OPERATIVSYSTEM 2009, RÄKNEÖVNING 1, v. 38 (14-18.9.2008)

OBS. Räkneövning inlämnas elektronisk på adressen https://xprog28.cs.abo.fi/ro.nsf.
Deadline: Måndag 21.9.2009 kl. 16.00.

1. Med kommandot `strace` (finns i de många linux-distributioner) kan man lista alla de anrop som görs till kärnan. Kompilera och exekvera HelloWorld-exemplet (nedan). Räkna upp alla systemanrop som görs, samt frekvensen av dessa (detta kan göras på datorn *lenz.cs.abo.fi*, som också fungerar som kursens experimentdator [obs] login behövs, fås via RÖ-assistent). Kör sedan motsvarande för ett större programpaket (t.ex. firefox), och lista de 10 mest använda systemanropen (använd parametern *–c* i *strace*)(2p)

```
// File helloworld.c
#include <stdio.h>

int main() {
  printf("Hello World\n");
}
// ------------------

Kompilering och körning
% gcc helloworld.c –o helloworld
% ./helloworld
% strace ./helloworld
```

2. En del funktionsanrop i systembiblioteket resulterar i anrop till operativsystemets kärna (kernel), andra inte. Varför går inte alla systembibliotekanrop till kärnan? Vilkendera är snabbare? (2p)

   (Ledning: jämför funktionsanropen *read()* och *fread()*, sök med t.ex. read vs. fread)

3. Ett soft-realtidssystem har fyra periodiska händelser med perioder om 50, 100, *y* och 250 ms, där *y* är 10 * dag i månaden för ditt födelsedatum (t.ex *y*=170 för födelsedag 17.7). Anta att de fyra händelserna kräver 35, 20, 10 och *x* ms processortid respektive. Vilket är det största värde på *x* för vilket systemet är skedulerbart? Beräkna och motivera. (2p)

4. På nästa sida ges ett program som delar sig i två processer. Faderprocessen läser från tangentbordet och matar till ett rör (pipe). Dotterprocessen läser från röret och skriver ut på skärmen. Modifiera programmet så att en tredje process behandlar strängen, så att den gör en palindrom av strängen (dvs. inverterar bokstavsordningen). Kommunikationen sköts fortsättningsvis med rör. (4p)

   Källkoden hittar du också på http://www.abo.fi/~jbjorkqv/opsys/pipes.c

```
/* pipes.c
 * compile using: gcc -o pipes pipes.c
 * run using ./pipes
 */

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

#define BUFFERSIZE 100

int main() {

  int pid;
  char buffer[BUFFERSIZE]; /* Allocate buffer for  temporary data */
  int files[2];  /* File descriptor in unix is a integer, used
                    together with operation read, write */

  /* Create the pipe, files[0] and files[1] will contain
     the file descriptors of both ends, note, writing will happen
     to files[1], reads from files[0] */
  pipe(files);

  /* Create the child process */
  if ((pid = fork())) {
    /* OK, I'm the parent */
    while (1) {
      printf("Parent; Enter string to put into the pipe: ");
      /* Read from the standard input (=keyboard...) */
      gets(buffer);

      /* Write the complete buffer to the pipe */
      write(files[1], buffer, BUFFERSIZE);
      sleep(1);
    }
  } else {
    /* Ok, I'm the child */
    while(1) {
      /* Read from the pipe, read the complete buffer */
      read(files[0], buffer, BUFFERSIZE);
      /* Output what was read ... */
      printf("Child: From pipe; '%s'\n", buffer);
    }
  }
}
```