S3, 7);

   printf("%s%s%s\n", S1, S2, S3);
}
```

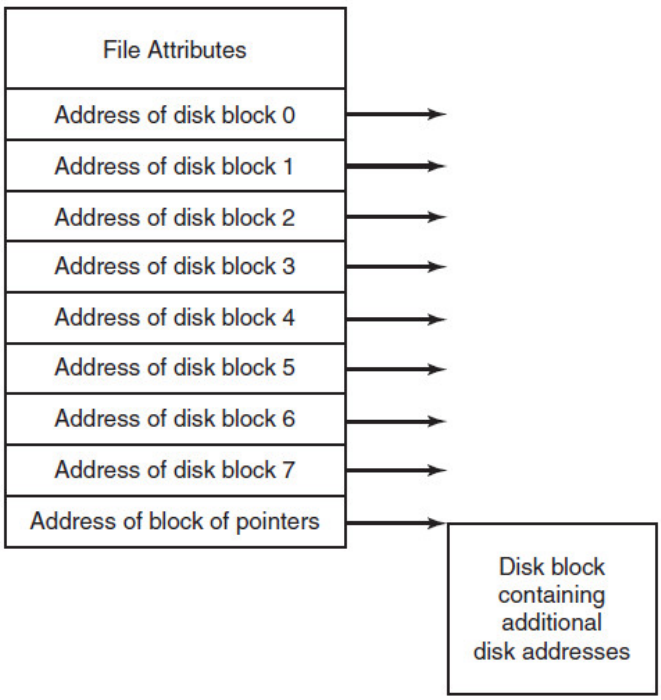
Hva skrives ut når C-programmet ovenfor kjøres?

- Windows for winners
- Windows for loosers
- Linux win Windows loose
- Linux for winners

Deloppgave E

Figuren nedenfor viser en i-node som, i tillegg til metada/filattributter, kan lagre

- 8 direkte adresser/pekere til diskblokker
- 1 peker til en indirekte blokk, som inneholder et antall direkte pekere til diskblokker



Anta at et filsystem som bruker slike i-noder har en blokkstørrelse på 4 KB = 4096 bytes, og at hver peker (diskadresse) er på 4 byte. Hva er maksimal filstørrelse i dette filsystemet?

- 1032 KB
- 3096 KB
- 4128 KB
- 8256 KB

Maks poeng: 7.5

9 Oppgave 9: Tallsvaroppgave – Regning med schedulingalgoritmer

Deloppgave A

De seks prosessene A,B,C, D, E og F skal kjøres på et batchsystem der det brukes First-Come-First-Served scheduling. Systemet har bare én prosessor med én kjerne. Ankomsttid og total kjøretid (CPU-tid) i millisekunder for hver prosses er gitt i tabellen nedenfor. Du kan anta at et context switch tar 0 (null) millisekunder. Hva er gjennomsnittlig turnaround time i millisekunder for de seks prosessene?

Prosess	Ankomst	Kjøretid
A	0	4
B	7	2
C	9	10
D	13	8
E	25	1
F	35	3

← Skriv svaret ditt (et heltall) her

Deloppgave B

De samme seks prosessene som i deloppgave A ovenfor, med samme ankomsttider og kjøretider, skal nå kjøres på et interaktivt system der det brukes vanlig Round-Robin scheduling. Systemet har bare én prosessor med én kjerne. Quantum (time slice) er lik 4 millisekunder og et context switch tar 0 (null) millisekunder.

Hvor mange millisekunder blir den totale ventetiden i readykøen for prosess C?

← Skriv svaret ditt (et heltall) her

Maks poeng: 10

10 Oppgave 10: Tekstoppgave – Deadlocks

Deloppgave A

Gi en kort forklaring av hva det betyr at en mengde med prosesser er i en "resource deadlock":

Deloppgave B

Hva er de fire nødvendige betingelsene som alle må være til stede for at deadlock skal kunne forekomme? Forklar hver betingelse med 1-2 setninger:

Deloppgave C

Anta at det bare er én forekomst av hver enkelt ressurs i systemet. Er de fire betingelsene fra deloppgave B da også tilstrekkelige, slik at vi alltid får en deadlock når alle fire er til stede i systemet? Gi en kort begrunnelse for svaret ditt:

Deloppgave D

Anta at det kan være flere forekomster av hver enkelt ressurs i systemet. Er de fire betingelsene fra deloppgave B da også tilstrekkelige, slik at vi alltid får en deadlock når alle fire er til stede i systemet? Gi en kort begrunnelse for svaret ditt:

Maks poeng: 10

11 Oppgave 11: Programmering – Tråder

C-programmet nedenfor starter 10 tråder som alle "teller opp" samme delte variabel (heltallet *number*). Hver tråd skriver ut verdien av *number* etter å ha telt opp "sin del" (satt til 10000 for alle trådene). Programmet forsøker å unngå race conditions ved å la trådene låse på en global spin-lock (variabelen *lock)*.

```
#include <pthread.h>
#include <stdio.h>
#define N_THREADS 10
int number = 0;  // Global variable
int lock = 0;    // Global spin-lock

void *count_and_print(void *v)
{
   int i;
   // Wait until spin-lock is set to 0 (zero)
   while (lock != 0);
   // "Lock" by setting spin-lock to 1 (one)
   lock = 1;
   // Do the counting and print out result
   for (i = 0; i < 10000; i++)
      number++;
   printf("%d ", number);
   // Unlock and exit
   lock = 0;
   pthread_exit(NULL);
}

int main(int argc, char * argv[])
{
   pthread_t threads[N_THREADS];
   int i;
   for(i = 0; i < N_THREADS; i++)
      pthread_create(&threads[i], NULL, count_and_print, NULL);
   for (i = 0; i < N_THREADS; i++)
      pthread_join(threads[i], NULL);
   printf("\n");
   exit(NULL);
}
```

Ved en kjøring av programmet ble utskriften denne:

```
10000 19526 21850 31452 33744 43744 53744 63744 73744 83744B
```

En annen kjøring av programmet ga dette resultatet:

```
9761 18858 22480 31528 31605 40022 42860 43291 50455 55117
```

Skriv om programmet slik at det alltid vil gi denne utskriften:

```
10000 20000 30000 40000 50000 60000 70000 80000 90000 100000
```

Skriv besvarelsen din i kodefeltet nedenfor. Du kan gjerne "lime inn" C-koden gitt ovenfor og bare skrive om denne litt.

```
1
```

Maks poeng: 10

12 Oppgave 12: Programmering – Interprosesskommunikasjon

Skriv et C-program som bruker tre tråder, *Thread1*, *Thread2* og *main*:

- *Thread1* skal skrive ut "Introduction to OS course" fem ganger
- *Thread2* skal skrive ut "Welcome to the " fem ganger
- *main* skal vente inntil *Thread1* og *Thread2* begge er ferdige, og deretter skrive ut "Østfold University College!"

Bruk betingede variable (conditional variables) og mutex i *pthreads* til å synkronisere de tre trådene, slik at utskriften fra programmet alltid ser slik ut:

```
Welcome to the Introduction to OS course
Welcome to the Introduction to OS course
Welcome to the Introduction to OS course
Welcome to the Introduction to OS course
Welcome to the Introduction to OS course
Østfold University College!
```

Deler av C-programmet du skal skrive er gitt nedenfor (her antar vi at alle POSIX/pthreads variable er initialisert riktig i *main*). Du skal skrive koden som mangler i *Thread1*, *Thread2* og *main*, slik at programmet fungerer som beskrevet ovenfor.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

// Your global variables here
...

void* Thread1()
{
    // Your code for Thread1
    ...
}

void* Thread2()
{
    //Your code for Thread2
    ...
}

int main(int argc, char** argv)
{
    //Your code for initializing local or global variables
    ...
    pthread_t t1;
    pthread_t t2;
    pthread_create(&t1, NULL, (void*)Thread2, NULL);
    pthread_create(&t2, NULL, (void*)Thread1, NULL);
    //Your code for the rest of the program
    ...
    return 0;
}
```

Skriv C-programmet ditt her:

```
1
```

Maks poeng: 10

Egne kommentarer

Her kan du legge inn evt. kommentarer til oppgavene og til din egen besvarelse.