Operating Systems practicum 2

Roan Kattouw and Jan Paul Posma

8 juni 2011

iwish

iwish heet in onze inzending simpelweg `shell`.

Extra features

Onze shell heeft de volgende features die niet in de opdracht vermeld staan:

- Een prompt die aangeeft of de gebruiker `root` is (`#`) of niet (`$`)
- De speciale commando's `cd` en `exit` worden ondersteund. Ctrl+D bij een lege invoer beëindigt de shell ook
- I/O redirection ondersteunt append mode (`command >> file`)
- I/O redirection van en naar willekeurige file descriptors (`3< file`, `2> file`, `2>> file` en `2> &1`)
- Verkorte syntax voor I/O redirection (`command<infile>outfile`), met correct onderscheid tussen `foo2>bar` en `foo 2>bar`
- Gebruik van backslashes om karakters met een speciale betekenis te escapen, bijv. in bestandsnamen (`cat foo\ \&\ bar`)
- Gebruik van single of double quotes voor hetzelfde doel (`cat 'foo & bar'`)
- Bash-compatible gedrag voor `foo | bar & baz` (`foo` en `bar` in de achtergrond, `baz` in de voorgrond)

Commando parsing

De input van de gebruiker wordt eerst geparset en omgezet naar een datastructuur met linked lists die de verschillende commando's, hun argumenten en I/O redirections representeren. Dit gebeurt in `parseCommandLine()` in `command.c`. We hebben ervoor gekozen om linked lists te gebruiken omdat dit gemakkelijker was dan te werken met arrays van van te voren ombekende grootte.

```
#define READ 1
#define WRITE 2
#define APPEND 4
#define PIPE 8

#define BACKGROUND 1
#define PIPED 2

struct redirection
{
        /* FD being redirected */
        int fromfd;
        /* Filename to redirect to, or NULL if redirecting to an FD */
        char *filename;
        /* FD to redirect to. Only used if filename is NULL */
        int tofd;
        /* Bitmap of READ, WRITE, APPEND, PIPE */
        int mode;
        struct redirection *next;
};

struct argument
{
        char *s;
        struct argument *next;
};

struct command
{
        /* Pointer to the head of the argument list */
        struct argument *firstArg;
```

```
        /* Pointer to the head of the redirection list or NULL */
        struct redirection *redir;
        /* Bitmap of BACKGROUND, PIPED */
        int mode;
        /* PID once started */
        pid_t pid;
        struct command *next;
};
```

Elk commando bestaat uit een lijst argumenten, gerepresenteerd door een linked list van `struct arguments`. Het eerste argument is de naam van het programma. Commando's worden gescheiden door `|` of `&`. Er kunnen dus meerdere commando's op één command line voorkomen. De code voor het parsen van command lines en het bouwen van deze datastructuren bevindt zich in `command.c`.

Commando's uitvoeren

Als een command line eenmaal is omgezet in een linked list van commando datastructuren, kunnen we deze uitvoeren. De code hiervoor bevindt zich in `execute.c`, met als hoofdfunctie `executeCommand()`. De volgende stappen worden uitgevoerd:

- Redirections naar bestanden worden omgezet naar redirections naar file descriptors door het bestand te openen en de verkregen file descriptor in het `tofd` veld te zetten
- Pipes worden toegevoegd als redirections van `stdin` of `stdout` naar de betreffende file descriptor
- Voor elk commando wordt een nieuw proces gemaakt. Het PID hiervan wordt opgeslagen in het `pid` veld van `struct command`
- Elk kindproces
  - Zet zijn I/O redirections op. Pipes vallen hier ook onder dankzij de preprocessing hierboven
  - Sluit de files en pipes die waren geopend voor andere kinderen
  - Zet de linked list van argumenten om in een array en roept `execvp()` aan. Deze functie zorgt automatisch voor `$PATH` re-

solution en het doorgeven van omgevingsvariabelen (de `env` parameter van `execve()`)

- De ouder sluit alle files en pipes die het geopend heeft voor I/O redirection
- De ouder roept `waitpid()` aan op elk kindproces dat niet de `BACKGROUND` flag heeft

Achtergrondprocessen opruimen met SIGCHLD

De shell wacht niet op kinderen die in de achtergrond draaien, maar gaat direct verder. Als die kinderen op een gegeven moment termineren, moeten ze toch worden opgeruimd door de shell, anders worden ze zombies. Een manier om dit te doen is om elke keer dat de prompt in beeld wordt gebracht na het uitvoeren van een commando een non-blocking wait (met de `WNOHANG` flag) te doen op PID -1 totdat dit `ECHILD` geeft. Wij hebben ervoor gekozen om het `SIGCHLD` signaal af te vangen en in de handler een wait te doen op het getermineerde kind. Het PID van het kind staat in de `siginfo_t` structure die als de tweede parameter aan de signal handler wordt meegegeven als de signal handler geïnstalleerd is met de `SA_SIGINFO` flag.

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
```