OPERATIVSYSTEM 2011, ÖVNING 4, 1.12.2011, deadline 9.12.2011

1. En serieport (används ibland ännu ;) tar emot en byte åt gången över en serielinje. Vid designen av drivrutinen övervägdes det att antingen använda aktivt väntande eller avbrottsbaserad läsning av inkommande data.
a) Om betjäning av avbrott (dvs. utföra avbrottsrutinen samt all extra tid som går åt till att aktivera avbrottsfunktionaliteten) tar 500 ns, vid vilken bit-rate (bit/sekund) lönar det sig att gå över från avbrottsbaserad till aktivt väntande? Vi kan negligera tiderna det tar att föra över data från serieporten till minnet.
b) Generellt använder sig hårdvaran ofta av en buffert med 16 bytes, för att kunna mellanlagra på serieporten inkommande data. Vilken effekt har det för valet mellan aktivt väntande och avbrottsbaserad läsning?
(2p)

2. SSD (Solid State Disk) börjar bli allt vanligare i bärbara datorer. SSD finns i två varianter, SLC och MLC. Sök på nätet skillnaden mellan SSD gjort med SLC och MLC teknik. Är någondera att föredra (att varför), eller lönar det sig med vanlig magnetisk hårdskiva, för följande: (2p)

a) Som systemskiva för operativsystem
b) Lagring för multimediamaterial (Video / Audio)
c) Lagring av databas

Uppgör tabell, med följande layout, och fyll i fördelar / nackdelar

		Systemskiva	Multimedia	Databas
SSD SLC	Fördel			
	Nackdel			
SSD MLC	Fördel			
	Nackdel			
Magnetisk skiva	Fördel			
	Nackdel			

3. Förfrågningar inkommer till hårdskivedrivrutinen för cylindrar 20, 2, 52, 9, 17, 42 och 28, i given ordning. En sökning tar 8 ms per cylinder man flyttar sig över. Hur lång total söktid behövs för
    1. First come, first served
    2. Närmaste cylindern till näst
    3. Hissalgoritmen (till att börja med på väg uppåt)

   Vi antar i samtliga fall att diskarmen till att börja med finns på cylinder 20. (2p)

4. Datastrukturer för filsystemet ext2 är i linux definierade i filen */usr/include/linux/ext2_fs.h*. Nedan är några rader ur filen:

.....

```
#define EXT2_NDIR_BLOCKS               12
#define EXT2_IND_BLOCK                 EXT2_NDIR_BLOCKS
#define EXT2_DIND_BLOCK                (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK                (EXT2_DIND_BLOCK + 1)
```

```
#define EXT2_N_BLOCKS                   (EXT2_TIND_BLOCK + 1)
...
__le32  i_block[EXT2_N_BLOCKS];/* Pointers to blocks */'
...
```

där konstanten *EXT2_NDIR_BLOCKS* anger hur många block man direkt addresserar från i-noden. Konstanterna *EXT2_IND_BLOCK*, *EXT_DIND_BLOCK* resp. *EXT2_TIND_BLOCK* anger index för enkel, dubbel respektive trippel indirektion till block-adresser. Block-adresser är 32 bit. Om block-storleken är a) 1kB b) 4kB, hur stor är den största filstorleken i filsystemet?

5. Med kommandot *mkisofs* (finns på t.ex. lenz.cs.abo.fi) kan man bygga upp ett ISO9660-filsystem. Ett exempel på ett sådant filsystem finns på

http://www.abo.fi/~jbjorkqv/opsys_09/test.iso

Ett ISO9660 filsystem består av sektorer på 2048 byte. De första 16 sektorerna (0-15) består av nollor (de är reserverade för framtida bruk). Sektor 17 innehåller *primary volume descriptor*, med format som i början på *struct t_isovoldesc* i koden nedan. Ladda ner *test.iso*, kompilera och testa koden. Lägg därefter till kod som dessutom skriver ut sys_ident samt totala antalet sektorer samt storleken på hela filsystem i bytes. Beskriv koden och bifoga resultat. (2p)

```
#include <stdio.h>

#define FILENAME "test.iso"

char volident[] = {67,68,48,48,49,1};

struct t_isovoldesc {
  char start;
  char desc[7];
  char sys_ident[32];
  char vol_ident[32];
  char zeros[8];
  long long sectors;
};

struct t_iso_sector {
  char data[2048];
};

int main() {
  FILE *fd;

  struct t_iso_sector iso_sector;
  struct t_isovoldesc *isovoldesc;

  fd = fopen(FILENAME, "r");
  while (!feof(fd)) {
    fread(&iso_sector, sizeof(iso_sector), 1, fd);
```

```
    isovoldesc = (struct t_isovoldesc *)&iso_sector;
    /* Search for sector containing volume descriptor */
    if (strcmp(isovoldesc->desc, volident)==0) break;
  }
  printf("Volume name: %s\n", isovoldesc->vol_ident);
  return 0;
}
```