Practicum Operating Systems
Sessie 3: Memory management

Arnold Meijster

10 juni 2011

1 Opgave: programma analyse

Beschouw het onderstaande programma (je kunt het via Nestor downloaden). Leg uit wat de uitvoer van het programma is, en hoe deze tot stand komt. Leg in je beschrijving in ieder geval uit wat de functie is van de code op de regels 7, 11-14, 20, 25, 31-32, 35, en 40-44.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <errno.h>
 4 #include <sys/mman.h>
 5 #include <signal.h>
 6
 7 #define PAGESIZE 4096  /* type in shell: getconf PAGE_SIZE */
 8
 9 void *address;
10
11 void MemoryWrite(int a) {
12     printf ("Memory was protected.\n");
13     mprotect(address, 1024, PROT_WRITE | PROT_READ);
14 }
15
16 int main(void) {
17   char *arr;
18   int ok;
19
20   signal(SIGSEGV, MemoryWrite);
21
22   /* Allocate a buffer; it will have the default
23    * protection of PROT_READ|PROT_WRITE.
24    */
25   ok = posix_memalign(&address, PAGESIZE, 1024);
26   if (ok != 0) {
27     perror("Couldn't malloc 1024 page-aligned bytes");
28     exit(errno);
29   }
30
31   arr = (char*)address;
32   arr[666] = 42;        /* Write; ok */
33
34   /* Mark the buffer read-only. */
35   if (mprotect(address, 1024, PROT_READ)) {
```

```
36      perror("Couldn't mprotect");
37      exit(errno);
38    }
39
40    printf ("val = %d\n", arr[666]);    /* Read; ok */
41
42    arr[666] = 2*arr[666];          /* Write; generates SIGSEGV */
43
44    printf ("val = %d\n", arr[666]);
45
46    return EXIT_SUCCESS;
47  }
```

2 Opgave: Simulated shared memory

In deze opgave gaan we een mini DSM-systeempje bouwen. DSM staat voor Distributed Shared Memory.

Het idee is het volgende. Processen die d.m.v. `fork()` zijn gemaakt delen het textsegment niet en kunnen dus niet bij elkaars data. We kunnen dit echter wel simuleren m.b.v. `mprotect()`. We maken een systeem met 2 processen en één page (4kB) shared memory. Natuurlijk heeft een echt shared memory systeem veel meer shared memory, maar voor deze opgave is dit voldoende. De shared page implementeren we als volgt. Op ieder proces wordt er één page memory gealloceerd dat met behulp van `mprotect` wordt beschermd tegen read/write-operaties. Op ieder moment beschikt slechts één van de processen over de page met de actuele (werkelijke) inhoud. Laten we aannemen dat dit proces 1 is. Als proces 1 in de 'shared' page wil lezen/schrijven, dan kan dit zonder enige belemmering. Echter, als proces 2 in de 'shared' page wil schrijven, dan stuurt dit proces een signaal naar proces 1, waarna proces 1 de actuele page stuurt naar proces 2. Daarna zijn de rollen van de twee processen, wat betreft ownership van de actuele page, omgedraait. Proces 2 kan dus nu in de 'shared' page lezen/schrijven.

Implementeer het bovenstaande idee. Bedenk zelf hoe de processen elkaar signaleren. Er zijn twee signals beschikbaar voor 'eigen' gebruik, namelijk `SIGUSR1` en `SIGUSR2`.

Laat zien dat jouw implementatie werkt m.b.v. van een eenvoudig ping-pong programma. Introduceer hiervoor een variabele `turn` in de 'shared' page en maak gebruik van busy-waiting. Maak je niet druk over efficientie!

3 Opgave: Een optimalisatie

Eigenlijk is het niet nodig dat er slechts één actuele kopie van de 'shared' page is. Immers, als beide processen een kopie van de actuele page bezitten, en allebei voeren ze (voorlopig) slechts read-operaties uit, dan zouden ze dit zonder belemmering op een lokale kopie kunnen doen (zonder message passing). Voer deze replicatie-strategie in in jouw programma. Bedenk zelf welke administratie je moet doen om het systeem consistent te houden. Probeer nu opnieuw het ping-pong programma. Merk je een verbetering?