本章主要介绍驱动开发中的一些基础知识，包括内核链表与结构体、内核自旋锁、字符串转换方法和驱动对象介绍等。在驱动开发中，内核链表和结构体是常用的数据结构，本章将介绍如何使用内核链表和结构体来管理和操作数据。内核自旋锁是一种保护共享资源的机制，本章将介绍如何使用内核自旋锁来保护驱动程序中的共享资源。

字符串转换方法是驱动开发中必不可少的一部分，本章将介绍如何使用内核提供的字符串转换函数来处理字符串操作。最后，本章还将介绍驱动对象的概念和使用方法，包括设备对象、文件对象、IO 请求包等。

通过学习本章的内容，读者将了解驱动开发中的一些基础知识和常用技术，可以更好地理解驱动程序的开发和调试过程。同时，本章所涉及的内容也是深入学习驱动开发的基础，为后续章节的内容打下了坚实的基础。

在`Windows`内核中，为了实现高效的数据结构操作，通常会使用链表和结构体相结合的方式进行数据存储和操作。内核提供了一个专门用于链表操作的数据结构`LIST_ENTRY`，可以用来描述一个链表中的每一个节点。

使用链表来存储结构体时，需要在结构体中嵌入一个`LIST_ENTRY`类型的成员变量，用来连接相邻的节点。通过一些列链表操作函数，如`InitializeListHead、InsertHeadList、InsertTailList、RemoveEntryList`等，可以对链表中的结构体进行插入、删除、遍历等操作。

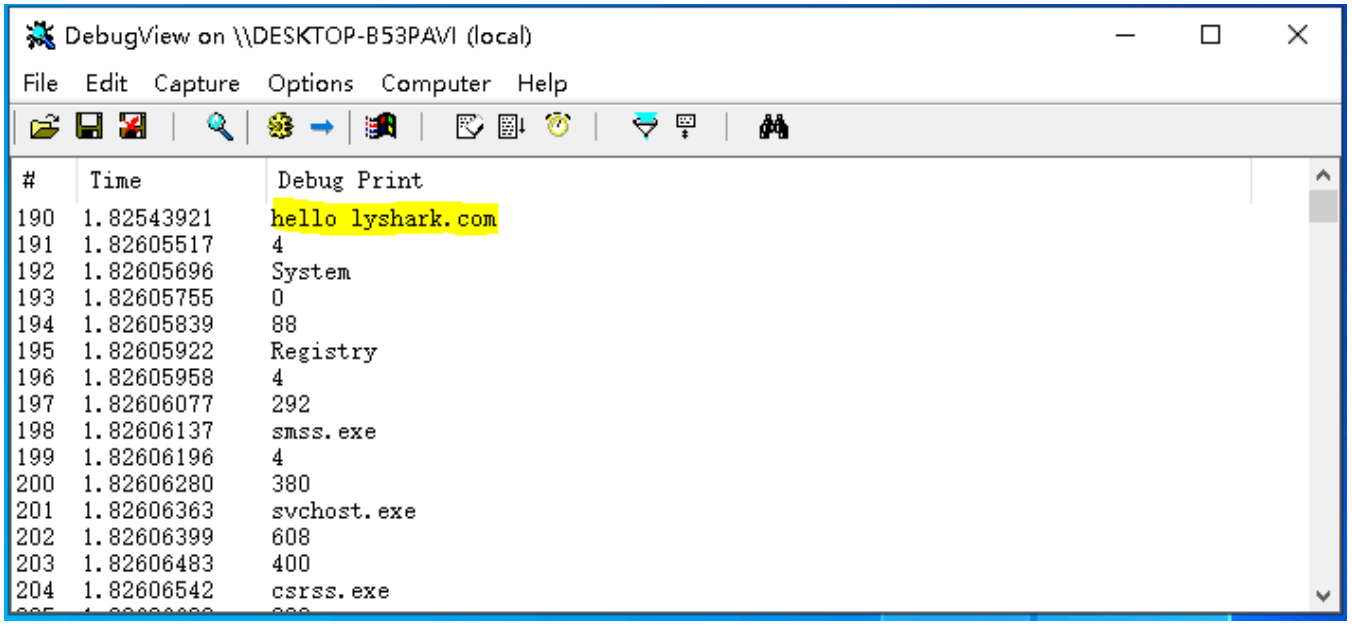
下面是一个简单的实现，用于枚举所有用户进程，并将进程信息存储到链表结构体中：

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

extern PVOID PsGetProcessPeb(_In_ PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
extern NTKERNELAPI PVOID PsGetProcessWow64Process(_In_ PEPROCESS Process);
extern NTKERNELAPI UCHAR* PsGetProcessImageFileName(IN PEPROCESS Process);
extern NTKERNELAPI HANDLE PsGetProcessInheritedFromUniqueProcessId(IN PEPROCESS Process);

typedef struct
{
    DWORD Pid;
    UCHAR ProcessName[2048];
    DWORD Handle;
    LIST_ENTRY ListEntry;
}ProcessList;

// 根据进程ID返回进程EPROCESS结构体失败返回NULL
PEPROCESS LookupProcess(HANDLE Pid)
{
    PEPROCESS eprocess = NULL;
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    Status = PsLookupProcessByProcessId(Pid, &eprocess);
    if (NT_SUCCESS(Status))
    {
        return eprocess;
    }
    return NULL;
}

// 内核链表操作
BOOLEAN GetAllProcess()
{
    PEPROCESS eproc = NULL;
```

```
    LIST_ENTRY linkListHead;

    // 初始化链表头部
    InitializeListHead(&linkListHead);
    ProcessList *pData = NULL;

    for (int temp = 0; temp < 100000; temp += 4)
    {
        eproc = LookupProcess((HANDLE)temp);
        if (eproc != NULL)
        {
            STRING nowProcessnameString = { 0 };
            RtlInitString(&nowProcessnameString, PsGetProcessImageFileName(eproc));

            // DbgPrint("进程名: %s --> 进程PID = %d --> 父进程PPID = %d\r\n",
            // PsGetProcessImageFileName(eproc), PsGetProcessId(eproc),
PsGetProcessInheritedFromUniqueProcessId(eproc));

            // 分配内核堆空间
            pData = (ProcessList *)ExAllocatePool(PagedPool, sizeof(ProcessList));
            RtlZeroMemory(pData, sizeof(ProcessList));

            // 设置变量
            pData->Pid = (DWORD)PsGetProcessId(eproc);
            RtlCopyMemory(pData->ProcessName, PsGetProcessImageFileName(eproc),
strlen(PsGetProcessImageFileName(eproc)) * 2);
            pData->Handle = (DWORD)PsGetProcessInheritedFromUniqueProcessId(eproc);

            // 插入元素到
            InsertTailList(&linkListHead, &pData->ListEntry);
            ObDereferenceObject(eproc);
        }
    }

    // 输出链表内的数据
    while (!IsListEmpty(&linkListHead))
    {
        LIST_ENTRY *pEntry = RemoveHeadList(&linkListHead);
        pData = CONTAINING_RECORD(pEntry, ProcessList, ListEntry);

        DbgPrint("%d \n", pData->Pid);
        DbgPrint("%s \n", pData->ProcessName);
        DbgPrint("%d \n", pData->Handle);
        ExFreePool(pData);
    }
    return TRUE;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}
```

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark \n");

    GetAllProcess();

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

DbgView是一款用于监视内核和应用程序调试输出的工具，可以输出各种调试信息和日志信息，包括`OutputDebugString`函数输出的信息。当我们在内核中调用`OutputDebugString`函数输出信息时，可以通过`DbgView`查看输出结果，我们手动上述代码后将可以在`DbgView`中看到输出的进程信息，如下图所示；



如果需要在内核模式中返回一个结构体，可以通过定义一个结构体指针作为函数参数，将结构体指针作为函数返回值来实现。返回结构体，则可以这样来写代码。

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

typedef struct
{
    int count;
    char username[256];
    char password[256];
}MyData;

// 模拟返回一个结构
BOOLEAN GetProcess(PVOID OutPut)
{
    RtlZeroMemory(OutPut, sizeof(MyData));
    MyData *data = OutPut;

    data->count = 100;
    RtlCopyMemory(data->username, "lyshark.com", sizeof("lyshark.com"));
```

```
    RtlCopyMemory(data->password, "https://www.cnblogs.com/lyshark",
sizeof("https://www.cnblogs.com/lyshark"));
    return TRUE;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark \n");
    PVOID Ptr = (PVOID)ExAllocatePool(NonPagedPool, sizeof(MyData));


    GetProcess(Ptr);

    MyData *data = (MyData *)Ptr;

    DbgPrint("count = %d \n", data->count);
    DbgPrint("username = %s \n", data->username);
    DbgPrint("password = %s \n", data->password);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

输出效果如下图所示;