

在内核开发中，经常需要进行进程和句柄之间的互相转换。进程通常由一个唯一的进程标识符（PID）来标识，而句柄是指对内核对象的引用。在Windows内核中，`EPROCESS` 结构表示一个进程，而HANDLE 是一个句柄。

为了实现进程与句柄之间的转换，我们需要使用一些内核函数。对于进程PID和句柄的互相转换，可以使用函数如 `OpenProcess` 和 `GetProcessId`。`OpenProcess`函数接受一个PID作为参数，并返回一个句柄。`GetProcessId`函数接受一个句柄作为参数，并返回该进程的PID。

对于进程PID和 `EPROCESS` 结构的互相转换，可以使用函数如 `PsGetProcessId` 和 `PsGetCurrentProcess`。`PsGetProcessId`函数接受一个 `EPROCESS` 结构作为参数，并返回该进程的PID。`PsGetCurrentProcess` 函数返回当前进程的 `EPROCESS` 结构。

最后，对于句柄和 `EPROCESS` 结构的互相转换，可以使用函数如 `ObReferenceObjectByHandle`和 `PsGetProcessId`。`ObReferenceObjectByHandle`函数接受一个句柄和一个对象类型作为参数，并返回对该对象的引用。`PsGetProcessId` 函数接受一个 `EPROCESS`结构作为参数，并返回该进程的PID。

掌握这些内核函数的使用，可以方便地实现进程与句柄之间的互相转换。在进行进程和线程的内核开发之前，了解这些转换功能是非常重要的。

进程PID与进程HANDLE之间的互相转换: 进程 PID 转化为 HANDLE 句柄，可通过 `ZwOpenProcess` 这个内核函数，传入 PID 传出进程 HANDLE 句柄，如果需要将 HANDLE 句柄转化为 PID 则可通过 `ZwQueryInformationProcess` 这个内核函数来实现，具体转换实现方法如下所示；

在内核开发中，经常需要进行进程 PID 和句柄 HANDLE 之间的互相转换。将进程 PID 转化为句柄 HANDLE 的方法是通过调用 `ZwOpenProcess` 内核函数，传入PID作为参数，函数返回对应进程的句柄HANDLE。具体实现方法是，定义一个 `OBJECT_ATTRIBUTES` 结构体和 `CLIENT_ID` 结构体，将进程PID赋值给 `CLIENT_ID` 结构体的 `UniqueProcess` 字段，调用 `ZwOpenProcess` 函数打开进程，如果函数执行成功，将返回进程句柄HANDLE，否则返回NULL。

将句柄 HANDLE 转化为进程 PID 的方法是通过调用 `ZwQueryInformationProcess` 内核函数，传入进程句柄和信息类别作为参数，函数返回有关指定进程的信息，包括进程PID。具体实现方法是，定义一个 `PROCESS_BASIC_INFORMATION` 结构体和一个 `NTSTATUS` 变量，调用 `ZwQueryInformationProcess` 函数查询进程基本信息，如果函数执行成功，将返回进程PID，否则返回0。

其中 `ZwQueryInformationProcess` 是一个未被导出的函数如需使用要通过 `MmGetSystemRoutineAddress` 动态获取到，该函数的原型定义如下：

```
NTSTATUS ZwQueryInformationProcess(
    HANDLE          ProcessHandle,
    PROCESSINFOCLASS ProcessInformationClass,
    PVOID           ProcessInformation,
    ULONG           ProcessInformationLength,
    PULONG           ReturnLength
);
```

函数可以接受一个进程句柄 `ProcessHandle`、一个 `PROCESSINFOCLASS` 枚举类型的参数 `ProcessInformationClass`、一个用于存储返回信息的缓冲区 `ProcessInformation`、缓冲区大小 `ProcessInformationLength` 和一个指向ULONG类型变量的指针 `ReturnLength` 作为参数。

在调用该函数时，`ProcessInformationClass` 参数指定要获取的进程信息的类型。例如，如果要获取进程的基本信息，则需要将该参数设置为 `ProcessBasicInformation`；如果要获取进程的映像文件名，则需要将该参数设置为 `ProcessImageFileName`。调用成功后，返回的信息存储在 `ProcessInformation` 缓冲区中。

在调用该函数时，如果 `ProcessInformation` 缓冲区的大小小于需要返回的信息大小，则该函数将返回 `STATUS_INFO_LENGTH_MISMATCH` 错误代码，并将所需信息的大小存储在 `ReturnLength` 指针指向的 `ULONG` 类型变量中。

`ZwQueryInformationProcess` 函数的返回值为 `NTSTATUS` 类型，表示函数执行的结果状态。如果函数执行成功，则返回 `STATUS_SUCCESS`，否则返回其他错误代码。

掌握这些转换方法可以方便地在内核开发中进行进程PID和句柄HANDLE之间的互相转换。

```
#include <ntifs.h>

// 定义函数指针
typedef NTSTATUS(*PfnZwQueryInformationProcess)(
    __in HANDLE ProcessHandle,
    __in PROCESSINFOCLASS ProcessInformationClass,
    __out_bcount(ProcessInformationLength) PVOID ProcessInformation,
    __in ULONG ProcessInformationLength,
    __out_opt PULONG ReturnLength
);

PfnZwQueryInformationProcess ZwQueryInformationProcess;

// 传入PID传出HANDLE句柄
HANDLE PidToHandle(ULONG PID)
{
    HANDLE hProcessHandle;
    OBJECT_ATTRIBUTES obj;
    CLIENT_ID clientid;

    clientid.UniqueProcess = PID;
    clientid.UniqueThread = 0;

    // 属性初始化
    InitializeObjectAttributes(&obj, 0, OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
    0, 0);

    NTSTATUS status = ZwOpenProcess(&hProcessHandle, PROCESS_ALL_ACCESS, &obj,
    &clientid);
    if (status == STATUS_SUCCESS)
    {
        // DbgPrint("[*] 已打开 \n");
        ZwClose(&hProcessHandle);
        return hProcessHandle;
    }

    return 0;
}

// HANDLE句柄转换为PID
ULONG HandleToPid(HANDLE handle)
{
    PROCESS_BASIC_INFORMATION ProcessBasicInfor;

    // 初始化字符串，并获取动态地址
```

```

    UNICODE_STRING UtrZwQueryInformationProcessName =
RTL_CONSTANT_STRING(L"ZwQueryInformationProcess");
    ZwQueryInformationProcess =
(PfnZwQueryInformationProcess)MmGetSystemRoutineAddress(&UtrZwQueryInformationPro
cessName);

    // 调用查询
    ZwQueryInformationProcess(
        handle,
        ProcessBasicInformation,
        (PVOID)&ProcessBasicInfor,
        sizeof(ProcessBasicInfor),
        NULL);

    // 返回进程PID
    return ProcessBasicInfor.UniqueProcessId;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("[ -] 驱动卸载 \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark \n");

    // 将PID转换为HANDLE
    HANDLE ptr = PidToHandle(6932);
    DbgPrint("[*] PID --> HANDLE = %p \n", ptr);

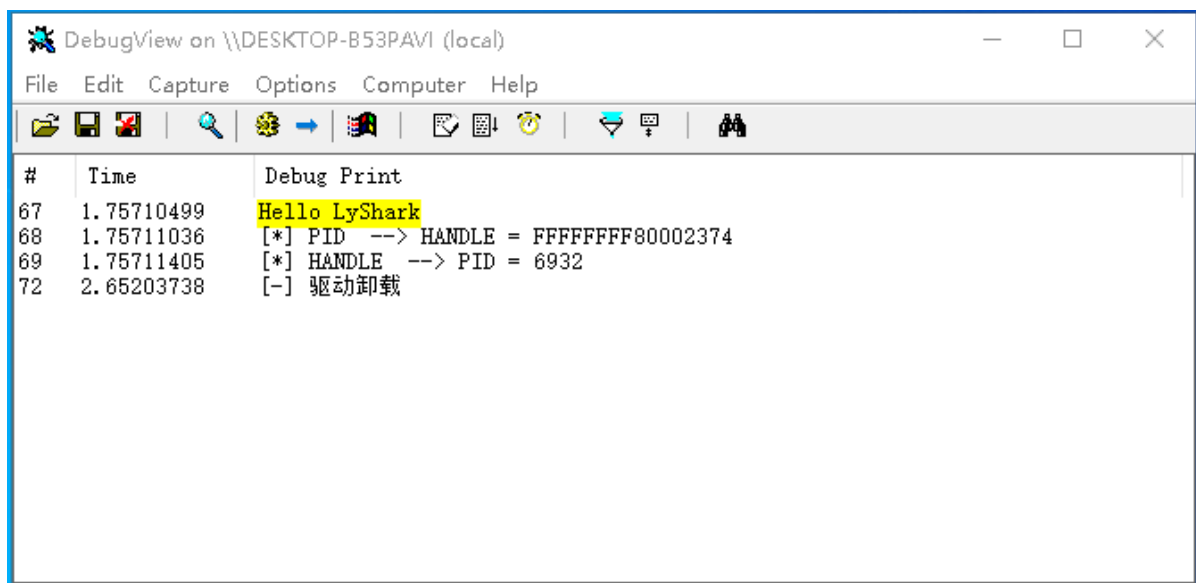
    // 句柄转为PID
    ULONG pid = HandleToPid(ptr);

    DbgPrint("[*] HANDLE --> PID = %d \n", pid);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

编译并运行如上这段代码片段，将把进程PID转为HANDLE句柄，再通过句柄将其转为PID，输出效果图如下所示；



进程PID转换为EProcess结构: 通过 `PsLookupProcessByProcessId` 函数, 该函数传入一个 `PID` 则可获取到该PID的 `EProcess` 结构体, 具体转换实现方法如下所示;

本段代码展示了如何使用Windows内核API函数 `PsLookupProcessByProcessId` 将一个PID (Process ID) 转换为对应的 `EProcess` 结构体, `EProcess`是Windows内核中描述进程的数据结构之一。

代码段中定义了一个名为 `PidToObject` 的函数, 该函数的输入参数是一个 `PID`, 输出参数是对应的 `EProcess` 结构体。

在函数中, 通过调用 `PsLookupProcessByProcessId` 函数来获取对应PID的 `EProcess` 结构体, 如果获取成功, 则调用 `ObDereferenceObject` 函数来减少 `EProcess` 对象的引用计数, 并返回获取到的 `EProcess` 指针; 否则返回0。

在 `DriverEntry` 函数中, 调用了 `PidToObject` 函数将PID 6932转换为对应的 `EProcess` 结构体, 并使用 `DbgPrint` 函数输出了转换结果。最后设置了驱动程序卸载函数为 `UnDriver`, 当驱动程序被卸载时, `UnDriver` 函数会被调用。

```
#include <ntifs.h>
#include <windef.h>

// 将Pid转换为Object or EProcess
PEPROCESS PidToObject(ULONG Pid)
{
    PEPROCESS pEprocess;

    NTSTATUS status = PsLookupProcessByProcessId((HANDLE)Pid, &pEprocess);

    if (status == STATUS_SUCCESS)
    {
        ObDereferenceObject(pEprocess);
        return pEprocess;
    }

    return 0;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("[-] 驱动卸载 \n");
}
```

```

}

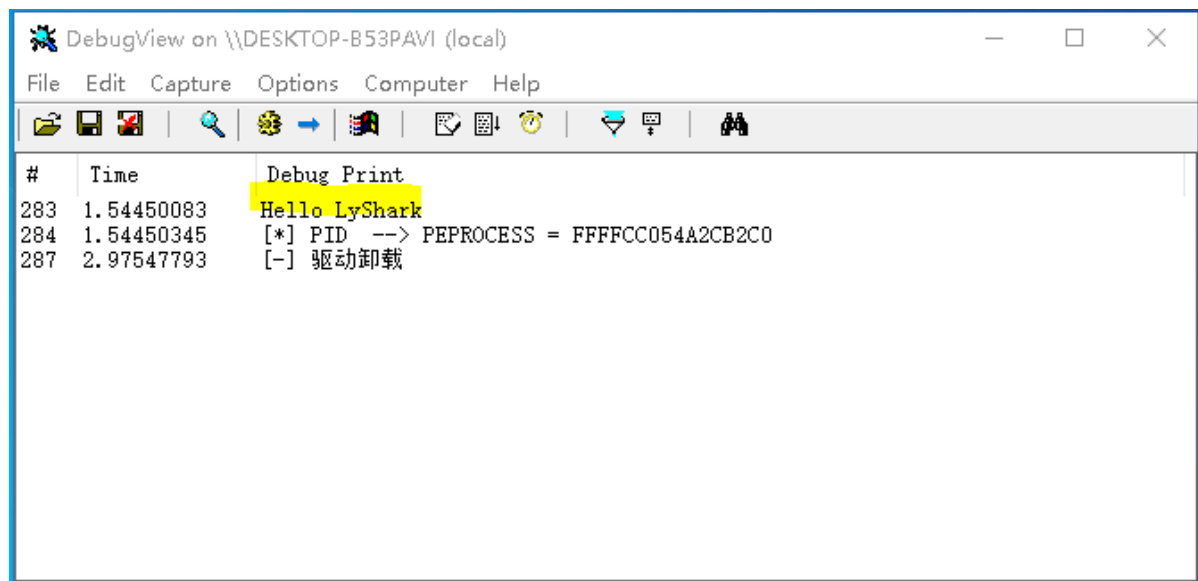
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark \n");

    // 将PID转换为PEPROCESS
    PEPROCESS ptr = PidToObject(6932);
    DbgPrint("[*] PID --> PEPROCESS = %p \n", ptr);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

编译并运行如上这段代码片段，将把进程PID转为EProcess结构，输出效果图如下所示；



进程HANDLE与EPROCESS互相转换: 将 Handle 转换为 EProcess 结构可使用内核函数

ObReferenceObjectByHandle 实现，反过来 EProcess 转换为 Handle 句柄可使用

ObOpenObjectByPointer 内核函数实现，具体转换实现方法如下所示；

首先，将 Handle 转换为 EProcess 结构体，可以使用 ObReferenceObjectByHandle 内核函数。该函数接受一个 Handle 参数，以及对应的对象类型（这里为 EProcess），并返回对应对象的指针。此函数会对返回的对象增加引用计数，因此在使用完毕后，需要使用 ObDereferenceObject 将引用计数减少。

其次，将 EProcess 结构体转换为 Handle 句柄，可以使用 ObOpenObjectByPointer 内核函数。该函数接受一个指向对象的指针（这里为 EProcess 结构体的指针），以及所需的访问权限和对象类型，并返回对应的 Handle 句柄。此函数会将返回的句柄添加到当前进程的句柄表中，因此在使用完毕后，需要使用 CloseHandle 函数将句柄关闭，以避免资源泄漏。

综上所述，我们可以通过这两个内核函数实现 Handle 和 EProcess 之间的相互转换，转换代码如下所示；

```

#include <ntifs.h>
#include <windef.h>

// 传入PID传出HANDLE句柄
HANDLE PidToHandle(ULONG PID)
{
    HANDLE hProcessHandle;

```

```

OBJECT_ATTRIBUTES obj;
CLIENT_ID clientid;

clientid.UniqueProcess = PID;
clientid.UniqueThread = 0;

// 属性初始化
InitializeObjectAttributes(&obj, 0, OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
0, 0);

NTSTATUS status = ZwOpenProcess(&hProcessHandle, PROCESS_ALL_ACCESS, &obj,
&clientid);
if (status == STATUS_SUCCESS)
{
    // DbgPrint("[*] 已打开 \n");
    ZwClose(&hProcessHandle);
    return hProcessHandle;
}

return 0;
}

// 将Handle转换为EProcess结构
PEPROCESS HandleToEprocess(HANDLE handle)
{
    PEPROCESS pEprocess;

    NTSTATUS status = ObReferenceObjectByHandle(handle, GENERIC_ALL,
*PsProcessType, KernelMode, &pEprocess, NULL);
    if (status == STATUS_SUCCESS)
    {
        return pEprocess;
    }

    return 0;
}

// EProcess转换为Handle句柄
HANDLE EprocessToHandle(PEPROCESS eprocess)
{
    HANDLE hProcessHandle = (HANDLE)-1;

    NTSTATUS status = ObOpenObjectByPointer(
        eprocess,
        OBJ_KERNEL_HANDLE,
        0,
        0,
        *PsProcessType,
        KernelMode,
        &hProcessHandle
    );

    if (status == STATUS_SUCCESS)
    {
        return hProcessHandle;
    }
}

```

```

    return 0;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("[ - ] 驱动卸载 \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark \n");

    // 将Handle转换为EProcess结构
    EPROCESS eprocess = HandleToEprocess(PidToHandle(6932));
    DbgPrint("[*] HANDLE --> EProcess = %p \n", eprocess);

    // 将EProcess结构转换为Handle
    HANDLE handle = EprocessToHandle(eprocess);
    DbgPrint("[*] EProcess --> HANDLE = %p \n", handle);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

编译并运行如上这段代码片段，将把进程 HANDLE 与 EProcess 结构互转，输出效果图如下所示；

#	Time	Debug Print
85	3.15353322	Hello LyShark
86	3.15353966	[*] HANDLE --> EProcess = FFFFCC054A2CB2C0
87	3.15354109	[*] EProcess --> HANDLE = FFFFFFFF800026EC
93	5.50794983	[-] 驱动卸载