POS04-C. Avoid using PTHREAD_MUTEX_NORMAL type mutex locks

Pthread mutual exclusion (mutex) locks are used to avoid simultaneous usage of common resources. Several types of mutex locks are defined by pthreads: `NORMAL`, `ERRORCHECK`, `RECURSIVE`, and `DEFAULT`.

POSIX describes `PTHREAD_MUTEX_NORMAL` locks as having the following undefined behavior [Open Group 2004]:

> *This type of mutex does not provide deadlock detection. A thread attempting to relock this mutex without first unlocking it shall deadlock. An error is not returned to the caller. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior.*

The `DEFAULT` mutex pthread is also generally mapped to `PTHREAD_MUTEX_NORMAL` but is known to vary from platform to platform [SOL 2010]. Consequently, `NORMAL` locks should not be used, and `ERRORCHECK` or `RECURSIVE` locks should be defined explicitly when mutex locks are used.

Noncompliant Code Example

This noncompliant code example shows a simple mutex being created using `PTHREAD_MUTEX_NORMAL`. Note that the caller does not expect a return code when `NORMAL` mutex locks are used.

```
pthread_mutexattr_t attr;
pthread_mutex_t mutex;
size_t const shared_var = 0;

int main(void) {
  int result;

  if ((result = pthread_mutexattr_init(&attr)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_NORMAL)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutex_init(&mutex, &attr)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutex_lock(&mutex)) != 0) {
    /* Handle Error */
  }

  /* Critical Region*/

  if ((result = pthread_mutex_unlock(&mutex)) != 0) {
    /* Handle Error */
  }

  return 0;
}
```

Compliant Solution

This compliant solution shows an `ERRORCHECK` mutex lock being created so that return codes will be available during locking and unlocking:

```
pthread_mutexattr_t attr;
pthread_mutex_t mutex;
size_t const shared_var = 0;

int main(void) {
  int result;

  if ((result = pthread_mutexattr_init(&attr)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_ERRORCHECK)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutex_init(&mutex, &attr)) != 0) {
    /* Handle Error */
  }
  if ((result = pthread_mutex_lock(&mutex)) != 0) {
    /* Handle Error */
  }

  /* Critical Region*/

  if ((result = pthread_mutex_unlock(&mutex)) != 0) {
    /* Handle Error */
  }

  return 0;
}
```

Risk Assessment

Using `NORMAL` mutex locks can lead to deadlocks or abnormal program termination.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
POS04-C	Low	Unlikely	Medium	P2	L3

Bibliography

[Open Group 2004]
[SOL 2010]