# Operating Systems practicum 2

Roan Kattouw and Jan Paul Posma

8 juni 2011

## iwish

## UTCtime

Allereerst hebben we de system call gemaakt en daarna de library call.

### System call

De system call hebben we bij de andere time functies gezet in `/usr/src/servers/pm/time.c`. Aangezien het naïef is om te denken dat we simpelweg alle oude leap-seconden van het huidige aantal kunnen aftrekken, kijken we daadwerkelijk naar de exacte tijd. Het is namelijk ook mogelijk dat er bijvoorbeeld een oude gebeurtenis wordt nagebootst in een Virtual Machine, met de klok dus teruggezet. In veruit de meeste gevallen zal echter simpelweg het getal 24 (aantal leap-seconden tot nu toe) moeten worden afgetrokken. Daarom controleren we eerst of we dit kunnen doen; dat is sneller.

```
#define LEAP(x) if(rt>=x) { rt--; }

PUBLIC int do_utctime()
{
  clock_t uptime, boottime;
  time_t rt;
  int s;

  if ( (s=getuptime2(&uptime, &boottime)) != OK)
panic(__FILE__, "do_utctime couldn't get uptime", s);
```

```
  rt = (time_t) (boottime + (uptime/system_hz));
  mp->mp_reply.reply_utime = (uptime%system_hz)*1000000/system_hz;

  /* Correct for leap seconds.
   */
  if ( rt >= 1230768023 )
  {
  /* This will be used in practically all cases.
   */
    rt -= 24;
  }
  else
  {
  /* If for some reason the clock is behind, do the correction right.
   */
    LEAP(78796800); LEAP(94694400); LEAP(126230400); LEAP(157766400); LEAP(1893024
    LEAP(220924800); LEAP(252460800); LEAP(283996800); LEAP(315532800); LEAP(36279
    LEAP(394329600); LEAP(425865600); LEAP(489024000); LEAP(567993600); LEAP(63115
    LEAP(662688000); LEAP(709948800); LEAP(741484800); LEAP(773020800); LEAP(82045
    LEAP(867715200); LEAP(915148800); LEAP(1136073600); LEAP(1230768000);
  }

  mp->mp_reply.reply_time = rt;

  return(OK);
}
```

Verder moesten nog enkele bestanden worden aangepast. In `/usr/src/servers/pm/proto.h` hebben we het functieprototype toegevoegd:

```
_PROTOTYPE( int do_utctime, (void) );
```

Verder hebben we voor de unused positie 69 gekozen voor onze system call.
In `/usr/src/include/minix/callnr.h` zorgt dit voor:

```
#define UTCTIME 69
```

Daarnaast hebben we in `/usr/src/servers/pm/table.c` op positie 69 `no_sys` vervangen door `do_utctime`.

### Library call

Het maken van een library call is relatief eenvoudig. We zijn wel in twee valkuilen gelopen. Ten eerste moet er ook een `utctime.s` bestand worden aangemaakt, wat niet overal vermeld wordt. Daarnaast werkt het ook niet wanneer het testprogramma gecompileerd wordt met `gcc` in plaats van `cc`.

Allereerst hebben we in `/usr/src/lib/posix/_utctime.c` een wrapper gemaakt, afgekeken van de `time` system call:

```
#include <lib.h>
#define time _time
#include <time.h>

PUBLIC time_t utctime(tp)
time_t *tp;
{
  message m;
  if (_syscall(MM, UTCTIME, &m) < 0) return( (time_t) -1);
  if (tp != (time_t *) 0) *tp = m.m2_l1;
  return(m.m2_l1);
}
```

Hierbij moest ook `_utctime.c` in `/usr/src/lib/posix/Makefile.in` worden toegevoegd. De functiedefinitie hebben we in `/usr/src/include/time.h` geplaatst:

```
_PROTOTYPE( time_t utctime, (time_t *_timeptr) );
```

Tenslotte moest de eerdergenoemde `utctime.s` worden aangemaakt, namelijk op `/usr/src/lib/syscall/utctime.s`:

```
.sect .text
.extern __utctime
.define _utctime
.align 2

_utctime:
jmp __utctime
```

Natuurlijk moest ook `utctime.s` worden toegevoegd aan `/usr/src/lib/syscall/Makefile.in`.

Ten slotte een eenvoudig test-programma, `test.c`:

```
#include<time.h>
#include<stdio.h>

int main()
{
  printf("time    = %u\n", time(0));
  printf("utctime = %u\n", utctime(0));
  return 0;
}
```

Dit laat inderdaad de gewenste uitvoer zien:

```
time    = 1307460054
utctime = 1307460030
```