

Operating Systems

Daniel Gruss

2023-10-08

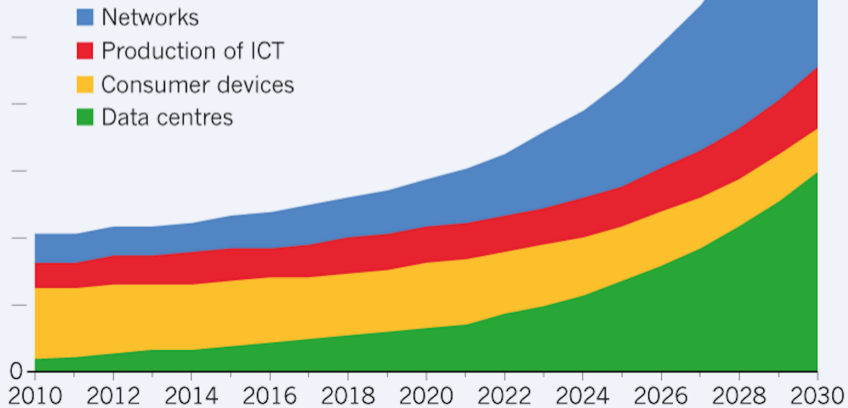
Why OS?

9,000 terawatt hours (TWh)

©nature

ENERGY FORECAST

20.9% of projected
electricity demand



It's 2030. You get to make an OS design decision:

one page table per process

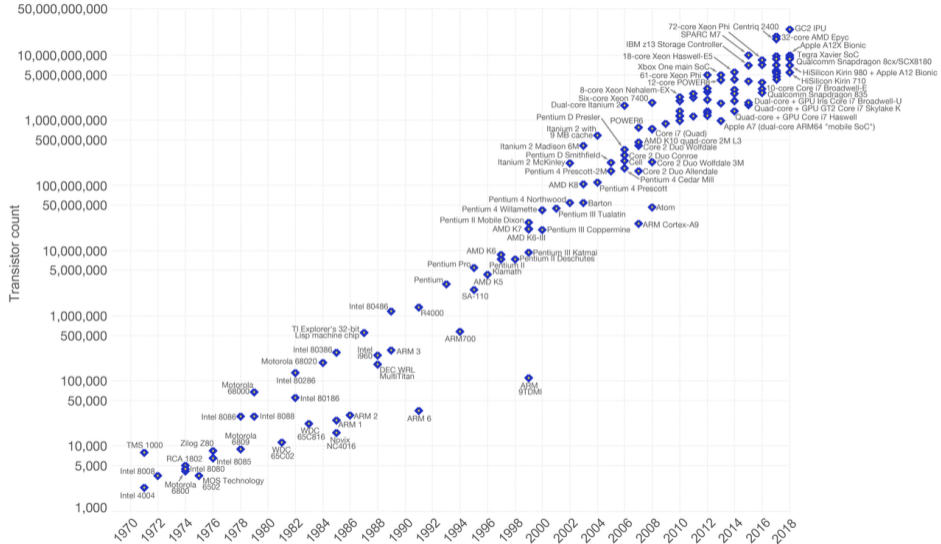
vs

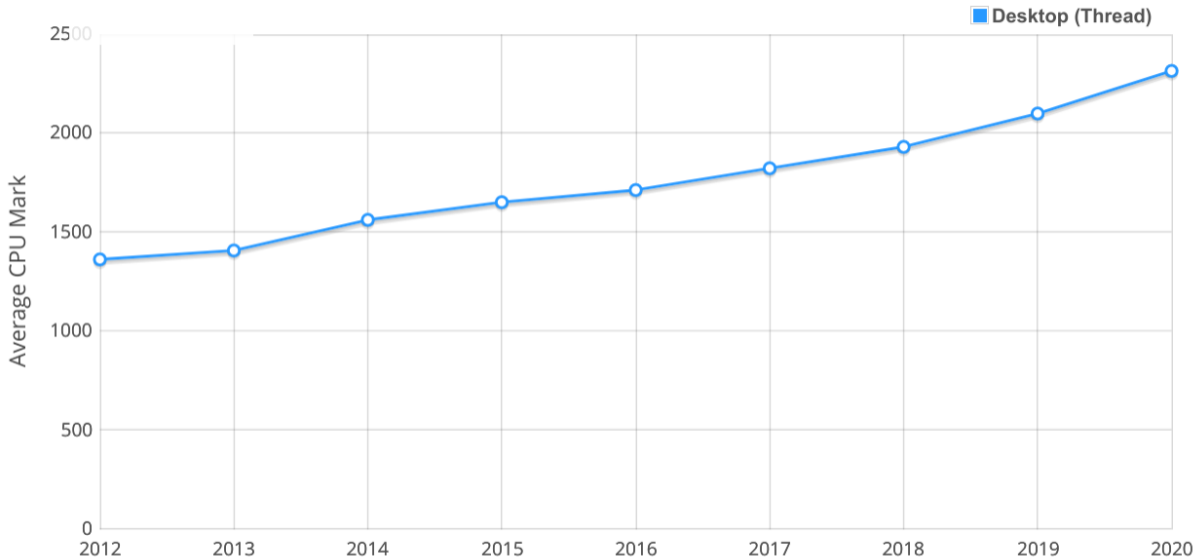
half a page table per process + 1 for the kernel

0.1%

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

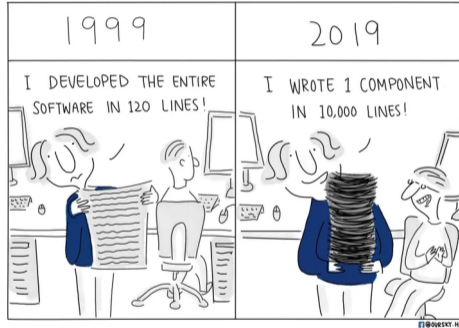






YESS...

MOAR SOFTWARE COMPLEXITY

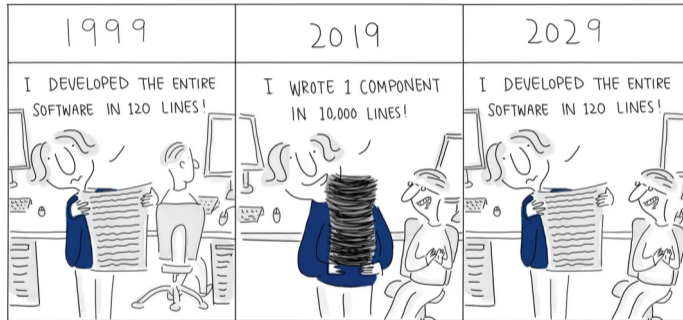


SWEB is C and C++

Which programming language would you prefer?

Table 4. Normalized global results for Energy, Time, and Memory

Total					
Energy		Time		Mb	
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84



You and SWEB

Organizational Details

SWEB

Assignment 1

Booting SWEB

You and SWEB

Organizational Details

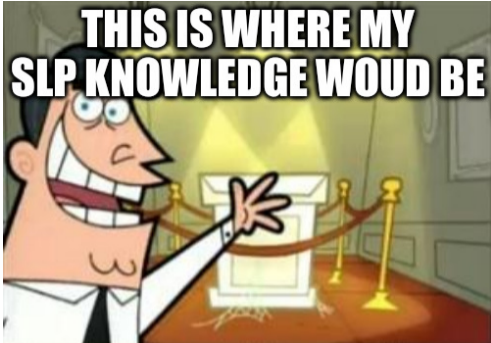
SWEB

Assignment 1

Booting SWEB



- Knowledge from earlier lectures
 - ESP, OOP1(!), SLP(!!), CON, ...
 - **Everything** taught in SLP, OOP1, and ESP will be practically applied **by you** in OS
- Reasonable C/C++ experience
- Team work + Time management





- Self assess whether you are prepared:
- “If I would do SLP/OOP1/ESP/CON again **with the knowledge you have now** I would **be able** to get a **1** easily.”
- Hint: ask a colleague whether they agree with your self-assessment ;)



- 3-4 (if SLP is one of them): Congratulations, you are well prepared!
- 2-3: You are not well prepared → additional time investment necessary.
- 1-2: You are not prepared → huge time investment or positive grade at risk.
- 0: You are seriously unprepared → even with huge time investment positive grade is at risk.

- Depends significantly on your knowledge and experience ...
- ... and on your team members.
- Short good solutions add around 4000 lines of code → 1000 lines of code per member



- No need to be afraid!
- Before COVID, 35%-45% of students got a 1 in OS, every semester.
- Is it easy?
- No.
- Is it doable?
- Yes, very much so.



- No one should be afraid of OS.
- We even added a course to prepare you for OS (and other courses): SLP!

→ SLP is sort of the “booster” for OS

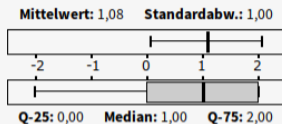
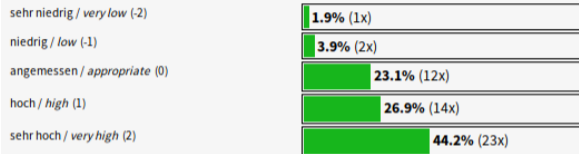
- Statistics confirm: if you did well in SLP, OS becomes very much doable
- Steep learning curve in the beginning? No worries, you will manage!



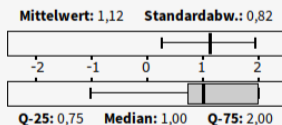
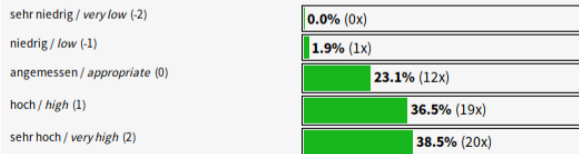
Note to students

While we have made an effort to simplify these projects for you, most Duke students find these projects sufficiently difficult to dominate their lives during the one-semester course. A common misconception from earlier semesters is that we are sadistic individuals who enjoy seeing students suffer. Actually, this is not the case. We enjoy seeing students who are proud of what they have accomplished and excited by the power that flows from a relatively small set of simple abstractions in an operating system, even a toy one like Nachos.

Wie beurteilen Sie Ihren für diese LV nötigen Arbeitsaufwand - gemessen an den ECTS-Punkten, die Sie erhalten? / How would you rate the workload for this course - measured against the ECTS credit points you received? (52 x beantwortet)



Wie beurteilen Sie die inhaltlichen Anforderungen (Schwierigkeit) dieser LV? / How would you rate the content requirements (difficulty) of this course? (52 x beantwortet)



Wie zufrieden sind Sie mit der LV insgesamt? /

How satisfied are you with this course in general? (52 x beantwortet)

sehr unzufrieden / *highly dissatisfied* (1)

1.9% (1x)

(2)

1.9% (1x)

(3)

5.8% (3x)

(4)

15.4% (8x)

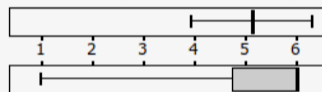
(5)

21.2% (11x)

sehr zufrieden / *highly satisfied* (6)

53.9% (28x)

Mittelwert: 5,13 Standardabw.: 1,18



Q-25: 4,75 Median: 6,00 Q-75: 6,00

Wie stark trifft die folgende Aussage Ihrer Meinung nach auf die LV zu?:

"In dieser LV werden alle Studierenden gleichermaßen fair behandelt (unabhängig von Lerntyp, Geschlecht, Ethnie)" /
How well does the following statement describe this course, in your opinion?

"On this course, all students are treated fairly and equally (regardless of their learning style, gender and ethnicity)." (52 x beantwortet)

trifft überhaupt nicht zu / *not at all well* (1)

0.0% (0x)

(2)

0.0% (0x)

(3)

1.9% (1x)

(4)

1.9% (1x)

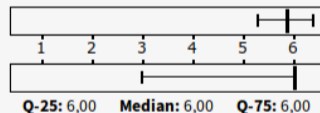
(5)

5.8% (3x)

trifft völlig zu / *very well* (6)

90.4% (47x)

Mittelwert: 5,85 Standardabw.: 0,53



Vielleicht hätte man hervorheben sollen, dass uSTL NICHT kompatibel zu STL ist! `ustl::list` gehört eigentlich verboten. Man kommt in Teufels Küche, wenn man von ihr ableiten möchte. Man gibt sich einer Illusion hin, die nicht stimmt. Einfügen und Löschen sind NICHT effizient und Iteratoren verlieren die Gültigkeit, wo sie es nicht sollten. Trotzdem war es hilfreich.

Check the standard!

```
iterator erase (iterator position);  
iterator erase (iterator first, iterator last);
```

erase verwenden ohne den iterator upzudaten geht mit der GNU C++ STL *meistens...*
→ gefährlich sich anzugewöhnen den iterator zu ignorieren



`http://baptiste-wicht.com/posts/2012/12/
cpp-benchmark-vector-list-deque.html`

Was gefällt Ihnen an dieser LV besonders gut? 28 Antwort(en) vorhanden.

- Der "Klick" Moment wenn man endlich verstanden hat worums geht und die Einstellung zur LV von "Sehr abgeneigt" zu "Begeistert" umschwingt. [WS16/17]

- Assertions sind eine Wunderwaffe beim Debuggen ;)
- wenn man selber welche einfügt, dann sind sie noch hilfreicher ;)
- gegen ende hin waren ca 1/3 der zeilen Asserts - Hat jedoch dem tutor nicht so gut gefallen weil dadurch der code unleserlich war

**STUDENT DISCOVERING THE
F-KEYS IN SWEB AFTER EXERCISE IS OVER**



**FFFFFFF
FFFFFFF
FFFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU
UUUU-**



- We try to improve our support constantly
- Feedback, Evaluations

“We will not lower the bar, but we will do what we can to help you over it.”



- Bad time management
- Pair programming (it's not efficient enough!!)
- Problems with working in a team
- No C/C++ experience
- Not willing to learn and use C/C++



**REALIZING THAT YOU
ONLY NEED ABOUT 20 MORE
HOURS OF WORK TO GET A 1**

IT'S 2 HOURS BEFORE THE DEADLINE

You and SWEB

Organizational Details

SWEB

Assignment 1

Booting SWEB

- Will help you with all your problems
 - Especially: if you're stuck on a problem for more than a few hours → ask
- Interactive Programming units
- Student Debates
- Design reviews
- Question hours
- Review meetings (Abgabegespräche)



- Website: <https://iaik.tugraz.at/os>
- Discord (tag your tutor, also in your own group channels!)
- Email: os@iaik.tugraz.at
- Consultation hour: send us an email
- *During the question days: slower response times on Discord/email ;)*



- Take place before the design deadlines
- You are expected to bring a sketch/summary of your design
- You are expected to have done proof of concept implementations by then
- Prepare to defend your ideas



- Only code! No design document!
- Proof of concept implementation
- You get up to 5 points, depending on the test system score of your submission.



- You have to:
 - push regularly into the provided repository
 - use your real name and email address for commits

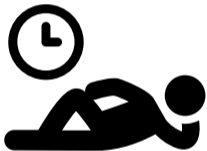
- Personal maximum number of points
- Unlock points:
 - For each commit you unlock 0.5 points
 - For each 10 LoC added you unlock 0.5 points
 - Max. 10 points per day can be unlocked
- Average: > 55 points unlocked
- Test system shows your current personal maximum number of points

- We even give you master access to your repository, **but:**
 - renaming repo = your group fails the course
 - changing repo path = your group fails the course
 - adding/removing group members = your group fails the course

- Every team member has to participate:
 - Making coffee, fetching pizza, etc. is **not** enough
- We expect **all** members to have a high level overview of design and implementation
- We expect **every** member to be able to read, explain and **change** their own implementation, even if it's the code of another team member
- Otherwise: 0 points



Every team member must contribute points to the mandatory task categories!



- Set internal deadlines for your team members
 - No idea how to split the work and choose good internal deadlines? Ask tutor for recommendations!
- Missed deadline? → re-assign task
- Repeatedly missed deadline → tell tutor to remove team member from team



- Team member(s) not contributing enough → contact tutor via mail!
 - There will be more students with the same problem as you have
 - Team doesn't want to split but you want to leave? → also possible
- Merges are possible
- Don't pull anyone through
- small number of bonus points as compensation (if **we** think it's appropriate)

Not happy with your tutor's resolution? → **Talk to me**, we'll figure out a solution

- Existing syscall wrapper functions are POSIX-compatible
 - Do not change signatures!
 - We have automated tests using the POSIX interface
- If you implement new functions, try to make signatures POSIX-compatible

- Goal: a stable and fault tolerant operating system
- How? By writing test programs
- Think of test cases while designing and implementing
- Think of basic test scenarios as well as corner cases

- Writing **numerous** test programs is unavoidable
- Test programs are supposed to show whether your implementation works according to the assignment
- You will get points for the test programs
- We have our own secret test programs

- Bad: Not knowing of a problem until the review meeting
- Better: Knowing of a problem but not solving it (probably because time ran out)
- Best: Knowing of a problem sufficiently before the deadline and solving it (maybe with the help of a teaching assistant)

A man with a white beard and hair, wearing a light blue button-down shirt, is sitting at a desk. He is looking at a laptop screen with a thoughtful expression. He is holding a white mug in his left hand. The background shows a modern office setting with a white shelf holding two small potted plants and a white circular object.

**REALIZING THAT YOU
ONLY NEED ABOUT 20 MORE
HOURS OF WORK TO GET A 1**

The same man from the top panel is shown in the same office setting. He is now smiling broadly at the laptop screen, appearing much more confident and relaxed. He is still holding the white mug.

IT'S 2 HOURS BEFORE THE DEADLINE

- Tag the commit you want to submit: `git tag SubmissionD1 [commit_hash]`
- Push to repository: `git push / git push --tags`
- Read the commit ID:

```
git show SubmissionD1
commit 196bc4a704f37d7f969d27a258b513693e3b30f4
Author: Daniel Gruss <daniel.gruss@iaik.tugraz.at>
```

Test System will acknowledge your submission!

- Small fixes → small deduction
- Do this at the same time:
 - Mail to os@iaik.tugraz.at!
 - Submit and retag your fixed version!

Reference (=100%)

- Design: 5 points
- Task 1: 20 points
- Task 2: 10 points
- Elective tasks*: 15 points (or more)

Minimum

- Design: 1 point
- Mandatory tasks: 15 points
- And in total: 25 points (=50%)

(*) See the list of elective tasks on the Website!



Reference (=100%)

- Design: 5 points
- Mandatory tasks: 40 points
- Elective tasks*: 5 points (or more)

Minimum

- Design: 1 point
- Mandatory tasks: 15 points
- And in total: 25 points (=50%)

(*) See the list of elective tasks on the Website!

Minimum requirements

- A1: 25 of 50 points
- A2: 25 of 50 points

Limits

- A1: max. 60 points
 - Unlimited if $A2 \geq 40$ points
- A2: unlimited points



- mid-term exam: 35%
- the practicals: 65%



To pass the class, you have to acquire

- overall at least 55%
- at least 50% of the possible points on written exam
- at least 50% of the possible points in the practicals



- Didn't pass any assignment or exam?
- No worries, we have a second chance for everyone.
- DD → you can continue without Second Chance but with 0 DD/PoC points
- PoC → Second Chance PoC (examined by Lecturers)
- Exercise → Second Chance Exercise (examined by Tutors + Lecturers)
- Exam → Second Chance Exam

Getting a positive grade remains as easy, getting a 1 gets harder (with every second chance)



Points

1. Take the team score
2. cap it by the personal unlocked points
3. apply DD deduction
4. apply AG deduction

→ final points for assignment



Points (Example)

1. You get 48 points as a team → 48
2. Your personal unlocked points are 43 → 43
3. Your group had 4 points on the PoC but you didn't say enough in the DD → 39
4. You get a 75% deduction because you couldn't really figure out where to do what in your SWEB in the AG → 10 points → second chance



Points (Example)

1. You get 48 points as a team → 48
2. Your personal unlocked points are 57 → 48
3. You have a 25% deduction because you didn't say enough in the DD → 36
4. You get a 20% deduction because you took too long with making changes → 29
→ barely positive



Points (Example)

1. You get 45 points as a team → 45
2. Your personal unlocked points are 70 → 45
3. You participated in the DD properly → 45
4. You get a 5% deduction because you took rather long with making changes → 43
→ good grade



Marks

- genügend: 55-66
- befriedigend: 67-78
- gut: 79-89
- sehr gut: 90+

As a group

- You explain what you implemented and how you tested it
- Together with the teaching assistant you determine the points of your group

As a group member

- You are able to read, explain and change the code
- You can implement small new features or extend existing ones
 - just quickly, no rigorous testcase writing
 - tutor will stop you as soon as it's clear that you deserve the points
- Otherwise you will get less points

- Discussions with other teams are appreciated
- But: **no collaboration!**
- We check for plagiarism
- Similarities → teams are questioned
- Both teams: 0 points in either case
- At least one team: “Ungültig/Täuschung” (with all its consequences)

- Do not provide your source code to other teams
- Make sure your source code is protected against unintended access from others
- Do not use source code from previous years
 - Code from another team → plagiarism
 - Your own code (not exactly the same team)
→ not allowed
 - Your own code (exactly the same team)
→ allowed, probably not the best idea ;)

Questions so far?

A “lot of pressure”?

because...

- “you never know whether you already have enough points”
- because exercise interview, because of exam → no worry - there are second chances
- also: each group gets 2 🎲 tokens
- Use one 🎲 to ask whether your group has 0-20, 20-40, 40-60, or 60+ points according to the test system’s prediction

→ **Wisely** choose as a team when to use them!

⚠️ test system’s prediction **can deviate substantially** from tutor’s grading



**THE TESTSYSTEM
DECLARING US
ONE OF THE TOP TEAMS**

**OUR OS NOT
ABLE TO SWAP
IN COW PAGES**

A meme image featuring two men. On the left, a man with curly hair is wearing a VR headset and smiling broadly, looking upwards. On the right, a man with dark hair and a beard is looking towards the first man with a neutral expression. The background is a plain, light-colored wall.

**ME THINKING MY ASSIGNMENT
IN OS IS WORKING PRETTY GOOD**

MICHAEL

A “lot of pressure”?



because...

- “2 hour exercise interview is really long”
 - we can offer a break
- agree with your tutor beforehand whether you want a break in the middle
- but: no coding during the break

You and SWEB

Organizational Details

SWEB

Assignment 1

Booting SWEB



- VU Amsterdam: Minix (1987), Minix3 (2005)
- Berkeley: Nachos (1992)
- Stanford: Pintos (2004)
- Graz
 - Nachos until 2006
 - Now: SWEB

- BS KU 2004/2005
- Advanced group of students together with Philip Lawatsch and Bernhard Tittelbach
- Many subsequent projects
- BS KU: since 2007 SWEB only

- Minimalistic operating system kernel
- Runs on x86-32, x86-64, ARM
- Emulated using qemu
- Important features are missing
- Your task: Make your SWEB a beautiful, feature-rich kernel



- Mouse driver
- Window manager
- Network driver
- Soundblaster driver
- Gameboy emulator
- 3D game engine
- Running it on a small ARM board with only 256KB RAM



- Try out the tutorials on <http://iaik.tugraz.at/os>
 - Set up development environment
 - Implement your first syscall
- Get acquainted with the source code: Try out implementing things in SWEB
- Start with the practicals **NOW!** (or rather already a week ago)

You and SWEB

Organizational Details

SWEB

Assignment 1

Booting SWEB

- Base line SWEB: a user process **is a** (kernel) thread
- We want: multiple threads per user process
- What do we have to change?

- Each thread has its own instances of some resources
 - id, stack, registers, status, ...
- Other resources are shared among all threads
 - memory, files, ...

- How to “use” multithreading?
- Syscalls!
- Which ones?
- You decide - but function names and arguments have to be POSIX-compatible!
- Minimum requirements:

`pthread_create`

`pthread_exit`

`pthread_cancel`

`pthread_join`

- By definition: Operating system is written by people who know what they do
- User programs?
- System calls provide a “safe” interface
- Control flow is transmitted to kernel code
- Typical syscalls: `fork()`, `read()`, `write()`, `execve()`, `wait()`, `exit()`
- You will step through a syscall in the tutorial this week!

- `fork()` creates a new process by duplicating the calling process
- The new process (=the child), is an exact duplicate of the calling process (=the parent)
- Interesting in combination with multithreading!

- Replaces the current process image with a new process image
- `exec` with arguments → more points
- `fork()/exec()` combination often used



- `sleep()` sets a thread asleep for a given number of seconds
- `clock` returns how much cpu time the current process consumed

- `exit()` terminates the current process
- Already implemented, but ...
- ... you will break the current implementation with multithreading



- I/O syscalls are already implemented, but ...
- ... they use global (not process specific) file descriptors
- Why is that a problem?

- Threads need synchronization
- Kernel has mutexes and condition variables
- We want: mutexes, condition variables and semaphores, both in kernelspace and userspace
- Pure userspace implementation (except for initialization and for going to sleep)
- Implement test programs (Readers-Writers-Problem, Sleeping Barber, etc.)



- Own ideas are the most fun!
- See <https://www.iaik.tugraz.at/teaching/materials/os/assignments/> for suggestions
- Please note: Assignment 1 tasks will **only** be counted in Assignment 1 assessment
Assignment 2 tasks will **only** be counted in Assignment 2 assessment

Assignment 1

- Deadline Design-PoC: **See Timeline**

- The design proof-of-concept comprises your implementation so far, as is. It will be graded by the test system and you will talk 30 minutes with your tutor about your progress.

- Deadline Implementation: **See Timeline**

- Minimum requirements:

- At least **15** points from the two mandatory parts (multi threading and fork)
- At least **1** point from the design proof-of-concept
- At least **25** points in total

- 100% equals 50 points

- [Tutors Points List A1 \(not intended for student use\)](#)

In Assignment 1 you will acquire basic knowledge about the inner workings of multi threading and multi processing on modern operating systems.

Task 1: Multi Threading

You will start with the [basic SWEB kernel](#) which has neither multi threading nor multi processing implemented for userspace programs. Instead there is one thread per process. You have to figure out what to change to make SWEB support multi threading.

Threads share their address space with all other threads within the same process. However, each thread has its own stack, own set of cpu register values and maybe other resources. When the operating system interrupts a thread, it has to store these resources so that the thread execution can be continued later.

Interaction between userspace and kernelspace is only possible through syscalls. You can implement any syscall you like. However, for reasons of automated testing of your implementation you are required to **comply with the POSIX standard in terms of function names and parameters**.

You will need to implement at least the following syscalls:

1. `pthread_create` to create a new thread
2. `pthread_exit` to allow a thread to terminate itself
3. `pthread_cancel` to terminate another thread
4. `pthread_join` to wait for termination of another thread

Furthermore you are required to implement several user programs which demonstrate that your implementations works as expected.

Task 2: Fork

We want to give our user programs the ability to start new user programs. The POSIX compatible way of doing so, is to use the fork syscall (and the exec syscall). Fork creates an exact copy of the currently running process, with only the calling thread copied to the new process. The copied thread in the new process continues execution right after the fork call. We want to remind you to comply exactly to the POSIX standard for reasons of automated testing.

Additional Tasks

The two mandatory tasks are designed to give 30 points (of 50 points). We expect you to choose your own additional tasks. In the [Tutors Points List A1 \(not recommended for student use\)](#) you find a long list with numerous suggestions what you could implement. **DON'T** just use this list. Neither you nor we will be happy with the result. These are bonus tasks, come up with your own ideas, come up with your own improved variants of features.

We strongly encourage implementing your own interesting ideas, as long as they are related to this course, by giving you points. But, before you start, ask your tutor. For tasks that fall into one of the topics of Assignment 2 you can't get any points in Assignment 1 (and the other way round).



- Don't use it.
- No explanation. No details on how much you have to do for which point.
- Not suitable to choose tasks.
- Not suitable to estimate your points (points vary a lot!).

- Tutors Point List says **2 points**
 - “ah, actually cancel only works up to 5 times, then it stops working” → 0-1 points
 - “actually, we have implemented it including cancelstate, canceltype, cleanup push, pthread keys, correct cancel and join interaction” without any problems → maybe 8-12 points
- **You see?** The **2 points** doesn't tell you much! Can be a lot more, can be a lot less, all depends on how much you do.
- 2 points refers to a minimal correct working function that implements the most basic functionality of that syscall function



- “Is it mandatory to implement every sentence and everything that is mentioned on the (for example) `pthread_cancel` man page?”
 - No. Minimal correctly working cancel is 2 points. Everything beyond (cancel types, cancel states, etc.) is all elective/bonus.
- **70%** of the points you will collect in the OS exercises will be elective/bonus.

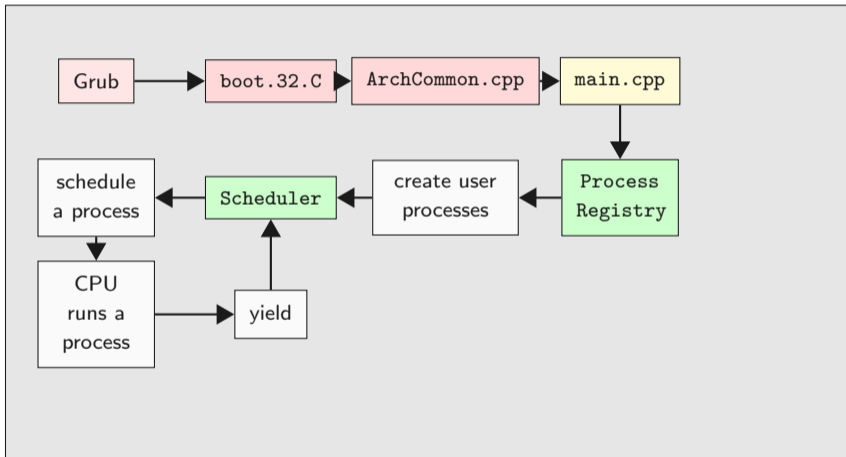
You and SWEB

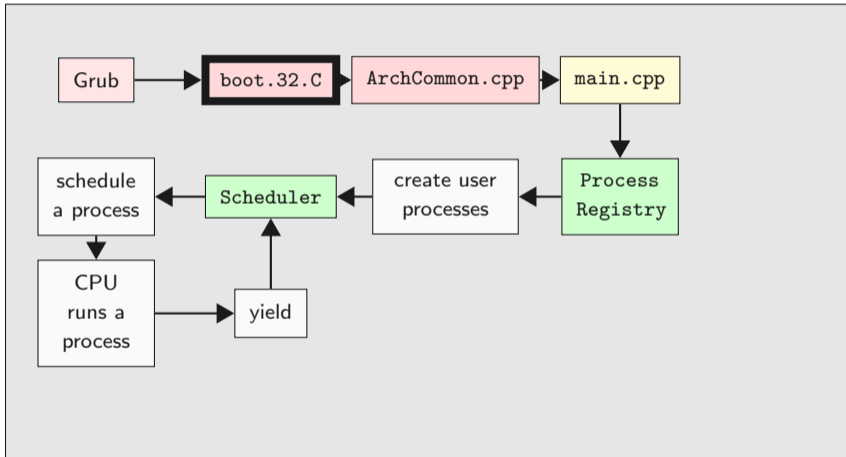
Organizational Details

SWEB

Assignment 1

Booting SWEB

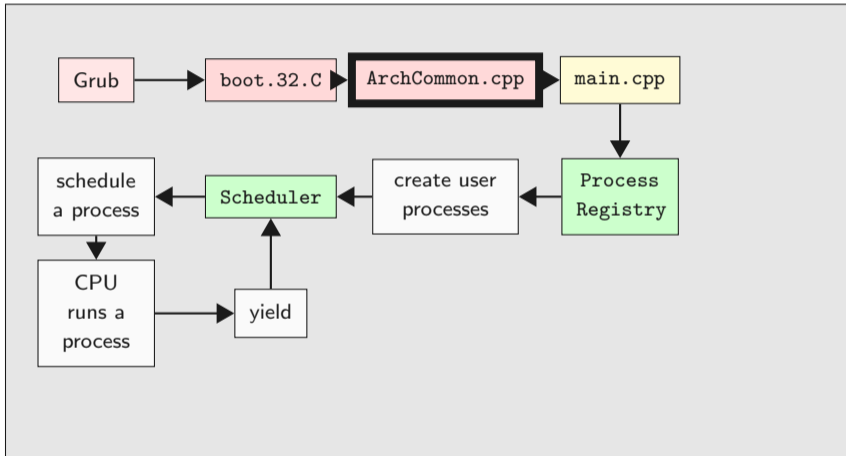




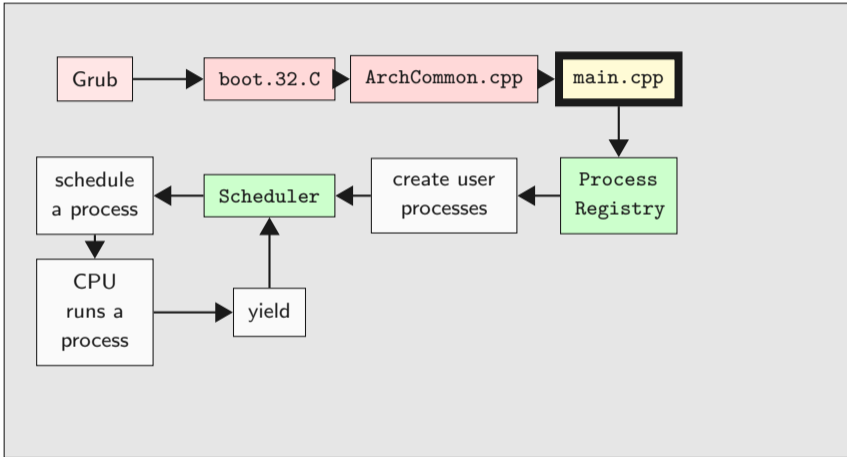
```
extern "C" void entry()
{
    PRINT("Booting...\n");
    // ...
    PRINT("Initialize Kernel Paging Structs\n");
    // ...
    PRINT("Enable Paging...\n");
    // ...
    PRINT("Calling entry64()...\n");
    asm("ljmp %[cs], $entry64-BASE\n" : : [cs]"i"(KERNEL_CS));
}
```



- 32 bit
- Works on physical addresses
- Setup hardware
- Setup paging
- Jump into sane, virtual C world



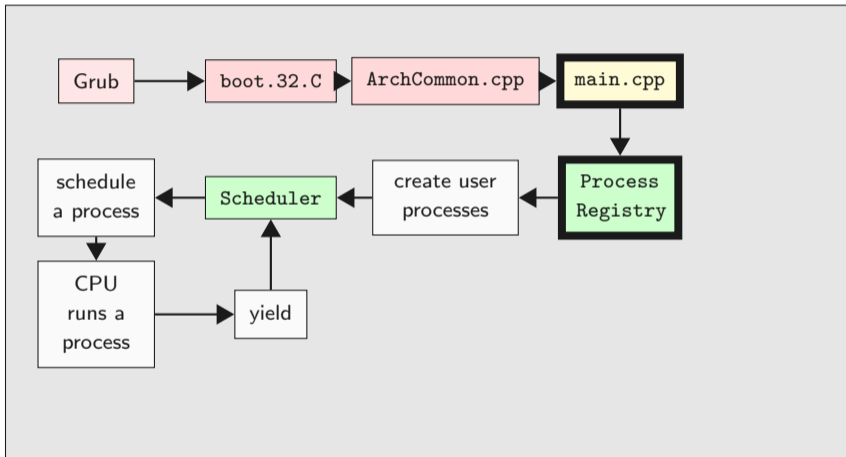
```
extern "C" void entry64()
{
    // ...
    PRINT("Switch to our own stack...\n");
    asm("mov %[stack], %%rsp\n"
        "mov %[stack], %%rbp\n" : : [stack]"i"(boot_stack + 0x4000));
    PRINT("Loading Long Mode Segments...\n");
    // ...
    PRINT("Calling startup()...\n");
    asm("jmp *%[startup]" : : [startup]"r"(startup));
}
```




```
extern "C" void startup() {
    removeBootTimeIdentMapping();
    system_state = BOOTING;
    PageManager::instance();
    writeLine("PageManager and KernelMemoryManager created\n");
    // ...
    Scheduler::instance()->addNewThread(new ProcessRegistry(...));
    // ...
    system_state = RUNNING;
    ArchInterrupts::enableInterrupts();
    Scheduler::instance()->yield();
    //not reached
}
```



- Setup kernel objects
- Setup more hardware
- Enable interrupts
- Handover control to Scheduler

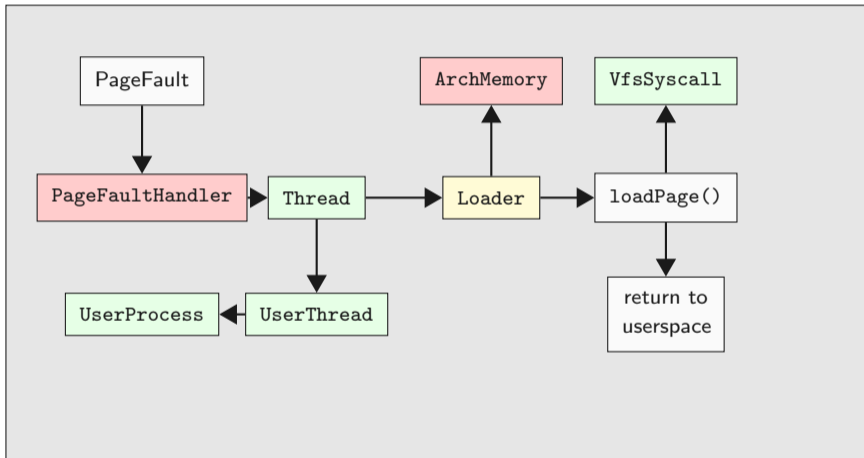


```
void ProcessRegistry::Run()
{
    debug(PROCESS_REG, "mounting userprog-partition \n");
    // ...
    vfs_syscall.mount("idea1", "/usr", "minixfs", 0);
    debug(PROCESS_REG, "mount idea1\n");

    for (size_t i = 0; progs_[i]; i++)
    {
        createProcess(progs_[i]);
    }
}
```



- Kernel Thread
- Mounts hard disk
- Creates user processes
- Sleeps until last user process died, ...
- ... and unmounts the hard disk again



Status quo

- Derived from Thread
- Bad design is easier for you to improve ;)
- Executes binary code
- Has a virtual address space (Loader → ArchMemory)
- Has a userspace part and a kernel part

- Every 54.925439ms
- Implemented in InterruptUtils.cpp

```
extern "C" void irqHandler_0()  
{  
    ArchCommon::drawHeartBeat();  
    Scheduler::instance()->incTicks();  
    Scheduler::instance()->schedule();  
    ArchInterrupts::EndOfInterrupt(0);  
    arch_contextSwitch();  
}
```


- List of threads
- `schedule()` on IRQ0
- `schedule()` on IRQ65 (yield)
- `void Scheduler::addNewThread(Thread *thread);`
- Contains `IdleThread` (hlt if idle)
- Contains `CleanupThread`
 - Calls `delete` on dead threads

A process is scheduled:

- only if `switch_to_userspace_ == 1`
 - Scheduler loads register values from `ArchThreadRegisters` member variable
- Implicitly sets RIP from `ArchThreadRegisters`
- RIP initially points to binary entry point
 - CPU switches to user mode and continues with given RIP



```
userspace/libc/src/nonstd.c
```

```
extern int main();
```

```
void _start()
```

```
{
```

```
    exit(main());
```

```
}
```

libc is in userspace!

```
userspace/tests/helloworld.c
#include <stdio.h>
int main()
{
    puts("hello , world");
    return 0;
}
```

- Off-the-shelf Hello World Program
- Brian Kernigham, 1974

userspace/libc/printf.c

```
int puts(const char *output_string)
{
    // ...
    characters_written = write(STDOUT_FILENO, (void*) output_string,
    string_length);
    // ...
}
```

- In libc again
- Sanity checks
- Actually only a wrapper for syscall write



```
userspace/libc/write.c
ssize_t write(int file_descriptor, const void *buffer, size_t count)
{
    return __syscall(sc_write, file_descriptor,
                    (long) buffer, count, 0x00, 0x00);
}
```

- POSIX!
- fwrite is a simple write wrapper
- Down the rabbit hole: __syscall

```
arch/x86/64/common/userspace/syscalls.c
size_t __syscall(size_t arg1, size_t arg2, size_t arg3,
                 size_t arg4, size_t arg5, size_t arg6)
{
    asm("int $0x80\n" : "=a"(arg1)
        : "a"(arg1), "b"(arg2), "c"(arg3),
          "d"(arg4), "S"(arg5), "D"(arg6));
    return arg1;
}
```

- Copy arguments to registers
- Issue interrupt 0x80

→ switch to kernel space

```
arch/x86/64/common/source/arch_interrupts.S
```

```
arch_syscallHandler:
```

```
    pushall
```

```
    movq %rsp,%rdi
```

```
    movq $0,%rsi
```

```
    call arch_saveThreadRegisters
```

```
    call syscallHandler
```


InterruptUtils.cpp

```
extern "C" void syscallHandler() {
    thread->switch_to_userspace_ = 0;
    threadRegisters = thread->kernel_regs_;
    ArchInterrupts::enableInterrupts();

    auto ret = Syscall::syscallException(
        thread->user_regs_->rax,
        thread->user_regs_->rbx,
        thread->user_regs_->rcx,
        thread->user_regs_->rdx,
        thread->user_regs_->rsi,
        thread->user_regs_->rdi);
}
```

Syscall.cpp

```
size_t Syscall::syscallException(size_t syscall_number, size_t arg1,
    size_t arg2, size_t arg3, size_t arg4, size_t arg5) {
    switch (syscall_number)
    {
        // ...
        case sc_write:
            return_value = write(arg1, arg2, arg3);
            break;
        // ...
    }
}
```

syscallException only calls the right methods with the right number of parameters

- `sc_write` constant defined in `Syscall.h`
- `Syscall::write` method is regular in-kernel C++

```
size_t Syscall::write(size_t fd, pointer buffer, size_t size)
{
    if (fd == fd_stdout) //stdout
    {
        debug(SYSCALL, "Syscall::write: %.*s\n", (int)size, (char*) buffer);
        kprintf("%.*s", (int)size, (char*) buffer);
    }
    // ...
}
```

```
common/include/console/debug.h
const size_t MAIN          =Ansi_Red      | ENABLED;
const size_t THREAD       =Ansi_Magenta | ENABLED;
const size_t USERPROCESS =Ansi_Cyan     | ENABLED;
const size_t PROCESS_REG  =Ansi_Yellow  | ENABLED;
const size_t BACKTRACE    =Ansi_Red     | ENABLED;
const size_t USERTRACE    =Ansi_Red     | ENABLED;
//group memory management
const size_t PM           =Ansi_Green   | ENABLED;
const size_t KMM          =Ansi_Yellow;
```

Add your own debug tags!

“This is not the printf you are looking for”



- `kprintf` → SWEB Terminal
 - `kprintfd` → Host
 - Cannot be disabled by flags
- Prefer debug

```
#include <stdio.h>
int main()
{
    puts("hello, world");
    return 0;
}
```

What does return 0; do?

```
userspace/libc/src/nonstd.c
```

```
extern int main();
```

```
void _start()
```

```
{
```

```
    exit(main());
```

```
}
```


AANNND



IT'S GONE

```
size_t Syscall::syscallException(...) {  
    // ...  
    case sc_exit:  
        exit(arg1);  
        break;  
    // ...  
}  
void Syscall::exit(size_t exit_code) {  
    debug(SYSCALL, "Syscall::EXIT: called, exit_code: %zd\n", exit_code)  
    ;  
    currentThread->kill();  
}
```

```
void Thread::kill() {  
    switch_to_userspace_ = 0;  
    state_ = ToBeDestroyed;  
    if (currentThread == this) {  
        ArchInterrupts::enableInterrupts();  
        Scheduler::instance()->yield();  
    }  
}
```

Recall: Scheduler will call delete on thread if `state_ == ToBeDestroyed`

```
UserProcess::~~UserProcess() {
    delete loader_;
    vfs_syscall.close(fd_);
    delete working_dir_;
    process_registry_ ->processExit();
}
Thread::~~Thread() {
    delete user_registers_;
    delete kernel_registers_;
}
```




Get in touch with the source code!