CS 33

Multithreaded Programming VII

Implementing Mutexes

- Strategy
  - make the usual case (no waiting) very fast
  - can afford to take more time for the other case (waiting for the mutex)

Futexes

- Safe, *efficient* kernel conditional queueing in Linux
- All operations performed atomically
  - `futex_wait(futex_t *futex, int val)`
    - » if `futex->val` is equal to `val`, then sleep
    - » otherwise return
  - `futex_wake(futex_t *futex)`
    - » wake up one thread from `futex`'s wait queue, if there are any waiting threads

Ancillary Functions

- `int atomic_inc(int *val)`
  - **add 1 to `*val`, return its original value**
- `int atomic_dec(int *val)`
  - **subtract 1 from `*val`, return its original value**
-
```
int CAS(int *ptr, int old, int new) {
    int tmp = *ptr;
    if (*ptr == old)
        *ptr = new;
    return tmp;
}
```

Attempt 1

```
void lock(futex_t *futex) {
  int c;
  while ((c = atomic_inc(&futex->val)) != 0)
    futex_wait(futex, c+1);
}

void unlock(futex_t *futex) {
  futex->val = 0;
  futex_wake(futex);
}
```


Attempt 2

```
void lock(futex_t *futex) {
  int c;
  if ((c = CAS(&futex->val, 0, 1) != 0)
    do {
      if (c == 2 || (CAS(&futex->val, 1, 2) != 0))
        futex_wait(futex, 2);
    while ((c = CAS(&futex->val, 0, 2)) != 0))
}

void unlock(futex_t *futex) {
  if (atomic_dec(&futex->val) != 1) {
    futex->val = 0;
    futex_wake(futex);
  }
}
```


Memory Allocation

- Multiple threads
- One heap

Bottleneck?

Solution 1

- Divvy up the heap among the threads
  - each thread has its own heap
  - no mutexes required
  - no bottleneck
- How much heap does each thread get?

Solution 2

- **Multiple “arenas”**
  - each with its own mutex
  - thread allocates from the first one it can find whose mutex was unlocked
    - » if none, then creates new one
  - deallocations go back to original arena

Solution 3

- Global heap plus per-thread heaps
  - threads pull storage from global heap
  - freed storage goes to per-thread heap
    - » unless things are imbalanced
      - then thread moves storage back to global heap
  - mutexes on each heap
- What if one thread allocates and another frees storage?

Malloc/Free Implementations

- ptmalloc
  - based on solution 2
  - in glibc (i.e., used by default)
- tcmalloc
  - based on solution 3
  - from Google
- Which is best?

Test Program

```
const unsigned int N=64, nthreads=32, iters=10000000;
int main() {
  void *tfunc(void *);
  pthread_t thread[nthreads];
  for (int i=0; i<nthreads; i++) {
    pthread_create(&thread[i], 0, tfunc, (void *)i);
    pthread_detach(thread[i]);
  }
  pthread_exit(0);
}
void *tfunc(void *arg) {
  long i;
  for (i=0; i<iters; i++) {
    long *p = (long *)malloc(sizeof(long)*((i%N)+1));
    free(p);
  }
  return 0;
}
```

Not a Quiz

Which is fastest?

a) glibc (i.e., standard Linux)

b) Google

Compiling It ...

```
% gcc -o ptalloc alloc.c -lpthread
% gcc -o tcalloc alloc.c -lpthread -ltcmalloc
```


Running It (2014) …

```
$ time ./ptalloc
real    0m5.142s
user    0m20.501s
sys     0m0.024s
$ time ./tcalloc
real    0m1.889s
user    0m7.492s
sys     0m0.008s
```


Running It (2023) …

```
$ time ./ptalloc
real 0m0.666s
user 0m5.815s
sys 0m0.004s
$ time ./tcalloc
real 0m0.496s
user 0m4.197s
sys 0m0.008s
```


What's Going On (2014)?

```
$ strace -c -f ./ptalloc
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.040002          13      3007       520 futex
 …


$ strace -c -f ./tcalloc
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 …
  0.00    0.000000           0        59        13 futex
 …
```

What's Going On (2023)?

```
$ strace -c -f ./ptalloc
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 …
  0.02    0.000001           0         5         1 futex
 …


$ strace -c -f ./tcalloc
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 …
  0.26    0.000006          0        23         3 futex
 …
```


Test Program 2, part 1

```
#define N 64
#define npairs 16
#define allocsPerIter 1024
const long iters = 8*1024*1024/allocsPerIter;
#define BufSize 10240
typedef struct buffer {
  int *buf[BufSize];
  unsigned int nextin;
  unsigned int nextout;
  sem_t empty;
  sem_t occupied;
  pthread_t pthread;
  pthread_t cthread;
} buffer_t;
```

Test Program 2, part 2

```
int main() {
  long i;
  buffer_t b[npairs];
  for (i=0; i<npairs; i++) {
    b[i].nextin = 0;
    b[i].nextout = 0;
    sem_init(&b[i].empty, 0, BufSize/allocsPerIter);
    sem_init(&b[i].occupied, 0, 0);
    pthread_create(&b[i].pthread, 0, prod, &b[i]);
    pthread_create(&b[i].cthread, 0, cons, &b[i]);
  }
  for (i=0; i<npairs; i++) {
    pthread_join(b[i].pthread, 0);
    pthread_join(b[i].cthread, 0);
  }
  return 0;
}
```

Test Program 2, part 3

```
void *prod(void *arg) {
  long i, j;
  buffer_t *b = (buffer_t *)arg;
  for (i = 0; i<iters; i++) {
    sem_wait(&b->empty);
    for (j = 0; j<allocsPerIter; j++) {
      b->buf[b->nextin] = malloc(sizeof(int)*((j%N)+1));
      if (++b->nextin >= BufSize)
        b->nextin = 0;
    }
    sem_post(&b->occupied);
  }
  return 0;
}
```

Test Program 2, part 4

```
void *cons(void *arg) {
  long i, j;
  buffer_t *b = (buffer_t *)arg;
  for (i = 0; i<iters; i++) {
    sem_wait(&b->occupied);
    for (j = 0; j<allocsPerIter; j++) {
      free(b->buf[b->nextout]);
      if (++b->nextout >= BufSize)
        b->nextout = 0;
    }
    sem_post(&b->empty);
  }
  return 0;
}
```


Running It (2014) ...

```
$ time ./ptalloc2
real    0m1.087s
user    0m3.744s
sys     0m0.204s
$ time ./tcalloc2
real    0m3.535s
user    0m11.361s
sys     0m2.112s
```


Running It (2023) …

```
$ time ./ptalloc2
real     0m0.365s
user     0m1.378s
sys      0m0.536s
$ time ./tcalloc2
real     0m8.019s
user     1m1.348s
sys      0m7.161s
```

What's Going On (2014)?

```
$ strace -c -f ./ptalloc2
% time     seconds  usecs/call     calls    errors syscall
 …
------ ----------- ----------- --------- --------- ----------------
 …
 93.04    8.246196         117     70173     20775 futex
…
$ strace -c -f ./tcalloc2
% time     seconds  usecs/call     calls    errors syscall
 …
------ ----------- ----------- --------- --------- ----------------
99.92   47.796676         153    311012      7244 futex
 …
```


What's Going On (2023)?

```
$ strace -c -f ./ptalloc2
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 98.48   42.883196          79    539757    179723 futex
…
$ strace -c -f ./tcalloc2
 …
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.99  346.746205         146   2372684     44547 futex
 …
```


You'll Soon Finish CS 33 …

- You might
  - celebrate

  - take another systems course
    - » 320
    - » 1380
    - » 1660
    - » 1670
    - » 1680

  - become a 33 TA

Systems Courses Next Semester

- **CS 320 (Intro to Software Engineering)**
  - you've mastered low-level systems programming
  - now do things at a higher level
  - learn software-engineering techniques using Java, XML, etc.
- **CS 1380 (Distributed Systems)**
  - you now know how things work on one computer
  - what if you've got lots of computers?
  - some may have crashed, others may have been taken over by your worst (and smartest) enemy
- **CS 1660/1620/2660 (Computer Systems Security)**
  - liked buffer?
  - you'll really like 1660
- **CS 1670/1690/2670 (Operating Systems)**
  - still mystified about what the OS does?
  - write your own!

The End

Well, not quite …

Database is due on 12/15

Happy Coding and Happy Holidays!