POSIX IPC

Matt Ruffalo

March 26, 2013

Recap: POSIX Threads

- Lightweight
- One process can spawn many threads
- Memory is shared between threads (as opposed to shared copy-on-write data after a `fork()`)

Recap: POSIX Threads API

- Creating: `fork()` → `pthread_create(...)`
- Returning/Exiting: `exit(...)` → `pthread_exit(...)`
- Waiting: `wait(...)` → `pthread_join(...)`

Threads API: #include <pthread.h>

Creating threads

```
int pthread_create(pthread_t *thread,
    const pthread_attr_t *attr,
    void *(*start_routine) (void *), void *arg);
```

- `thread`: thread handle is stored here
- `attr`: Attributes (can be `NULL`)
- `start_routine`: Pointer to function to run in the thread
  - Function must take a `void*` argument and return a `void*`
- `arg`: *Single* argument that will be passed to the method

Threads API: #include <pthread.h>

Closing threads

```
void pthread_exit(void *retval);
```

- `retval`: value passed here will be available to whatever called `pthread_join` on this thread
- Careful: you may cause a segfault if the value goes out of scope when the thread ends!

Threads API: #include <pthread.h>

Waiting on threads

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

- Returns 0 on success; error number otherwise
- `retval`: value returned by the thread in `pthread_exit` will be stored here

Sleeping

Don't use `sleep()`

- The entire process is put to sleep, not just the calling thread
- Probably a good idea to use nanosleep instead (affects a single thread)

nanosleep: #include <time.h>

Waiting on threads

```
int nanosleep(const struct timespec *req,
    struct timespec *rem);

struct timespec {
    time_t tv_sec;          /* seconds */
    long   tv_nsec;         /* nanoseconds */
};
```

Required: `tv_nsec` $< 1,000,000,000$

POSIX Semaphores

```
#include <semaphore.h>
```

- Single semaphores are created at a time
- Identifiers are strings, not numeric
- Integrates with `pthreads` API: semaphores can optionally be accessed only in one process and its threads

POSIX Semaphores: Named

sem_open

```
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,
    mode_t mode, unsigned int value);
```

- Returns pointer to semaphore handle
- `name`: should start with `/`
- `oflag`: flags for opening/creation; can include `O_CREAT` and `O_EXCL`
- `mode`: permissions
- `value`: Initial value

POSIX Semaphores: Unnamed

sem_init

```
int sem_init(sem_t *sem, int pshared,
    unsigned int value);
```

- Returns 0 on success, -1 on failure
- `sem`: semaphore handle is stored here
- `pshared`: 1: share between processes; 0: accessible only in threads of this process
  - To share between different processes, store semaphore handle in a shared memory segment
- `value`: Initial value

POSIX Shared Memory: API

```c
#include <sys/mman.h>
#include <sys/stat.h> /* For mode constants */
#include <fcntl.h>    /* For O_* constants */

int shm_open(const char *name, int oflag, mode_t mode);
int shm_unlink(const char *name);
```

POSIX Shared Memory: Usage

shm_open

```
int shm_open(const char *name, int oflag,
    mode_t mode);
```

- Returns a shared memory file descriptor, or -1 on failure
- `name`: should start with `/`
- `oflag`: flags for opening/creation; can include `O_CREAT` and `O_EXCL`
- `mode`: permissions

POSIX Shared Memory: Deletion

shm_unlink

```
int shm_unlink(const char *name);
```

- Returns 0 on success, -1 on failure
- name: same as was used in shm_open

POSIX Shared Memory: Accessing

mmap, munmap

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot,
    int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```

- Attaches a file descriptor to a segment of memory (*e.g.* what's returned by `shm_open`)
- Can use in the same way as the pointer returned by `shmat`
- `munmap` when done, of course