libfiber Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Purpose and Characteristics

libfiber is a library that provides fibers. Fibers are individual execution units that operate on threads provided by the kernel, and have individual contexts and processing flows. By using fibers, application programs can utilize multiple processing flows, switching between and executing them as needed.

Note
As a general rule, the term "fiber" when used in this document refers to a fiber provided by libfiber.

Fibers have the following characteristics.

They do not require system calls, and therefore have a low switching cost

Fibers are implemented at the user level, and as such do not require kernel calls for switching or state changes.

There is no limit to how many fibers there may be

Because the creation of a fiber does not require the resources of the kernel, there is no limitation on the number of fibers that can be created. As many fibers may be generated as the memory allows.

Fibers to be used can be individually started and switched

Individual fibers to be used are explicitly specified in the fiber startup / switching interface.

Files

The files required for using libfiber are as follows.

Filename	Description
sce_fiber.h	Header file
libSceFiber_stub.a	Stub library file
libSceFiber_stub_weak.a	Weak import stub library file

Sample Programs

The sample programs provided for libfiber are as follows.

Basic Fiber Feature Sample (under sample_code/system/api_libfiber/)

These samples exemplify the usage of the basic fiber features.

Sample Program	Description
sample_fiber_basic	Sample using basic fiber features. 16 fibers are started.
sample_fiber_perf	Sample measuring the execution time of basic fiber APIs.

Reference Materials

Details of Each API

- libfiber Reference

2 Basic Fiber Specifications

Fiber Contexts

Individual fibers have the following contexts.

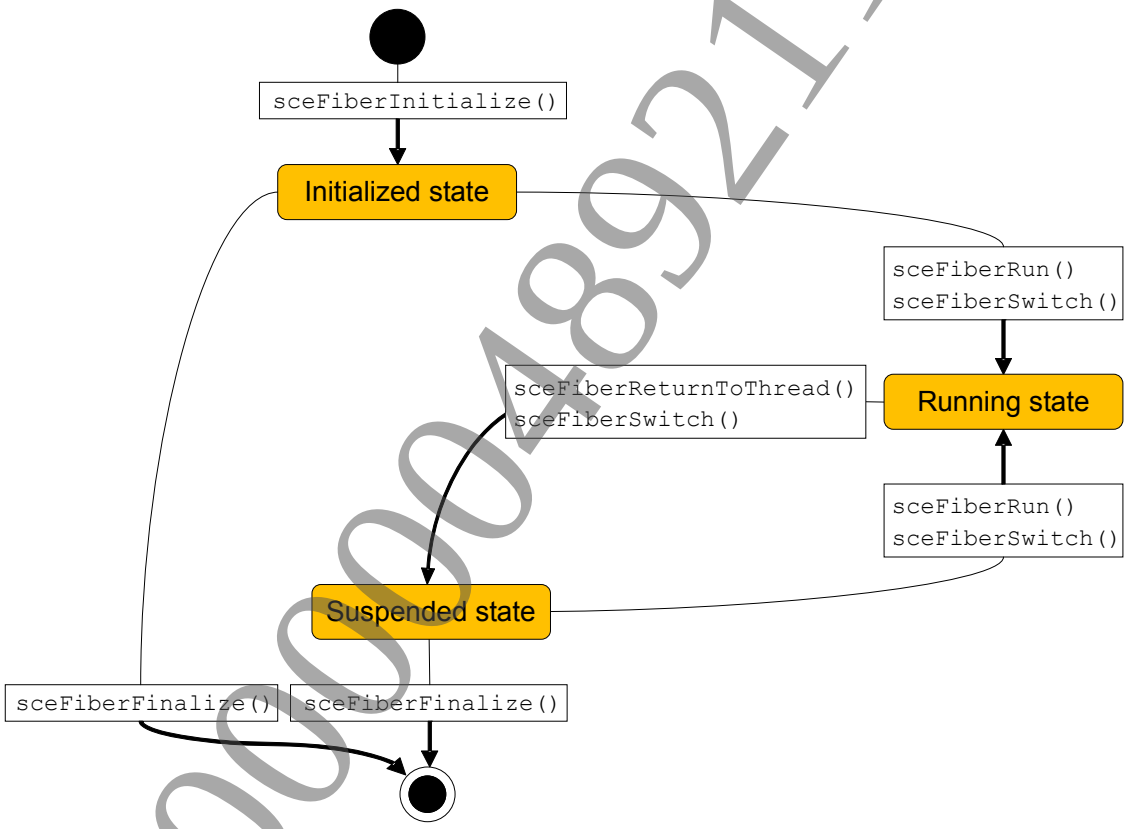
- Stack
- Register

Other thread resources, such as the Thread Local Storage and the process's resources, such as text and data areas, are shared.

Fiber State Transitions

The state transitions of a fiber are as follows.

Figure 1 Fiber State Transitions

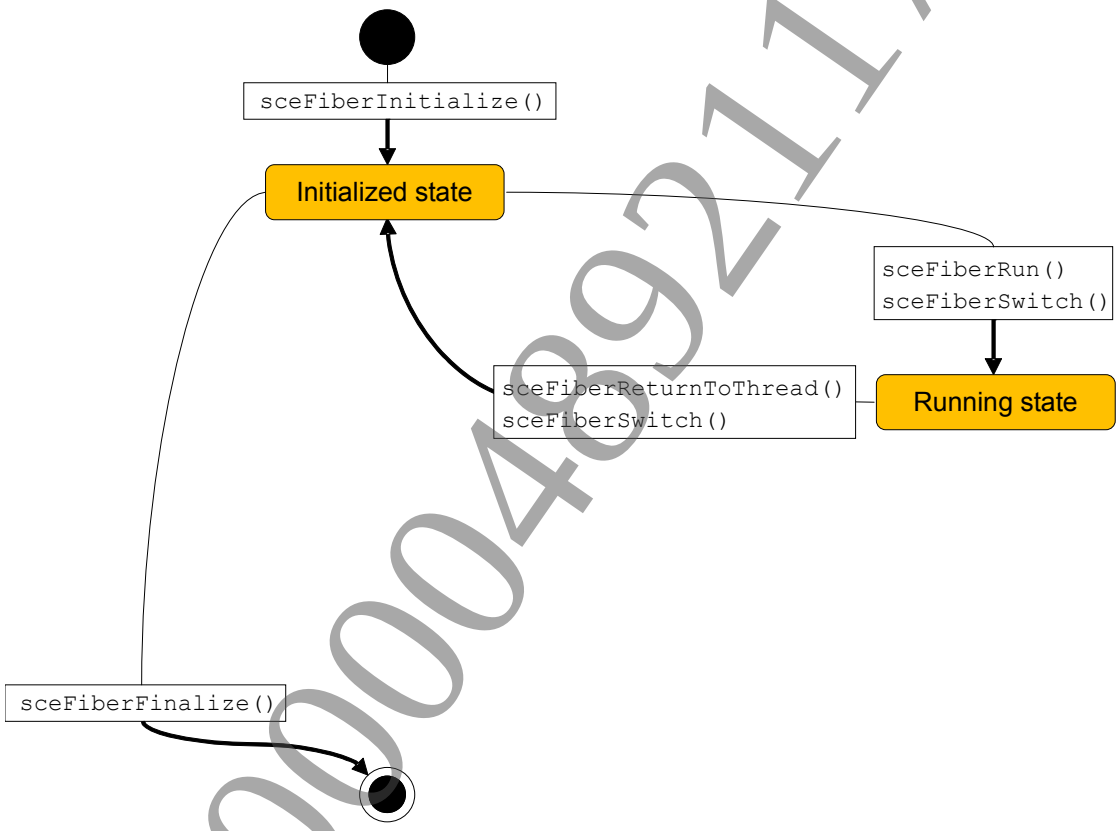


State	Description
Initialized state	This is the initial fiber state. Upon execution, the fiber context holds the required entry function and its arguments, and the initial value of the stack pointer.
Running state	The state when the fiber is being executed.
Suspended state	The execution of a fiber context is being temporarily suspended in this state. A fiber context in this state holds the information required for resuming its execution.

Immediately after being created, a fiber is in the initialized state. Calling `sceFiberRun()` or `sceFiberSwitch()` will change the state of the specified fiber in the initialized state to the running state on a thread which is supplied by the kernel. After that, if the fiber calls `sceFiberSwitch()` or `sceFiberReturnToThread()`, the state of the fiber will be changed to the suspended state. After that, if the fiber is not needed, use `sceFiberFinalize()` to destroy the fiber. If the fiber needs to be executed again, call `sceFiberRun()` or `sceFiberSwitch()` again. The fiber will switch to the running state, and continue from the point at which it was switched to the suspended state. The fiber is independent of the thread supplied by the kernel when the fiber switches to the running state. It can be running state on the other thread.

Normally, application programs specify a context storage area when initializing a fiber. If this area was not specified, the fiber cannot be suspended. If fiber execution is interrupted, the fiber will be returned to the initialized state, without saving the context. If the fiber is then switched to the running state, it is executed from the entry function. The state transitions of a fiber are as follows.

Figure 2 State Transitions of a Fiber with No Context Storage Area



This fiber uses the stack of the thread provided by the kernel. Application programs can mix standard fibers and fibers with no context storage areas.

Dynamic Memory Consumption

All libfiber functions only use the memory area passed to the function in its argument; libfiber functions do not use functions such as `malloc()` internally to dynamically consume memory.

Thread Safe Interfaces

libfiber is designed to be thread safe. libfiber functions can be applied by multiple threads to the same fiber.

3 Using Fibers

This chapter explains the basic procedure for executing a program on fibers. The source code provided in this chapter is taken from the sample program in sample_code/system/api_libfiber/sample_fiber_basic/main.c. When using it for programming, keep in mind that the error handling code has been omitted to simplify the code for the sake of this explanation.

Basic Procedure for Using Fibers

(1) Initialize libfiber

To use libfiber, first load the PRX of libfiber. Make sure to perform this processing before calling any function of libfiber.

```
int ret = sceSysmoduleLoadModule( SCE_SYSMODULE_FIBER );
```

(2) Create a fiber

To create a fiber, it is necessary to have an `SceFiber` structure and memory for context storage of the fiber. Allocate these first on the main memory. The created fiber will be identified by its `SceFiber` structure.

Then, call `sceFiberInitialize()` to create a fiber. As arguments, specify the `SceFiber` structure to be initialized, the fiber's entry function and its arguments, and the starting address and size of its context storage memory.

After being created, the fiber will be in the initialized state.

```
static SceFiber fibers[N_FIBER];
static char context_buffer[N_FIBER][SIZE_CONTEXT]
__attribute__((aligned(SCE_FIBER_CONTEXT_ALIGNMENT)));

ret = sceFiberInitialize(&fibers[i], "myFiber", fiberEntry, i,
(void*)context_buffer[i], SIZE_CONTEXT, NULL);
```

For the final argument of `sceFiberInitialize()`, specify a NULL pointer or a pointer to the `SceFiberOptParam` structure. There are currently no modifiable options available through `SceFiberOptParam` structure use.

(3) Execute fibers

To execute fibers, call `sceFiberRun()`. The fiber specified by the arguments will be executed.

```
ret = sceFiberRun(&fibers[0], 0xffffffff, &cause);
```

This function is completed by the fiber calling `sceFiberReturnToThread()`.

(4) Execute and terminate a fiber

A fiber's execution is started by its entry function. To switch execution with another fiber, call `sceFiberSwitch()`. The fiber will switch. The fiber to be switched must be explicitly specified.

```
ret = sceFiberSwitch(&fibers[(myId+1)%N_FIBER], myId, &newArgOnRun);
```

If another fiber calls `sceFiberSwitch()`, and this fiber is executed again, it will resume from the point at which `sceFiberSwitch()` was called.

To terminate the execution of a fiber, call `sceFiberReturnToThread()`. A fiber's execution is terminated, and `sceFiberRun()` call is completed.

```
ret = sceFiberReturnToThread(0, &newArgOnRun);
```

If `sceFiberRun()` is applied to this fiber again, the fiber will resume from the point at which `sceFiberReturnToThread()` was called.

(5) Confirm fiber status

The information of a specified fiber can be obtained by using `sceFiberGetInfo()`. For details on the information that can be obtained, refer to the `SceFiberInfo` structure. It is also possible to check the remaining amount of memory to be used for saving context for the fibers that were created during the measurement period. Start the measurement period with `sceFiberStartContextSizeCheck()` and stop it with `sceFiberStopContextSizeCheck()`. The measurement period is not during the time when libfiber starts.

In addition, it is possible to obtain the currently running fiber with `sceFiberGetSelf()`.

(6) Fiber Destruction

If a fiber is not needed, use `sceFiberFinalize()` to destroy the fiber. If the specified fiber is in the running state, calling this function will fail. When this function call completes, the memory area for the `SceFiber` structure can be freed.

```
ret = sceFiberFinalize(&fibers[i]);
```

(7) Terminate libfiber

Once you are finished using libfiber, unload the libfiber PRX.

```
int ret = sceSysmoduleUnloadModule( SCE_SYSMODULE_FIBER );
```

Basic APIs

libfiber functions used in the basic procedure for using fibers are as follows.

Fiber Interfaces

Function	Description
`sceFiberInitialize()`	Initializes fiber.
`sceFiberOptParamInitialize()`	Initializes options passed during fiber initialization.
`sceFiberFinalize()`	Destroys fiber.
`sceFiberRun()`	Executes a fiber from thread provided by kernel.
`sceFiberSwitch()`	Switches execution from fiber to a different fiber.
`sceFiberReturnToThread()`	Suspends fiber, and returns to thread provided by kernel.
`sceFiberGetSelf()`	Acquires fiber currently being executed.
`sceFiberGetInfo()`	Gets fiber related information.
`sceFiberRenameSelf()`	Renames fiber.
`sceFiberStartContextSizeCheck()`	Starts the measurement period for the volume of memory for the context of the fiber
`sceFiberStopContextSizeCheck()`	Stops the measurement period for the volume of memory for the context of the fiber
`sceFiberPushUserMarkerWithHud()`	Pushes a marker with support for the Razor HUD
`sceFiberPopUserMarkerWithHud()`	Pops a marker with support for the Razor HUD