SO2 Cheat Sheet

Compilarea nucleului

`make menuconfig` – configurarea nucleului Linux
`make` – compilarea nucleului Linux
`make modules_install` – instalează modulele
`make install` – instalează imagina de kernel
`build` – compilarea nucleului Windows
`/boot/grub/menu.lst` – fișierul de configurare a GRUB
`C:\boot.ini` – configurare booting pe Windows

Aplicații utile în kernel programming

`GCC, GNU Make, GDB` – suita standard pe Linux
`Windows Driver Kit (WDK)` – suita de driver development pe Windows
`Windows Research Kernel (WRK)` – o parte din sursele nucleului Windows
`WinDbg, LiveKd` – utilitare de debugging în Windows
`Linux Cross Reference (LXR)` – interfață web pentru source browsing

Crearea unui modul simplu în Linux

`kernel.h, init.h, module.h` – headere necesare
`MODULE_[DESCRIPTION|AUTHOR|LICENSE]` – informații modul
`static int init_func(void)` – entry point
`static void exit_func(void)` – exit point
`module_init, module_exit` – specificare funcții entry / exit point

Crearea unui modul simplu în Windows

`ntddk.h` – header necesar
`NTSTATUS DriverEntry(PDRIVER_OBJECT driver, PUNICODE_STRING registry)` – entry point
`driver->DriverUnload` – înregistrare exit point

Toolchains

`make, kbuild` – compilare module Linux
`lsmod, insmod, rmmod` – lucrul cu module în Linux
`printk, addr2line, objdump, netconsole` – debugging Linux
`nmake, sources` – compilare module Windows
`driver [list, load, unload]` – lucrul cu module în Windows
`WinDbg, !analyze -v` – debugging Windows

Alocare memorie

Linux
`void *kmalloc(size_t size, int flags);` – alocare memorie
`void *kzalloc(size_t size, int flags);` – alocare memorie inițializată la zero
`void kfree(const void *mem);` – eliberare memorie

Windows
```
PVOID ExAllocatePoolWithTag(
        IN POOL_TYPE PoolType,
        IN SIZE_T NumberOfBytes,
        IN ULONG Tag); – alocare memorie
VOID ExFreePoolWithTag(
        IN PVOID P,
        IN ULONG Tag); – eliberare memorie
```

Liste

Linux
`struct list_head` – listă
`list_add(struct list_head *new, struct list_head *head)` – inserează new după head
`list_del(struct list_head *entry)` – șterge un element din listă
`list_empty(struct list_head *head)` – verifică dacă o listă este goală
`list_entry(ptr, type, member)` – întoarce structura ce conține list_head-ul
`list_for_each(pos, head)` – parcurge o listă
`list_for_each_safe(pos, head)` – parcurge o listă (safe pentru ștergerea unui element)
Windows
`SINGLE_LIST_ENTRY` – listă
`LIST_ENTRY` – listă dublu înlănțuită
```
VOID PushEntryList(
        IN PSINGLE_LIST_ENTRY ListHead
        IN PSINGLE_LIST_ENTRY Entry);
        – inserează Entry
VOID PopEntryList(
        IN PSINGLE_LIST_ENTRY ListHead);
        – șterge primul element din listă
```

Locking

Linux
`spinlock_t` – tip spinlock
`spin_lock_init(spinlock_t *lock);` – inițializare spinlock
`spin_lock(spinlock_t *lock);` – obținere lock
`spin_unlock(spinlock_t *lock);` – eliberare lock
`struct semaphore` – tip semafor
`void sema_init(struct semaphore *sem, int val);` – inițializare semafor la valoarea val
`void down(struct semaphore *sem);` – decrementare semafor
`void up(struct semaphore *sem);` – incrementare semafor
`atomic_t` – tip atomic
```
atomic_set(atomic_t *v, int i)
int atomic_read(atomic_t *v);
void atomic_add(int i, atomic_t *v);
void atomic_sub(int i, atomic_t *v);
void atomic_inc(atomic_t *v);
void atomic_dec(atomic_t *v);
int atomic_inc_and_test(atomic_t *v);
int atomic_dec_and_test(atomic_t *v);
```
Windows
`KSPIN_LOCK` – tip spinlock
```
VOID KeInitializeSpinLock(
       IN PKSPIN_LOCK SpinLock); – inițializare spinlock
VOID KeAcquireSpinLock(
       IN PKSPIN_LOCK SpinLock,
       OUT PKIRQL OldIrql); – obținere lock
VOID KeReleaseSpinLock(
       IN PKSPIN_LOCK SpinLock,
       IN KIRQL NewIrql); – eliberare lock
```
`KESEMAPHORE` – tip semafor
```
VOID KeInitializeSemaphore(
       IN PRKSEMAPHORE Semaphore,
       IN LONG Count,
       IN LONG Limit); – inițializare semafor
```
`NSTATUS KeWaitForSingleObject(...)` – decrementare semafor
`LONG KeReleaseSemaphore(...)` – incrementare semafor
```
InterlockedCompareExchange(...)
InterlockedDecrement(...)
InterlockedExchange(...)
InterlockedExchangeAdd(...)
InterlockedIncrement(...)
```

Device drivere în Linux

```
mknod <fișier> <c/b> <major> <minor>
int register_chrdev_region(dev_t first, unsigned int count,
char *name)
void unregister_chrdev_region(dev_t first, unsigned int
count)
void cdev_init(struct cdev *cdev, struct file_operations
*fops)
int cdev_add(struct cdev *dev, dev_t num, unsigned int
count)
void cdev_del(struct cdev *dev)
int (*open) (struct inode *, struct file *)
ssize_t (*read) (struct file *, char __user *, size_t,
loff_t *);
ssize_t (*write) (struct file *, const char __user *,
size_t, loff_t *);
int (*ioctl) (struct inode *, struct file *, unsigned int,
unsigned long);
int (*release) (struct inode *, struct file *);
```

Cozi de așteptare – waitqueues

```
DECLARE_WAIT_QUEUE_HEAD(wait queue_head_t *q);
void init_waitqueue_head(wait_queue_head_t *q);
int wait_event(wait_queue_head_t *q, int condition);
int wait_event_interruptible(wait_queue_head_t *q, int
condition);
int wait_event_timeout(wait_queue_head_t *q, int condition,
int timeout);
```

```
int wait_event_interruptible_timeout(wait_queue_head_t *q,
int condition, int timeout);
void wake_up(wait_queue_head_t *q);
void wake_up_interruptible(wait_queue_head_t *q);
```

Userspace access în Linux

```
put_user(type val, type *address);
get_user(type val, type *address);
unsigned long copy_to_user(void __user *to, const void
*from, unsigned long n);
unsigned long copy_from_user(void *to, const void __user
*from, unsigned long n);
```

Device drivere în Windows

`DRIVER_OBJECT` – obiect aferent unui driver
`DEVICE_OBJECT` – obiect aferent unui dispozitiv
`IRP` – I/O Request Pachet – împachetarea unei cereri I/O
`IO_STACK_LOCATION` – poziția curentă din IRP în stiva de drivere
`\Device\MyDevice` – nume de dispozitiv în kernel space
`\??\MyDevice` – symbolic link la nume
`\\.\MyDevice` – nume de dispozitiv în user space

```
NTSTATUS IoCreateDevice(
       PDRIVER_OBJECT DriverObject,
       ULONG DeviceExtensionSize,
       PUNICODE_STRING DeviceName,
       DEVICE_TYPE DeviceType,
       LONG DeviceCharacteristics,
       BOOLEAN Exclusive,
       PDEVICE_OBJECT *DeviceObject);
```
– crearea unui obiect dispozitiv

```
NTSTATUS IoCreateSymbolicLink(
       PUNICODE_STRING SymbolicLinkName,
       PUNICODE_STRING DeviceName);
```
– crearea unui link simbolic (nume vizibil din user space)

```
NTSTATUS IoDeleteSymbolicLink(
       PUNICODE_STRING SymbolicLinkName);
```
– ștergerea unui link simbolic

```
NTSTATUS IoDeleteDevice(
       PDEVICE_OBJECT DeviceObject);
```
– ștergerea unui obiect dispozitiv

`device->DeviceExtension` – zonă alocată pentru structura proprie a unui dispozitiv
`driver->MajorFunction[]` – vector pentru rutinele de dispatch expuse de driver

```
NTSTATUS (*PDRIVER_DISPATCH)(
       PDEVICE_OBJECT DeviceObject,
       PIRP Irp);
```
– rutină de dispatch pentru drivere

Accesarea spațiului de adresă din user space

```
PVOID MmGetSystemAddressForMdlSafe(
       PMDL Mdl,
       MM_PAGE_PRIORITY Priority);
```
– obține o adresă non-paged aferentă bufferului descris de MDL

```
VOID ProbeForRead/ProbeForWrite(
       CONST VOID *Address,
       SIZE_T Length,
       ULONG Alignment);
```
– verifică un buffer din user-space

Evenimente – events

`KEVENT` – structură aferentă unui eveniment (event)

```
VOID KeInitializeEvent(
       PRKEVENT Event,
       EVENT_TYPE Type,
       BOOLEAN State);
```
– inițializare eveniment

```
LONG KeSetEvent(
       PKEVENT Event,
       KPRIORITY Increment,
       BOOLEAN Wait);
```
– activare eveniment (stare *signaled*)

```
LONG KeReadStateEvent(
       PRKEVENT Event);
```
– citește starea evenimentului

I/O pe porturi

Linux

```
#include <linux/ioport.h>
struct resource *request_region(unsigned long first,
unsigned long n, const char *name);
void release_region(unsigned long start, unsigned long n);
#include <asm/io.h>
unsigned inb(unsigned port);
void outb(unsigned char byte, unsigned port);
unsigned inw(unsigned port);
void outw(unsigned short word, unsigned port);
unsigned inl(unsigned port);
void outl(unsigned long word, unsigned port);
```

Windows

```
UCHAR READ_PORT_UCHAR(PUCHAR Port);
VOID WRITE_PORT_UCHAR(PUCHAR Port, UCHAR Value);
USHORT READ_PORT_USHORT(PUSHORT Port);
VOID WRITE_PORT_USHORT(PUSHORT Port, USHORT Value);
ULONG READ_PORT_ULONG(PULONG Port);
VOID WRITE_PORT_ULONG(PULONG Port, ULONG Value);
```

Întreruperi

Linux

```
#include <linux/interrupt.h>
int request_irq(unsigned int irq_no,
       irqreturn_t (*handler)(int irq_no, void *dev_id),
       unsigned long flags, const char *dev_name,
       void *dev_id);
```
flags: `IRQF_SHARED`, `IRQF_SAMPLE_RANDOM`, `IRQF_DISABLED`

```
void free_irq(unsigned int irq_no, void *dev_id);
irqreturn_t my_handler(int irq_no, void *dev_id);
```
irqreturn_t: `IRQ_HANDLED`, `IRQ_NONE`

Windows

```
ULONG HalGetInterruptVector(
       IN INTERFACE_TYPE InterfaceType,
       IN ULONG BusNumber,
       IN ULONG BusInterruptLevel,
       IN ULONG BusInterruptVector,
       OUT PKIRQL Irql,
       OUT PKAFFINITY Affinity);
```
InterfaceType: `Isa`, `Pci`, `Internal`

```
NTSTATUS IoConnectInterrupt(
       OUT PKINTERRUPT *InterruptObject,
       IN PKSERVICE_ROUTINE ServiceRoutine,
       IN PVOID ServiceContext,
       IN PKSPIN_LOCK SpinLock OPTIONAL,
       IN ULONG Vector,
       IN KIRQL Irql,
       IN KIRQL SynchronizeIrql,
       IN KINTERRUPT_MODE InterruptMode,
       IN BOOLEAN ShareVector,
       IN KAFFINITY ProcessorEnableMask,
       IN BOOLEAN FloatingSave);
```
InterruptMode: `LevelSensitive`, `Latched`

```
VOID IoDisconnectInterrupt(
       IN PKINTERRUPT InterruptObject);
BOOLEAN InterruptService(
       IN PKINTERRUPT Interrupt,
       IN PVOID ServiceContext);
```

Sincronizare cu întreruperi

Linux

```
void disable_irq(unsigned int irq);
```
– dezactivează o întrerupere

```
void disable_irq_nosync(unsigned int irq);
```
– dezactivează o întrerupere fără a aștepta terminarea

```
void enable_irq(unsigned int irq);
```
– activează o întrerupere

```
void local_irq_disable(void);
```
– dezactivează întreruperile pe procesorul curent

```
void local_irq_enable(void);
```
– reactivează întreruperile pe procesorul curent

```
void spin_lock_irq(spinlock_t *lock);
```
– spin_lock cu dezactivarea întreruperilor

```
void spin_unlock_irq(spinlock_t *lock);
```
– spin_unlock cu reactivarea întreruperilor

```
void spin_lock_irqsave(spinlock_t *lock,
       unsigned long flags);
```
– + salvarea stării întreruperilor

```
void spin_unlock_irqrestore(spinlock_t *lock,
       unsigned long flags);
```
– + revenirea la starea salvată a întreruperilor

Windows

```
BOOLEAN KeSynchronizeExecution(
       IN PKINTERRUPT Interrupt,
       IN PKSYNCHRONIZE_ROUTINE SynchronizeRoutine,
       IN PVOID SynchronizeContext);
```
– execută SynchronizeRoutine la IRQL egal cu al întreruperii

```
BOOLEAN mySynchronizeRoutine(
       PVOID Context);
```

Acțiuni amânabile

Linux – Taskleti

```
DECLARE_TASKLET(
       struct tasklet_struct *tasklet,
       void (*function)(unsigned long),
       unsigned long data);
DECLARE_TASKLET_DISABLED(
       struct tasklet_struct *tasklet,
       void (*function)(unsigned long),
       unsigned long data);
void tasklet_init(struct tasklet_struct *tasklet);

void tasklet_schedule(struct tasklet_struct *tasklet);
void tasklet_hi_schedule(struct tasklet_struct *tasklet);

void tasklet_enable(struct task_struct *tasklet);
void tasklet_disable(struct task_struct *tasklet);
```

Linux – Timere

```
#include <linux/sched.h>
void setup_timer(
       struct timer_list *timer,
       void (*function)(unsigned long),
       unsigned long data);

int mod_timer(
       struct timer_list *timer,
       unsigned long expires);

int del_timer(struct timer_list *timer);
int del_timer_sync(struct timer_list *timer);
```

Windows – DPC-uri

```
VOID KeInitializeDpc(
       IN PRKDPC Dpc,
       IN PKDEFERRED_ROUTINE DeferredRoutine,
       IN PVOID DeferredContext);
VOID DeferredRoutine(
       IN KDPC *Dpc,
       IN PVOID DeferredContext,
       IN PVOID SystemArgument1,
       IN PVOID SystemArgument2);
BOOLEAN KeInsertQueueDpc(
       IN PRKDPC Dpc,
       IN PVOID SystemArgument1,
       IN PVOID SystemArgument2);
```

Windows – Timere

```
VOID KeInitializeTimer(
       IN PKTIMER Timer);
BOOLEAN KeSetTimer(
       IN PKTIMER Timer,
       IN LARGE_INTEGER DueTime,
       IN PKDPC Dpc OPTIONAL);
BOOLEAN KeCancelTimer(
       IN PKTIMER Timer);
```

Locking softirq-uri

Linux

```
void local_bh_disable(void);
void local_bh_enable(void);
void spin_lock_bh(spinlock_t *lock);
void spin_unlock_bh(spinlock_t *lock);
void read_lock_bh(rwlock_t *lock);
void read_unlock_bh(rwlock_t *lock);
void write_lock_bh(rwlock_t *lock);
void write_unlock_bh(rwlock_t *lock);
```

Cozi de sarcini

Linux – Workqueues

```
#include <linux/workqueue.h>

DECLARE_WORK(
       name,
       void (*function)( struct work_struct *));
DECLARE_DELAYED_WORK(
       name,
       void (*function)( struct work_struct *));
INIT_WORK(
       struct work_struct *work,
       void (*function)( struct work_struct *));
INIT_DELAYED_WORK(
       struct delayed_work *work,
       void (*function)( struct work_struct *));

schedule_work(
       struct work_struct *work);
schedule_delayed_work(
       struct delayed_work *work,
       unsigned long delay);
int cancel_delayed_work(
       struct delayed_work *work);
void flush_scheduled_work(
       void);

struct workqueue_struct *create_workqueue(
       const char *name);
struct workqueue_struct *create_singlethread_workqueue(
       const char *name);

int queue_work(
       struct workqueue_struct *queue,
       struct work_struct *work);
int queue_delayed_work(
       struct workqueue_struct *queue,
       struct delayed_work *work,
       unsigned long delay);

void flush_workqueue(
       struct workqueue_struct *queue);
void destroy_workqueue(
       struct workqueue_struct *queue);
```

Windows – System worker threads

```
PIO_WORKITEM IoAllocateWorkItem(
       IN PDEVICE_OBJECT DeviceObject);
VOID IoFreeWorkItem(
       IN PIO_WORKITEM IoWorkItem);
VOID IoQueueWorkItem(
       IN PIO_WORKITEM IoWorkItem,
       IN PIO_WORKITEM_ROUTINE WorkerRoutine,
       IN WORK_QUEUE_TYPE QueueType,
       IN PVOID Context);
VOID WorkItem(
       IN PDEVICE_OBJECT DeviceObject,
       IN PVOID Context);
```

Thread-uri kernel

Linux – Kernel threads

```
#include <linux/kthread.h>
#include <linux/sched.h>

struct task_struct *kthread_create(
       int (*threadfn)(void *data),
       void *data, const char namefmt[], ...);
struct task_struct *kthread_run(
       int (*threadfn)(void *data),
       void *data, const char namefmt[], ...);
int wake_up_process(
       struct task_struct *p);
fastcall NORET_TYPE void do_exit(
       long code);
```

Windows – System threads

```
NTSTATUS PsCreateSystemThread(
       OUT PHANDLE ThreadHandle,
       IN ULONG DesiredAccess,
       IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
       IN HANDLE ProcessHandle OPTIONAL,
       OUT PCLIENT_ID ClientId OPTIONAL,
       IN PKSTART_ROUTINE StartRoutine,
       IN PVOID StartContext);
VOID StartRoutine(
       IN PVOID StartContext);
NTSTATUS PsTerminateSystemThread(
       IN NTSTATUS ExitStatus);
```

Linux – I/O Block Layer

```
int register_blkdev(
       unsigned int major,
       const char *name);
void unregister_blkdev(
       unsigned int major,
       const char *name);
```
– înregistrare/deînregistrarea unui dispozitiv de tip bloc
```
struct gendisk *alloc_disk(int minors);
```

```
void del_gendisk(struct gendisk *disk);
```
– alocarea/ștergerea unui disc
```
void add_disk(struct gendisk *disk);
```
– adăugarea unui disc
```
struct request_queue *blk_init_queue(
       request_fn_proc *rfn,
       spinlock_t *lock);
void blk_cleanup_queue(struct request_queue *q);
```
– crearea/ștergerea unei cozi de cereri
```
typedef void (request_fn_proc) (struct request_queue *q);
typedef int (make_request_fn) (
       struct request_queue *q,
       struct bio *bio);
```
– funcții pentru prelucrarea cererilor din coadă
```
struct request *blk_peek_request(struct request_queue *q);
void blk_start_queue(struct request_queue *q);
struct request *blk_fetch_request(struct request_queue *q);
void blk_requeue_request(
       struct request_queue *q,
       struct request *rq);
```
– funcții pentru lucrul cu cereri din coada de cereri
```
sector_t blk_rq_pos(const struct request *rq);
unsigned int blk_rq_bytes(const struct request *rq);
int blk_rq_cur_bytes(const struct request *rq);
```
– obținerea numărului de sectoare și octeți dintr-o cerere
```
bool blk_end_request(
       struct request *rq,
        int error,
       unsigned int nr_bytes);
bool __blk_end_request(
       struct request *rq,
       int error,
       unsigned int nr_bytes);
void blk_end_request_all(struct request *rq, int error);;
void __blk_end_request_all(struct request *rq, int error);;
```
– încheierea unei cereri
```
struct bio *bio_alloc(gfp_t gfp_mask, int nr_iovecs);
```
– alocarea unui bio
```
struct bio *bio_clone(struct bio *bio, gfp_t gfp_mask);
```
– clonarea unui bio existent într-un alt bio
```
void submit_bio(int rw, struct bio *bio);
```
– submiterea unui bio dispozitivului descris în structură
```
void complete(struct completion *x);
```
– marcarea unui eveniment (completion) ca fiind încheiat/realizat
```
alloc_page(gfp_mask);
```
– alocarea unei pagini
```
int bio_add_page(
       struct bio *bio,
       struct page *page,
       unsigned int len,
       unsigned int offset);
```
– adăugarea unei pagini unui bio
```
__bio_kmap_atomic(bio, idx, kmtype);
```
– maparea unui bio într-un spațiu din highmem
```
__bio_kunmap_atomic(addr, kmtype);
```
– demaparea unui bio dintr-un spațiu din highmem
```
bio_sectors(bio);
```
– numărul de sectoare cuprins de bio
```
static inline unsigned int bio_cur_bytes(struct bio *bio);
```
– numărul de octeți ocupat de bio
```
bio_data_dir(bio);
```
– direcția de parcurgere pentru bio (read/write)
```
bio_for_each_segment(bvl, bio, i);
```
– parcurgerea fiecărui segment al unui bio
```
rq_for_each_segment(bvl, _rq, _iter);
```
– parcurgerea fiecărui segment al unei cereri

Windows – Lucrul cu dispozitive

```
NTSTATUS IoCreateDevice(
       PDRIVER_OBJECT DriverObject,
       ULONG DeviceExtensionSize,
       PUNICODE_STRING DeviceName,
       DEVICE_TYPE DeviceType,
       ULONG DeviceCharacteristics,
       BOOLEAN Exclusive,
       PDEVICE_OBJECT *DeviceObject);
```
– creează un dispozitiv de tip DEVICE_OBJECT
```
NTSTATUS IoGetDeviceObjectPointer(
       PUNICODE_STRING ObjectName,
       ACCESS_MASK DesiredAccess,
       PFILE_OBJECT *FileObject,
       PDEVICE_OBJECT *DeviceObject);
```
– obține un pointer la obiectul descris de ObjectName
```
VOID ObDereferenceObject(PVOID Object);
```
– decrementarea contorului de referință a unui obiect; opus la `IoGetDeviceObjectPointer`
```
PIRP IoBuildSynchronousFsdRequest(
       ULONG MajorFunction,
       PDEVICE_OBJECT DeviceObject,
       PVOID Buffer,
       ULONG Length,
       PLARGE_INTEGER StartingOffset,
       PKEVENT Event,
       PIO_STATUS_BLOCK IoStatusBlock);
```
– crearea unui IRP pentru o cerere I/O procesată sincron
```
NTSTATUS IoCallDriver(
       PDEVICE_OBJECT DeviceObject,
       PIRP Irp);
```
– transmiterea unui IRP unui dispozitiv

Linux Memory Mapping

`struct page` - descrie o pagină fizică
`struct vm_area_struct` - descrie o zonă de memorie virtuală
`struct mm_struct` - descrie toate caracteristicile legate de memorie asociate unui proces
```
remap_pfn_range(
       struct vm_area_struct *vma,
       unsigned long addr,
       unsigned long pfn,
       unsigned long size,
       pgprot_t prot)
```
- mapează un spațiu de memorie fizică în spațiu virtual utilizator

Windows Memory Mapping

`struct MDL` - descrie layout-ul fizic al unui spațiu de adresă virtual
```
PMDL MmAllocatePagesForMdl(
       IN PHYSICAL_ADDRESS LowAddress,
       IN PHYSICAL_ADDRESS HighAddress,
       IN PHYSICAL_ADDRESS SkipBytes,
       IN SIZE_T TotalBytes)
```
- alocare pagini fizice pentru MDL
```
VOID MmFreePagesFromMdl(
       IN PMDL MemoryDescriptorList)
```
- eliberează pagini fizice alocate unui MDL
```
NTKERNELAPI VOID MmMapLockedPagesSpecifyCache(
       IN PMDL MemoryDescriptorList,
       IN KPROCESSOR_MODE AccessMode,
       IN MEMORY_CACHING_TYPE CacheType,
       IN PVOID BaseAddress,
       IN ULONG BugCheckOnFailure,
       IN MM_PAGE_PRIORITY Priority)
```
- mapare între zone de memorie fizică și o zonă de memorie virtual contiguă `VOID MmMapUnmapedLockedPages(`
```
       IN PVOID BaseAddress,
       IN PMDL MemoryDescriptorList)
```
- undo MmMapLockedPagesSpecifyCache
```
PVOID MmGetSystemAddressForMdlSafe(
       IN PMDL Mdl,
       IN MM_PAGE_PRIORITY Priority)
```
- întoarcere pointer către zona de memorie virtuală descrisă de un MDL `VOID IoFreeMdl(`
```
       IN PMDL Mdl)
```
- eliberare MDL