Part IV
Other Systems: III
Pthreads: A Brief Review

An algorithm must be seen to be believed.

Donald Erwin Knuth

The POSIX Standard: 1/2

- **POSIX (*P*ortable *O*perating *S*ystem *I*nterfaces) is a family of standards for maintaining compatibility between operating systems.**
- **POSIX is a Unix-like operating system environment and is currently available on Unix/Linux, Windows, OS/2 and DOS.**

The POSIX Standard: 2/2

- **Pthreads (POSIX Threads) is a POSIX standard for threads.**
- **The standard, POSIX.1c thread extension, defines thread creation and manipulation.**
- **This standard defines thread management, mutexes, conditions, read/write locks, barriers, etc.**
- **Except for the monitors, all features are available in Pthreads.**

Thread Creation

- **Always includes the `pthread.h` header file.**

```
int pthread_create(
      pthread_t             *tid,
      const pthread_attr_t  *attr,
      void                  *(*start)(void *),
      void                  *arg);
```

- **`pthread_create()` creates a thread and runs function `start()` with argument list `arg`.**
- **`attr` specifies optional creation attributes.**
- **The ID of the newly created thread is returned with `tid`.**
- **Non-zero return value means creation failure.**

Thread Join

- **Use `pthread_join()` to join with a thread.**
- **The following waits for `thread` to complete, and returns `thread`'s exit value if `value_ptr` is not `NULL`. Use `NULL` if you don't use exit value.**
- **Join failed if `pthread_join()` returns a non-zero value.**

```
int pthread_join(
     pthread_t  thread,
     void       **value_ptr);
```

Thread Exit

- **Use `pthread_exit()` to terminate a thread and return the value `value_ptr` to any joining thread.**
- **Exit failed if `pthread_exit()` returns a non-zero value.**
- **Use `NULL` for `value_ptr` if you don’t use exit value.**

```
int pthread_exit(
     pthread_t  thread,
     void       *value_ptr);
```

Mutex: 1/2

- A mutex has a type `pthread_mutex_t`.
- Mutexes initially are unlocked.
- Only the owner can unlock a mutex.
- Since mutexes cannot be copied, use pointers.
- Use `pthread_mutex_destroy()` to destroy a mutex. Make sure no thread is blocked inside.

```
pthread_mutex_t   mutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_init(
     pthread_mutex_t      *mutex,
     pthread_mutexattr_t  *attr);
int pthread_mutex_destroy(
     pthread_mutex_t      *mutex);
```

Mutex: 2/2

- If `pthread_mutex_trylock()` returns `EBUSY`, the lock is already locked. Otherwise, the calling thread becomes the owner of this lock.
- With `pthread_mutexattr_settype()`, the type of a mutex can be set to allow recursive locking or report deadlock if the owner locks again.

```
int pthread_mutex_lock(
     pthread_mutex_t      *mutex);
int pthread_mutex_unlock(
     pthread_mutex_t      *mutex);
int pthread_mutex_trylock(
     pthread_mutex_t      *mutex);
```

Condition Variables: 1/2

- **Conditions in Pthreads are usually used with a mutex to enforce mutual exclusion.**

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_init(
     pthread_cond_t             *cond,
     const pthread_condattr_t  *attr);
int pthread_cond_destroy(
     pthread_cond_t             *cond);
int pthread_cond_wait(
     pthread_cond_t             *cond,
     pthread_mutex_t            *mutex);
int pthread_cond_signal(
     pthread_cond_t             *cond);
int pthread_cond_broadcast(
     pthread_cond_t             *cond);
```

Condition Variables: 2/2

- `pthread_cond_wait()` **and** `pthread_cond_signal()` **are the** `wait()` **and** `signal()` **methods in *ThreadMentor*, and are** `wait()` **and** `notify()` **in Java.**
- `pthread_cond_signal()` **uses Mesa type and the released thread must recheck the condition.**

```
int pthread_cond_wait(
     pthread_cond_t            *cond,
     pthread_mutex_t           *mutex);
int pthread_cond_signal(
     pthread_cond_t            *cond);
int pthread_cond_broadcast(
     pthread_cond_t            *cond);
```

Simulating a Mesa Monitor: 1/2

- Use a mutex for protecting the monitor.
- Lock and unlock this mutex upon entering and exiting the monitor.
- When a thread calls a condition wait, it relinquishes the monitor mutex. Once blocked, the monitor mutex becomes available to other threads.
- The released thread (from a condition wait) becomes the new owner of the monitor mutex.

Simulating a Mesa Monitor: 2/2

```
pthread_mutex_t MonitorLock = PTHREAD_MUTEX_INITIALIZER;
Pthread_cond_t  cond = PTHREAD_COND_INITIALIZER;
```

monitor procedure

```
pthread_mutex_lock(&MonitorLock); // enter the monitor
   // other statements
   while (condition is not met)       // this is a Mesa type
      pthread_cond_wait(&cond, &MonitorLock);
   // other statements
pthread_mutex_unlock(&MonitorLock);  // exit monitor
```

monitor procedure

```
pthread_mutex_lock(&MonitorLock); // enter the monitor
   // other statements
   // cause condition to happen
   pthread_cond_signal(&cond);
   // other statements
pthread_mutex_unlock(&MonitorLock);  // exit monitor
```

Example: Reader-Writer: 1/3

- **If a writer is waiting, then readers should wait its turn. See pp. 22-26 of `10-Monitor.pdf`,**

```
pthread_mutex_t  m_lock;
pthread_cond_t   turn;

int reading;
int writing;
int Writers;

void initialization(void)
{
    pthread_mutex_init(&m_lock, NULL);
    pthread_cond_init(&turn, NULL);
    reading = writing = Writers = 0;
}
```

Example: Reader-Writer: 2/3

reader

Is the use of `if` good enough? Why?

```
void reader(void)
{
   pthread_mutex_lock(&m_lock); // lock monitor
   if (writers > 0)
      pthread_cond_wait(&turn, &m_lock);
   while (writing > 0)
      pthread_cond_wait(&turn, &m_lock);
   reading++;
   pthread_mutex_unlock(&m_lock);  // unlock monitor
   /* reading */
   pthread_mutex_lock(&m_lock); // lock monitor
   reading--;
   pthread_cond_signal(&turn);
   pthread_mutex_unlock(&m_lock); // unlock monitor
}
```

using `broadcast` can release multiple threads (e.g., readers)

Example: Reader-Writer: 3/3

writer

```
void writer(void)
{
   pthread_mutex_lock(&m_lock); // lock monitor
   writer++;
   while (reading > 0 || writing > 0)
      pthread_cond_wait(&turn, &m_lock);
   writing++;
   pthread_mutex_unlock(&m_lock);  // unlock monitor
   /* writing */
   pthread_mutex_lock(&m_lock); // lock monitor
   writing--;
   writers--;
   pthread_cond_signal(&turn);
   pthread_mutex_unlock(&m_lock); // unlock monitor
}
```

using `broadcast` can release multiple threads (e.g., readers)

Simulating a Hoare Monitor

- **Simulating a Hoare type monitor requires the use of general semaphores.**
- **The Pthreads standard does not have semaphores. Instead, POSIX.1b standard has the Unix semaphores.**
- **With POSIX.1b semaphores, it is easy to simulate a Hoare type monitor. Many OS textbooks discuss such a simulation. Also see our reading lists for such a solution.**

Languages vs. Libraries: 1/2

- **Libraries are extension to a sequential language.**
- **Programmers may try various approaches that fit his/her needs. Programs can be deployed without requiring any changes in the tools (e.g., compiler).**
- **Libraries may not be well-defined and completely portable. Some features may be difficult to define and/or implement (e.g., Hoare type monitors).**
- **Programs may be difficult to understand because API function calls can scatter everywhere and sometimes cryptic.**

Languages vs. Libraries: 2/2

- **With the language-based approach, the intent of the programmer is easier to express and understand, both by other programmers and by program analysis tools.**
- **Languages usually require the standardization of new constructs and perhaps new keywords.**
- **Language features are fixed. Each language may only support one or a few concurrent programming models, and may not be very flexible.**

The End

Sorting 10 Numbers w/o Array: 1/2

```
#include <stdio.h>

#define  MAX(a,b) (a > b ? a : b)
#define  MIN(a,b) (a < b ? a : b)

void main(void)
{
   int a, b, c, d, e, f, g, h, i, j;
   int max, min, k
   scanf("%d%d%d%d%d%d%d%d%d%d", &a, …, &j);
   min = MIN(MIN(MIN(MIN(MIN(MIN(MIN(MIN(MIN(a,b),c),d),e),f),g),h),i),j);
   max = MAX(MAX(MAX(MAX(MAX(MAX(MAX(MAX(MAX(a,b),c),d),e),f),g),h),i),j);
   for (k = min; k <= max; k++) {
      if (k == a) printf("%10\n", k);
      if (k == b) printf("%10\n", k);
      if (k == c) printf("%10\n", k);
      if (k == d) printf("%10\n", k);
      if (k == e) printf("%10\n", k);
      if (k == f) printf("%10\n", k);
      /*  other comparisons */
      if (k == j) printf("%10\n", k);
   }
}
```

Sorting 10 Numbers w/o Array: 2/2

```
#include <stdio.h>
#include <limits.h>

#define MAX          INT_MAX
#define MIN(x,y)     ((*x) <= (*y) ? (x) : (y))

void main(void)
{
   int a, b, c, d, e, f, g, h, i, j, *p;
   int count;

   scanf("%d%d%d%d%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f,&g,&h,&i,&j);

   for(count = 1; count <= 10; count++) {
      p = MIN(MIN(MIN(MIN(&a,&b),MIN(&c,&d)),MIN(MIN(&e,&f),MIN(&g,&h))),
                        MIN(&i,&j));
      printf("%10d ", *p);
      *p = MAX;
   }
   printf("\n");
}
```

Old Computers as Promised

Osborne 1
April 3, 1981

The first commercially successful portable computer

The first portable computer is perhaps the IBM 5100. This one uses BASIC and APL (i.e., *A P*rogramming *L*anguage)

Old Computers as Promised

Osborne 1
April 3, 1981

The first commercially successful portable computer

Old Computers as Promised

Toshiba T4400C
1990

World first TFT
486 Battery Powered Notebook

Memory: 36MB
Disk: 320MB

Old Computers as Promised

Toshiba T4400C
1990

World first TFT
486 Battery Powered
Notebook

Memory: 36MB
Disk: 320MB

Old Computers as Promised

Toshiba T4400C 1990

Windows 3.1!

Old Computers as Promised

Apple Mac G4 Cube
July 19, 2000

CPU: 500 MHz Power PC G4
OS: Mac OS 9

Old Computers as Promised

Harmon Kardon Speaker