libatomic Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Characteristics

libatomic provides functions for updating shared variables in parallel programs with indivisible (atomic) operations.

Since updates are executed with indivisible actions, even when a single variable is operated in parallel from several programs, the intermediate states during the processing cannot be monitored from the other programs.

libatomic is available on both the PlayStation®Vita and Win32 environments.

Embedding into a Program

Include sce_atomic.h in the source program (various header files will be automatically included as well).

2 Using the Library

Basic Procedure

The values of variables can be updated through indivisible operations by calling functions with the addresses of values in the shared memory to be updated as arguments.

The following code increments the value of the *val* variable by an indivisible operation.

```
#include <sce_atomic.h>

volatile int32_t val = 0;
sceAtomicAdd32(&val, 1);
```

3 Library Operation

Atomic Update System

libatomic is implemented using CPU atomic memory operation. System calls are not called internally.

Memory Orderings

When a function such as `sceAtomicAdd32Acquire()` has `Acquire` at the end, it guarantees that the memory operations performed by this function will always be executed before later memory operations.

When a function such as `sceAtomicAdd32Release()` has `Release` at the end, it guarantees that the memory operations performed before this function was called will always be executed before the memory operations of this function in other threads.

When a function such as `sceAtomicAdd32AcqRel()` has `AcqRel` at the end, it guarantees that the memory operations performed before this function was called will always be executed before the memory operations of this function, and guarantees that the memory operations of this function will always be executed before later memory operations.

When a function such as `sceAtomicAdd32Relaxed()` has `Relaxed` at the end or when a function such as `sceAtomicAdd32()` has nothing at the end, it does not guarantee the order of memory operations before or after.