Vorlesung Betriebssysteme

Prof. Dr. Jan Dünnweber

Verteilte Systeme und Betriebssysteme
Fakultät für Informatik und Mathematik
Ostbayerische Technische Hochschule Regensburg

Wintersemester 2013/14

Readers & Writers als POSIX Prozesse

Quelltext für Lese- und Schreib-Processe unter UNIX

```
/* A simple readers/writers program using a one-word shared memory. */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/mman.h>
#define SIZE sizeof(int)   /* size of [int] integer */
#define run_length 10      /* number of iterations in test run */

int main(void)  {
   pid_t pid;               /* variable to record process id of child */
   caddr_t shared_memory;   /* shared memory base address */
   int i_parent, i_child;   /* index variables */
   int value;               /* value read by child */
   /* set up shared memory segment */
  shared_memory=mmap(0, SIZE, PROT_READ | PROT_WRITE,
                             MAP_ANONYMOUS | MAP_SHARED, -1, 0);
  if (shared_memory == (caddr_t) -1) {
    perror ("error in mmap while allocating shared memory\n");
    exit (1);
  }
```

Readers & Writers in UNIX ... continued

Quelltext (Fortsetzung)

```
if ((pid = fork()) < 0)  { /* apply fork and check for error */
 perror ("error in fork");
 exit (1);
}
if (0 == pid) { /* child process */
    printf ("The child process begins.\n");
    for (i_child = 0; i_child < run_length; i_child++)  {
        sleep(1);  /* wait for memory to be updated */
        value = *shared_memory;
        printf ("Child's report:  current value = %2d\n", value); }
    printf ("The child is done\n");
  }  else  {    /* parent process */
    printf ("The parent process begins.\n");
    for (i_parent = 0; i_parent < run_length; i_parent++)  {
        *shared_memory = i_parent * i_parent;/* square into shared memory */
        printf ("Parent's report: current index = %2d\n", i_parent );
        sleep(1);  /* wait for child to read value */  }
    wait(pid);
    printf ("The parent is done\n");  }
  exit (0);  }
```

Experimente mit Shared Memory

- Übersetzen Sie den Quelltext von den vorigen beiden Folien, untersuchen Sie die Ausgabe und erklären Sie diese
- Entfernen sie den `sleep`-Aufruf aus dem Kindprozess und beschreiben Sie den Effekt
- Fügen Sie den `sleep`-Aufruf im Kindprozess wieder hinzu und untersuchen Sie, was passiert, wenn Sie diesen im Elternprozess entfernen
- Überlegen Sie sich, wie die Koordination der beiden Prozesse mittels *Sperren* anstelle des `sleep` hätte implementiert werden können
  - Machen Sie sich mit der Funktionsweise von *Sperren* vertraut: `http://de.wikipedia.org/wiki/Spinlock`
  - **Tipp:** Der Elternprozess sollte in das *shared-memory word* schreiben, wenn es den Wert -1 enthält. Anderenfalls sollte es der Kindprozess auslesen.