– Lösung zur Praktikumsaufgabe 9 –

Thema: *Echtzeitfähige Speicherverwaltung und Betriebssysteme*

1. Die angegebene Literaturstelle führt fünf verschiedene Verfahren auf, die Position des höchstwertigen gesetzten Bits in einem Wort zu finden. Der Code in Listing 1 vergleicht die Ausführungszeit von jeweils 100000 Iterationen für verschiedene Positionen des gesuchten Bits.

Listing 1: Vermessung verschiedener Algorithmen für die FFS-Operation

```
/*
  find the highest bit set in a word, several implementations

  cf. http://stackoverflow.com/questions/671815/what-is-the-fastest-
      most-efficient-way-to-find-the-highest-set-bit-msb-in-an-i
  https://stackoverflow.com/questions/2589096/find-most-significant-
      bit-left-most-that-is-set-in-a-bit-array

   The bsr instruction is described on page 568 of
   64-ia-32-architectures-software-developer-manual-325462.pdf
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <limits.h>
#include <x86intrin.h>

/* Number of tests per call */
#define ITERATIONS 100000UL

static inline uint64_t rdtsc()
{
  uint64_t tsc;

  __asm__(
          "cpuid          \n"    /* no out-of-order execution */
          "rdtsc          \n"
          :
          "=A" (tsc)
          :
          : "ebx", "ecx"           /* cpuid scratcht eax, ebx, ecx, edx
              */
          );
  return tsc;
}

/*
    https://stackoverflow.com/questions/2589096/find-most-significant-
        bit-left-most-that-is-set-in-a-bit-array
    o If the algorithm is used as published, the result is off-by-one
        (hence, -1).
 */
static inline unsigned int ffs_noname(unsigned int x)
{
```

```
    static const unsigned int bval[] =
    {0,1,2,2,3,3,3,3,4,4,4,4,4,4,4,4};

    unsigned int r = 0;
    if (x & 0xFFFF0000) { r += 16/1; x >>= 16/1; }
    if (x & 0x0000FF00) { r += 16/2; x >>= 16/2; }
    if (x & 0x000000F0) { r += 16/4; x >>= 16/4; }
    return r + bval[x] - 1;
}

/*
   "de Brujin multiplication"
   o only works with -m32
*/
static inline uint32_t ffs_dbm(uint32_t segsize) {
  static const uint32_t pos[32] = {0, 1, 28, 2, 29, 14, 24, 3,
    30, 22, 20, 15, 25, 17, 4, 8, 31, 27, 13, 23, 21, 19,
    16, 7, 26, 12, 18, 6, 11, 5, 10, 9};
  segsize |= segsize >> 1;
  segsize |= segsize >> 2;
  segsize |= segsize >> 4;
  segsize |= segsize >> 8;
  segsize |= segsize >> 16;
  segsize = (segsize >> 1) + 1;
  return pos[(segsize * 0x077CB531UL) >> 27];
}

static inline uint32_t ffs_naive(uint32_t segsize)
{
  int ret = 0;

  while (segsize >>= 1)
    ret++;
  return ret;
}

static inline uint32_t ffs_intrinsic(uint32_t segsize)
{
  return 31 - __builtin_clz(segsize);
}

static inline uint32_t ffs_asm(uint32_t segsize)
{
  unsigned int ret;

  asm("bsrl %1,%0" : "=r"(ret) : "r"(segsize));
  return ret;
}

int main(int argc, char* argv[])
{
  int c, d;
  unsigned long r, ret;
  unsigned long long a, b;
```

```
  /* Outer loop determines size of r */
  printf("Every algorithm is executed %ld times.", ITERATIONS);
  for (r=1, d=0; r!=0; r<<=1, d++) {
    printf("\nTesting with r=%lu (bit set at position %d)\n", r, d);

    printf("** bsr (bit-scan-reverse) machine instruction\n");
    a = rdtsc();
    for (c=0; c<ITERATIONS; c++) {
      ret = ffs_asm(r);
    }
    b = rdtsc();
    printf("ffs_asm() cycles=%llu, ret=%lu\n", (b-a), ret);

    printf("** gcc intrinsic __builtin_clz(r)\n");
    a = rdtsc();
    for (c=0; c<ITERATIONS; c++) {
      ret = ffs_intrinsic(r);
    }
    b = rdtsc();
    printf("ffs_intrinsic() cycles=%llu, ret=%lu\n", (b-a), ret);

    printf("** Naive implementation\n");
    a = rdtsc();
    for (c=0; c<ITERATIONS; c++) {
      ret = ffs_naive(r);
    }
    b = rdtsc();
    printf("ffs_naive() cycles=%llu, ret=%lu\n", (b-a), ret);

    printf("** deBrujin multiplication\n");
    a = rdtsc();
    for (c=0; c<ITERATIONS; c++) {
      ret = ffs_dbm(r);
    }
    b = rdtsc();
    printf("ffs_dbm() cycles=%llu, ret=%lu\n", (b-a), ret);

    printf("** noname algorithm\n");
    a = rdtsc();
    for (c=0; c<ITERATIONS; c++) {
      ret = ffs_noname(r);
    }
    b = rdtsc();
    printf("ffs_noname() cycles=%llu, ret=%lu\n", (b-a), ret);

  }
  exit(0);
}
```

Abbildung 1 zeigt das Zeitverhalten aller Verfahren. Testplattform ist ein Linux-Notebook mit CPU AMD Ryzen 5 5500U. Es ist ersichtlich, dass nur die naive Implementierung (Linksschieben des Wertes, bis 0 resultiert) eine variable Ausführungszeit hat. Sie hängt von der konkreten Position des ersten gesetzten Bits ab.

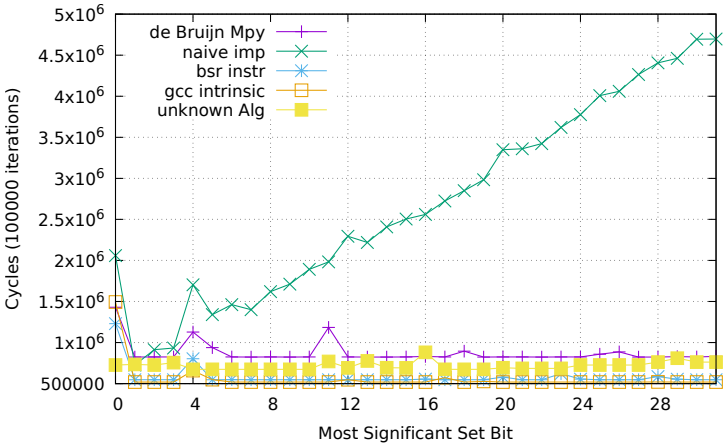

Abbildung 1: Timing für fünf verschiedene Implementationen der FFS-Operation

Abbildung 2 zeigt eine „Ausschnittvergrößerung“, um die Effizienz der einzelnen Verfahren zu vergleichen.

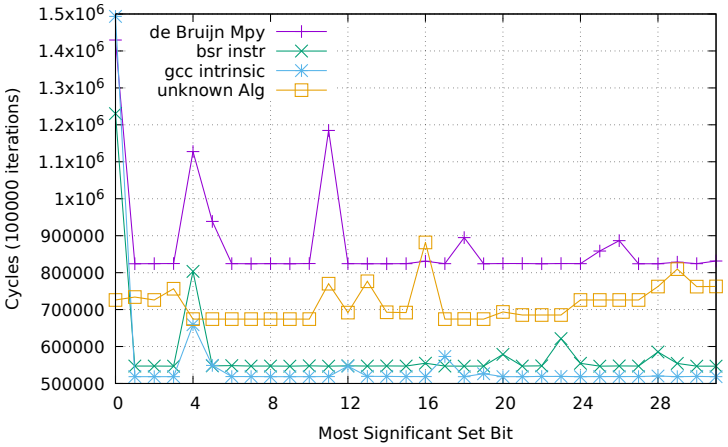

Abbildung 2: Timing für die vier Verfahren mit konstanter Ausführungszeit

2.
```
/*
   rtsignal1.c

   send MAXVALUE RT signals between father and son.
   - son installs handler (sigaction())
   - father sends signals (sigqueue())
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

#define MAXVALUE 5

int end=0;
```

```
void sigrtmin_handler(int num, siginfo_t *si, void *uc)
{
  printf("Received Signal No. %d.\n", num);
  printf("Sender PID: %u, Value: %d.\n", si->si_pid, si->si_value.
      sival_int);

  if (si->si_value.sival_int == MAXVALUE) {
    end=1;
  }
}

/* Son's code */
void son(void)
{
  struct sigaction sa;
  int ret;

  /* establish handler for SIGRTMIN+1 */
  sa.sa_sigaction = sigrtmin_handler;
  sa.sa_flags = SA_SIGINFO;
  ret = sigaction(SIGRTMIN+1, &sa, NULL);
  if (ret == -1) {
    perror("sigaction()");
    exit(EXIT_FAILURE);
  }
  while (!end) {
    pause();
    printf("Son has been woken.\n");
  }

  printf("Son ends.\n");
  exit(EXIT_SUCCESS);
}


int main(int argc, char *argv[])
{
  pid_t sonpid;
  int ret, c=0;
  union sigval sv;

  printf("Numbers of RT signals are between %d and %d\n", SIGRTMIN,
      SIGRTMAX);

  sonpid = fork();
  if (sonpid == -1) {
    perror("fork()");
    exit(EXIT_FAILURE);
  }
  if (sonpid == 0) {
    son();
    /* never returns */
  }

  for (c=0; c<=MAXVALUE; c++) {
```

```
      sleep(1);
      sv.sival_int = c;
      ret = sigqueue(sonpid, SIGRTMIN+1, sv);
      if (ret == -1) {
        perror("sigqueue()");
      }
    }

    waitpid(sonpid, NULL, 0);
    printf("Father ends.\n");
    exit(EXIT_SUCCESS);
  }
```

3.