进程监控SDK接口文档

|  |  |
| --- | --- |
| 版本号 | Ver\_1.2.0 |
| 制定 | 江民Linux研发组 |
| 日期 | 2018.2.27 |

SDK组成：

proc\_monitor.h

libproc\_monitor.so

1. 监控初始化
2. 回调函数

typedef enum {

PROC\_SNAPSHOT,

PROC\_ACTION\_CREATE,

PROC\_ACTION\_DESTROY,

}proc\_event\_type\_t;

typedef void (\*proc\_monitor\_callback\_t)(proc\_event\_type\_t event\_type, size\_t nmemb, Proc \*\*procs, void \*p\_user\_data);

说明：

监控过程中的回调函数。

注意，不要在本函数中有耗时的操作；同时，回传的数据会在函数结束后失效，如需保存数据，请自行拷贝。

参数：

event\_type：本次事件的标识

PROC\_SNAPSHOT 一次进程快照

PROC\_ACTION\_CREATE 进程行为：创建

PROC\_ACTION\_DESTROY 进程行为：销毁

p\_user\_data：定义回调函数的用户所使用的数据。

nmemb: 指针数组procs的数组长度，即进程的个数

procs: 一个指针数组。其中每个元素为Proc \*。Proc是一个关于进程信息的结构体，定义请参看proc.h头文件，简介如下：

struct Proc {

public:

\_\_attribute\_\_ ((visibility ("default"))) Proc(); //

ino\_t exec\_ino; // /proc/<pid>/exe -> exec file inode num

uint64\_t pid; // pid

int comparing\_state; // addtional : compare(time\_old, time\_now) state is[1:new\_proc, 2:destroy\_proc, 3:running\_proc]

time\_t starttime; // addtional : timestamp(s) of proc start

time\_t stoptime; // addtional : timestamp(s) of proc destroy

uint64\_t current\_time; // addtional : timestamp(ms) of read this proc

std::string name; // filename

std::string abs\_name; // path/name

std::string cmdline; // shell command line

…… // 省略若干个元素

};

返回：

暂定void，将来可能会根据是否拦截改为bool类型（比如，返回true放行，false拦截）

1. 初始化接口

typdedef proc\_monitor proc\_monitor\_t;

proc\_monitor\_t \*init\_proc\_monitor(proc\_monitor\_callback\_t callback, void \*p\_user\_data, int32\_t option\_count, const char \*(\*pp\_options)[2]);

参数：

callback：发生事件时的回调函数。

p\_user\_data：回调函数的p\_user\_data参数。

option\_count: 监控选项的个数

pp\_options: 是一个二维数组，表示的监控选项。pp\_options有option\_count个元素，每个元素均由一对const char \*组成key-value的形式。其中，(\*pp\_option)[0]为key，(\*pp\_option)[1]为value。具体的内容参考[设置参数](#设置参数)一节。

返回：

成功初始化返回非空指针，否则返回NULL。

说明：

初始化进程监控

1. 设置参数

bool set\_option(proc\_monitor\_t \*p\_monitor, const char \*p\_option\_name, const char \*p\_option\_value);

说明：

设置option，或改变option。SDK内部会依据option的名称，自动将p\_option\_value转换到实际的类型。

参数：

p\_option\_name:option的名称。

p\_option\_value:option的值。当实际的类型不是char \*时，需要强制类型转换为char \*。

返回：

设置成功返回true，否则返回false。

可设置参数：

|  |  |  |
| --- | --- | --- |
| 选项名称 | 参数类型 | 默认值 |
| snapshot\_delta  快照的刷新间隔，单位秒 | int32\_t | 3 |

1. 查询参数

const char\* get\_option(proc\_monitor\_t \*p\_monitor, const char \*p\_option\_name);

说明：

返回的option的值。

参数：

p\_option\_name:要查询的option名称。可选值参考[设置参数](#设置参数)。

返回：

返回option对应的值。当实际的类型不是char \*时，需要用户做类型转换。

出错则返回NULL。

1. 停止监控

void fin\_proc\_monitor(proc\_monitor\_t \*p\_proc\_monitor);

当程序退出时，调用本接口终止对进程的监控，并释放相关资源。

1. 使用实例

#include <stdio.h>

#include <inttypes.h>

#include "proc\_monitor.h"

void proc\_call\_back(proc\_event\_type\_t event\_type,

size\_t nmemb,

Proc \*\*procs,

void \*p\_user\_data){

const char \*event;

if(event\_type == PROC\_SNAPSHOT)

event = "PROC\_SNAPSHOT";

else if(event\_type == PROC\_ACTION\_CREATE)

event = "PROC\_ACTION\_CREATE";

else if(event\_type == PROC\_ACTION\_DESTROY)

event = "PROC\_ACTION\_DESTROY";

else

event = "UNKNOWN";

printf("event:%s\n", event);

for(size\_t i=0; i<nmemb; i++) {

printf( "\tpid:%" PRIu64 "\n"

"\tppid:%" PRIu64 "\n"

"\tpath:%s\n"

"\tstarttime:%lu\n"

"\tendtime:%lu\n"

,

procs[i]->pid,

procs[i]->ppid,

procs[i]->abs\_name.data(),

procs[i]->starttime,

procs[i]->stoptime

);

putchar(10);

}

putchar(10);

putchar(10);

putchar(10);

}

int32\_t main(int32\_t argc,char \*args[])

{

size\_t snapshot\_delta = 3;

const char \*options[1][2] = { {"snapshot\_delta", (const char \*)&snapshot\_delta } };

proc\_monitor\_t \*handle = init\_proc\_monitor(proc\_call\_back, NULL, sizeof(options)/sizeof(options[0]), options);

snapshot\_delta = 5;

set\_option(handle, "snapshot\_delta", (const char \*)&snapshot\_delta);

printf("exit with %c\n", getchar());

fin\_proc\_monitor(handle);

}

1. 部署

// TODO

描述如何安装。

依赖GLIBC\_2.2.5、CXXABI\_1.3.10。