Memory

CAOS, Lecture #08

Physical Address Space

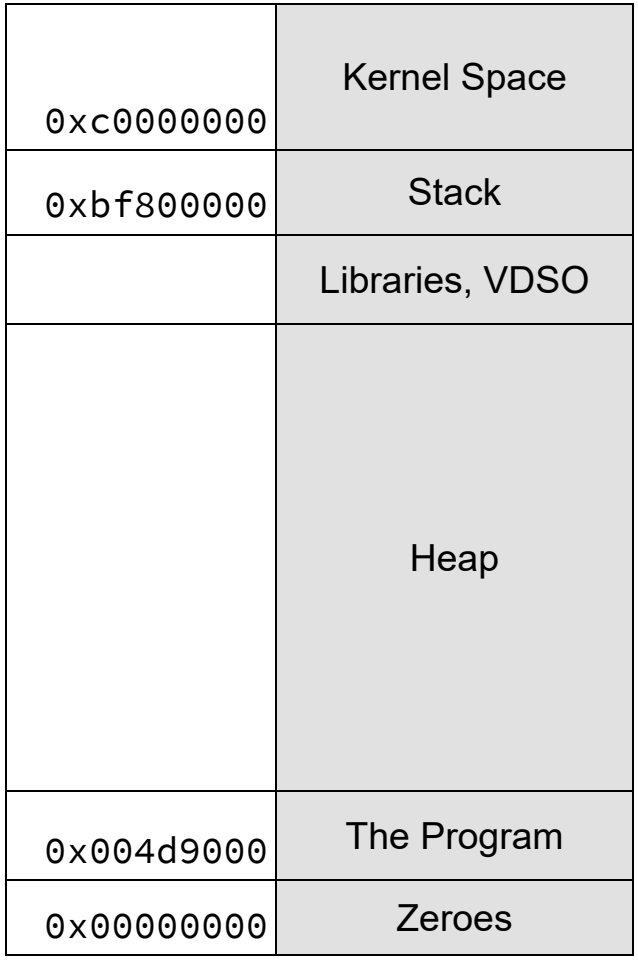
- Up to 2^{32}, 2^{36} (Intel) or 2^{64} bytes using address bus
- Used both by memory and peripherals

Virtual Address Space

- Each process has it’s own isolated memory
- Starting from 0 address
- For 32-bit systems: up to 4Gb
- For 64-bit systems: only 48 bit - up to 256Tb

Virtual Address Space (IA-32 - x86)

- Address Range `0x00000000` `0xffffffff`
- Available 3Gb (Linux) or 2Gb (Windows)
- Higher addresses are occupied by Kernel



Virtual Address Space

- Programs and Libraries are split into sections: .text, .data, .ro_data
- Stack grows upside down
- Head grows from down to up
- VDSO - small mapping from Kernel to Userspace

Virtual Address Space

- Programs and Libraries are split into sections: .text, .data, .ro_data

This fact allows to save memory usage if several instances of program are running or libraries in use

/proc/process_number/maps

- Text file in virtual file system procfs
- Subdirectory self - is a link to the current process

```
cat /proc/self/maps
```

/proc/process_number/maps

```
004d9000-004e1000 r-xp 00000000 08:02 655669     /usr/bin/cat
004e1000-004e2000 r--p 00007000 08:02 655669     /usr/bin/cat
004e2000-004e3000 rw-p 00008000 08:02 655669     /usr/bin/cat
|- address rang   |    |        |- st_dev |      |- file name
                  |    |                  |
    access right -|    |           inode -|
                  |
             offset -|
```

Addressing Modes

- Segment based (Real Mode)
- 2-Level Page-Based (x86)
- 3-Level Page-Based (x86+PAE)
- 4-Level Page-Based (x86_64)

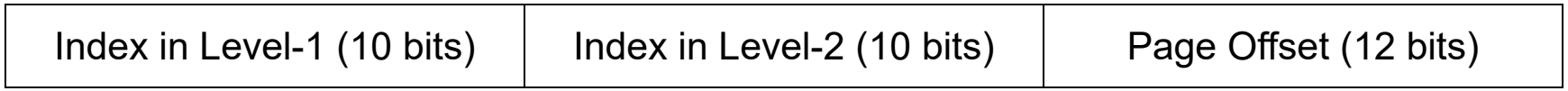
Segment Model

- All registers are 16-bit
- The problem: it is required to access up to 1Mb of memory
- Segment types:
  - code segment (CS)
  - stack segment (SS)
  - data segment (DS)
- Addr = (Segment << 4) + Offset

Page Model

- All address space memory space is split into pages of equal size
- Individual pages have attributes

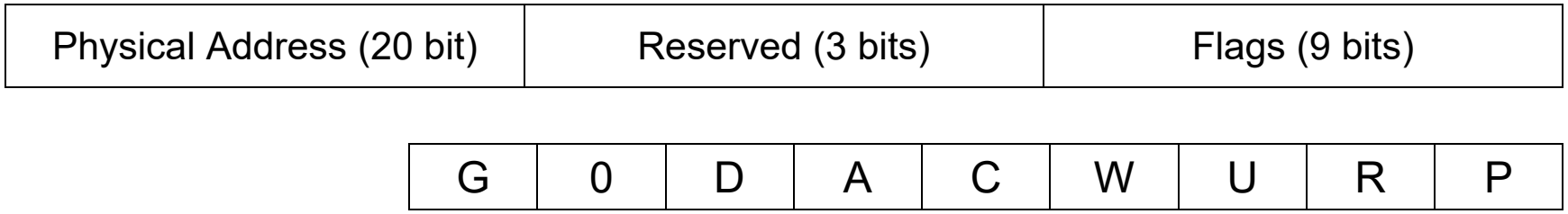
x86 Paging



- The CR2 register points to Level-1 table address
- Each Level-1 entry is an address of Level-2 table
- Level-2 table contains page entry

- For 32-bit systems:
  - –page size is 4K
  - –each table contains 1024 records

Page addressing x86



- G (global)
- D (dirty)
- A (accessed)
- C (caching)
- W (write-throught)
- U (userspace)
- R (read+write)
- P (present)

Page Fault

- P==0
- Exception generated by CPU
- Not and error:
  - for a memory it might be in swap
  - for a mapped file - not in a buffer
  - trying to modify Write-Protected: Copy-On-Write
- Exception is to be handled by Kernel

Hardware Support

- Memory Management Unit
- Translation Lookaside Buffer
- Page size is determined by Processor internals x86(64): 4Kb, 2Mb or 2Gb
- Huge Pages: might have distinct pages of big size

Memory Allocation

- C++: new и new[] **operators** - uses malloc/calloc in implementation
- Си: **functions** malloc/calloc allocates memory in a heap
- malloc/calloc/free - are not system calls!
- Might have different implementations

Different implementations problem

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

MSVC

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r);
}
```

Different implementations problem

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

MinGW

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r); // Segmentation Fault
}
```

mmap

```
void *mmap(void *addr, size_t length,
           int prot, int flags,
           int fd, off_t offset);
```

- addr - where to place (just a recomendation, might be NULL)
- length - the size of allocation (mapping)
- prot - memory protection flags (EXEC/READ/WRITE/NONE)
- flags - mapping flags (SHARED - shared mapping, PRIVATE - use Copy-On-Write strategy)
- fd, off_t - file to map and offset from beginning (might be -1 value)

malloc/calloc

- Might allocate new pages using mmap
- Has internal table to map allocated regions
- calloc zeroes allocated memory and might work faster then malloc + memset

Double Free Memory Corruption

```
void* ptr = malloc(SIZE);
free(ptr);
free(ptr);  // double free
```

If the argument does not match a pointer earlier returned by a function in POSIX.1-2008 that allocates memory as if by malloc(), or if the space has been deallocated by a call to free() or realloc(), the behavior is undefined.