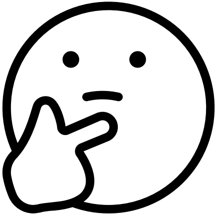
Operating Systems

Assignment 2

Daniel Gruss

2023-11-29

Topics



Presented Today:

- **Mandatory:** Virtual Memory (Copy On Write, Swapping)
- Shared Memory
- Memory Mapped I/O
- Dynamic Memory in the userspace

Other Topics:

- You can do basically anything OS related
- Just ask your Tutor how many points it brings

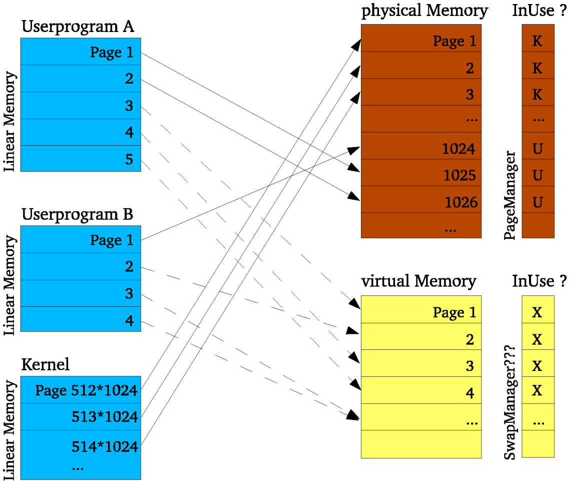
Page Replacement

Page Replacement

- Swap pages to the swap device (from RAM to HDD)
- Don’t forget to lock shared resources!

What does the OS need to know?

- Where is the swap device located?
- Where to find free space within the swap device?
- Has a page been swapped out, or is it within RAM?
- Where has a page been swapped to (target address)?



Userprogram A

Linear Memory

Page 1
2
3
4
5

Userprogram B

Linear Memory

Page 1
2
3
4

Kernel

Linear Memory

Page 512*1024
513*1024
514*1024
...

physical Memory

Page 1
2
3
...
1024
1025
1026
...

InUse ?

PageManager

K
K
K
...
U
U
U

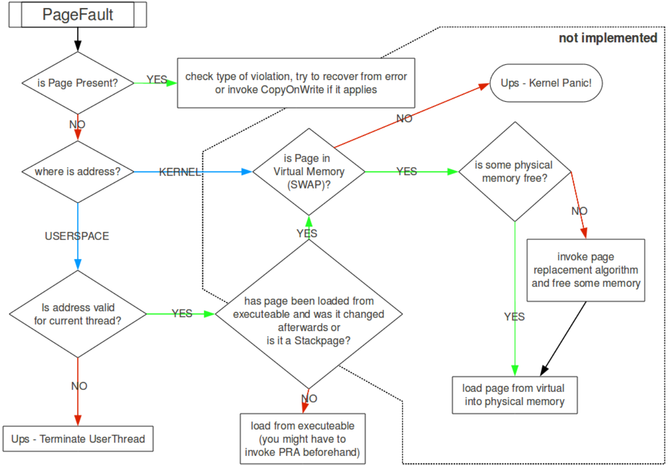
virtual Memory

Page 1
2
3
4
...

InUse ?

SwapManager???

X
X
X
X
...



PageFault

is Page Present?

YES

check type of violation, try to recover from error or invoke CopyOnWrite if it applies

NO

where is address?

KERNEL

is Page in Virtual Memory (SWAP)?

NO

Ups - Kernel Panic!

YES

is some physical memory free?

NO

invoke page replacement algorithm and free some memory

YES

load page from virtual into physical memory

USERSPACE

Is address valid for current thread?

YES

has page been loaded from executeable and was it changed afterwards or is it a Stackpage?

YES

NO

load from executeable (you might have to invoke PRA beforehand)

NO

Ups - Terminate UserThread

not implemented

Access the HDD in SWEB

- Virtual Memory is located at the third partition of the first HDD
    - (BD device number 3)
- Responsible Code: `arch_bd_*`

Example (Write to BD (Pseudocode))

```
size_t block = target block number;
pointer page_data = pointer to source data;

BDVirtualDevice* bd_device = BDManager::getInstance()->getDeviceByNumber(3);
bd_device->writeData(block*bd_device->getBlockSize(), PAGE_SIZE, page_data);
```

Do not use `BDRequest` directly unless you asked a Tutor!

Using `BDRequest` directly is unsafe!

Which pages are swappable?

- User space pages (where does it make sense?)
- Mark PTs/PDs/PDPTs as non-present and swapped out
- Kernel pages (has not been done before)

Page Table Usage (x86_64)

present == 0: entry invalid, all bits ignored by MMU
→ pagefault on access

writeable == 0: write protected

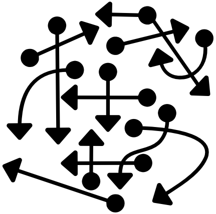
accessed, dirty == 1: has been accessed/modified

ignored_x : unused bits

page_ppn : physical page number

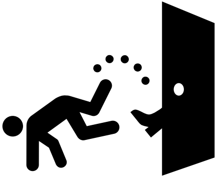
```
typedef struct
{
  uint64 present           :1;
  uint64 writeable         :1;
  uint64 user_access       :1;
  uint64 write_through     :1;
  uint64 cache_disabled    :1;
  uint64 accessed          :1;
  uint64 dirty             :1;
  uint64 size              :1;
  uint64 global            :1;
  uint64 ignored_2         :3;
  uint64 page_ppn          :28;
  uint64 reserved_1        :12;
  uint64 ignored_1         :11;
  uint64 execution_disabled :1;
} PageTableEntry;
```

Testing / How to use lots of RAM



- Use tests which use big arrays
  (e.g. size_t array[BIG_NUMBER];)
- Test **all** swapping-situations
- ..., without running into the limits of the kernel heap.
- Free memory can exhaust soon (even with a good PRA)!

Page Replacement Algorithms



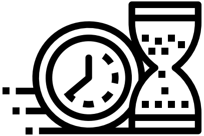
What does a PRA do?

- Searches for pages that have not been used for a while
- Runs if memory is needed or there is nothing to be done
- But not every time...

Which PRA?

- Recommended: Aging or WSRandom
- Create your own PRA (why is it better than other PRAs?)
- Bonus Points: User can switch PRAs

Sytem Time



Some PRAs need time information Where to get them from?

- Ticks, TSC, RTC
- Recycle parts of the `sleep`- or `clock`-implementation
- Derive the time from the tick sources
- Hint: `InterruptUtils.cpp`

Inverted Page Table (IPT)

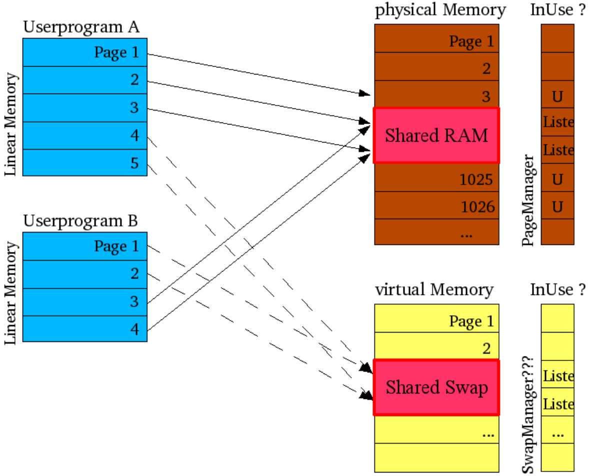
Shared Pages

- Pages may be used by several processes
- Aka: Page table entries of different user spaces point to the same physical page

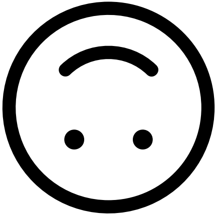
Where can pages be shared?

- RAM
- Swap
- Binary

And what's with copy-on-write?


Userprogram A
Linear Memory
Page 1
2
3
4
5
Userprogram B
Linear Memory
Page 1
2
3
4
physical Memory
Page 1
2
3
Shared RAM
1025
1026
...
InUse ?
U
Liste
Liste
U
U
PageManager
virtual Memory
Page 1
2
Shared Swap
...
InUse ?
Liste
Liste
...
SwapManager???

Virtual Memory and Shared Pages



Inverted Page Table connects a physical page/swapped page to all virtual page usages

Only one process uses the page:

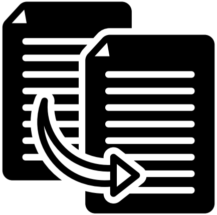
- Swap out page + inform process
- Which process owns the page?

Shared Pages (several processes use the same page):

- Swap out + inform all processes
- Which process**es** own the page?
- But what if a process terminates?

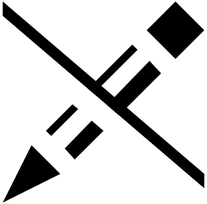
Copy On Write

Copy on Write

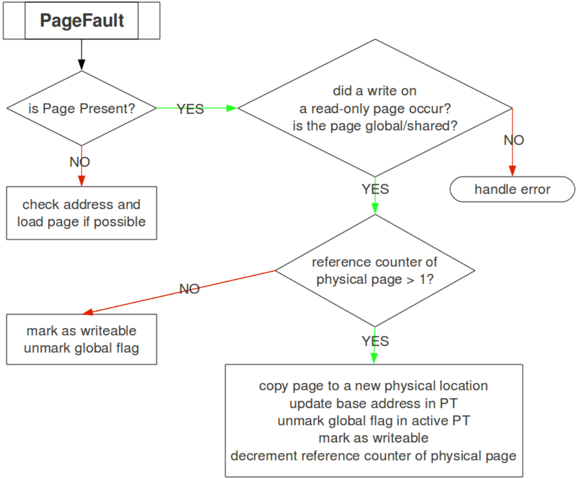


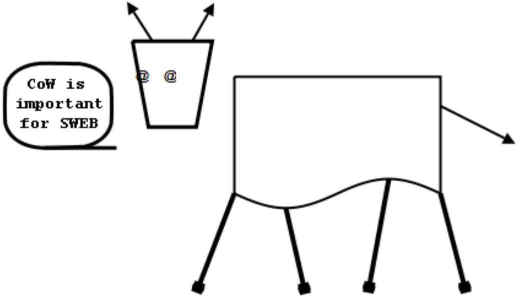
- Usage of `fork`():
  1. `fork`() clones a process (copy, copy, copy...)
  2. The child process often uses `exec`(...) after `fork`()
  3. There has been much useless deep copying and deleting
- Do we really have to copy all the stuff?
  - Both processes use the same physical and swapped pages
  - Two (or more) processes have the same pages in RAM/Swap Device
  - Works as long as no one is writing onto them

Read-Only Pages and Pagefaults

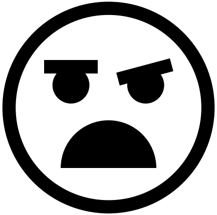


- How do we realize that someone wants to write onto a page
  - Usually we can't
  - The writeable-flag has to be zero
- Process tries to write onto a read-only page
  → PageFault
- What now?
  - → Is it a shared page?
  - → Copy page and link to the new one
  - → If only one process is left → no shared pages!

PageFault
is Page Present?
YES
NO
check address and load page if possible
did a write on a read-only page occur? is the page global/shared?
NO
handle error
YES
reference counter of physical page > 1?
NO
mark as writeable unmark global flag
YES
copy page to a new physical location
update base address in PT
unmark global flag in active PT
mark as writeable
decrement reference counter of physical page

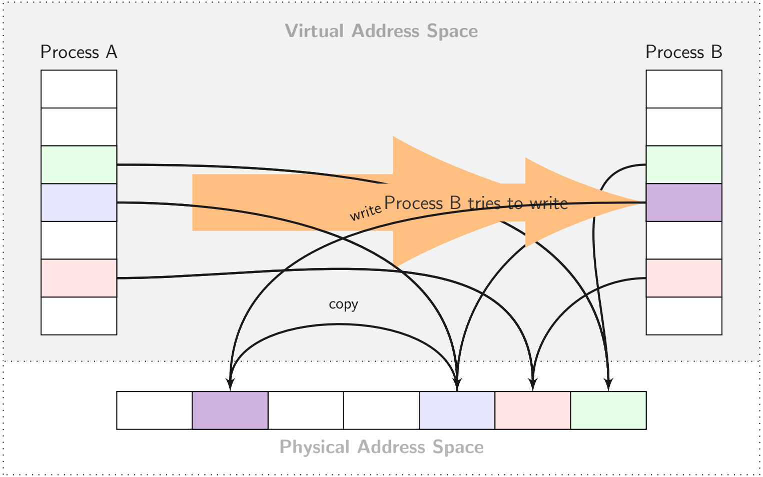

CoW is important for SWEB
@ @

What about the global flag?



- Don’t use it!
- “global” means “keep over next context switch”
- This is not what you want
- Will cause almost untraceable bugs!
- Use and rename an unused bit as “shared” flag instead

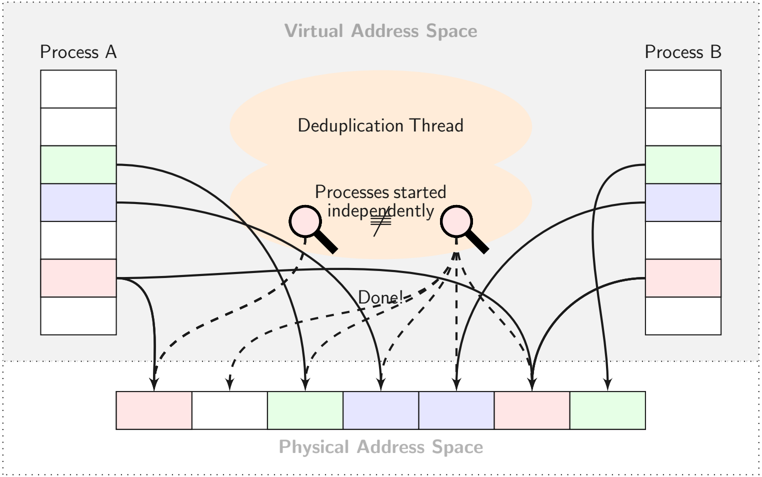
Copy-on-Write



Copy On Write Without Fork

- Starting /`usr`/`shell.sweb` twice, without fork
- Loading the same image in different programs
- Generating the same data in different programs
- → Page Deduplication

Page Deduplication



Additional Tasks

Shared Memory

1. Process A wants to share 3 pages with process B
2. Process A syscall: get 3 pages of shared memory ID 4
3. Kernel: maps 3 virtual pages (10-12) of A to physical pages 464, 9078, and 123
4. Process B syscall: get 3 pages of shared memory ID 4
5. Kernel: maps 3 virtual pages (22-24) of A to physical pages 464, 9078, and 123
6. → A and B now share 3 pages

Shared Memory: How To Implement

- Syscalls:

```
int shm_open(const char *name, int oflag, mode_t mode);
int shm_unlink(const char *name);
void *mmap(void *addr, size_t len, int prot, int flags, int fildes,
off_t off);
int munmap(void *addr, size_t len);
```

- Manages IDs (pseudo file-descriptor) and users of the shared regions
- `munmap` and `close` when the process ends or manually
- No reference to the shared memory object → destroy it

Memory Mapped File I/O

- Files are not accessed by using (`open`/`creat`/`close`/`read`/`write`) any longer, they are directy mapped into the address space
- Parts of the mapped file are copied into RAM on demand!
- They are written back when being unmapped (if they have been modified)
    - Depends on the flags set when being mapped
- If several processes have the same file mapped → Shared Memory

mmap / munmap

- `void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);`
- `int munmap(void *addr, size_t len);`

fildes
- `shm_open` or `open`
- Which processes opened the same file?

len
- Only multiples of `PAGE_SIZE`
- File size usually not `PAGE_SIZE`-aligned

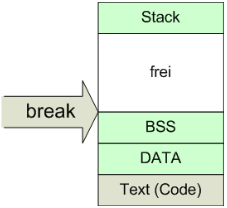
protection : Access rights for the mapped areas
- `PROT_READ`: How to prevent write accesses?
- `PROT_WRITE`: flags relevant!

flags :
- `MAP_PRIVATE`:
  - Copy-on-write
  - No write-back

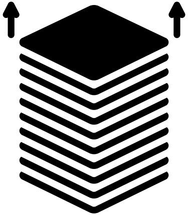
Userspace Dynamic Memory

- Memory allocation at runtime
- Implement malloc/free

Address space of a process:



How to use brk/sbrk



- `int brk(void *end_data_segment);`
- `void *sbrk(int increment);`
- Linker symbol `_end`

Example (sbrk/break in userspace)

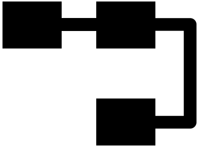
```
extern _end;
// ...
size_t heap_start = &_end;
size_t heap_end = heap_start + 4096;
if ( brk(heap_end) == 0)
{
  //do stuff in dynamic memory
}
```

Userspace Memory Management

`brk` and `sbrk` are complicated to use - let's implement:

- `malloc(size_t size)`/`free(void *p)` in libc
- Manages the allocated memory regions
  - Requests pages from the kernel
  - Frees unused pages again
  - therefore uses `brk()`/`sbrk()`

Userspace Memory Management



- simple implementation:
  doubly-linked list containing the memory regions
- Don’t forget about locking!

Design / Submissions

Design

Proof-of-Concept-Implementation as in Assignment 1

Recommendation: Start with swapping

Alternative tasks

- Normal way: mandatory task virtual memory
- You want to go the normal way? Just ignore this slide...
- Alternative: Discuss with me about substituting the mandatory task with either security or driver development as your new mandatory task
- This is not possible without discussing it with me!

Submission

- As in Assignment 1
- Tags:
  - Design/Proof-of-Concept: `SubmissionD2`
  - Implementation: `SubmissionI2`

Deadlines

- Design-PoC: 15.12.2023, 18:00
  - Individual feedback meetings ideally between 18.-20.12.
- Implementation: 19.01.2024, 18:00
- Since 2011 we went to a pub after the implementation deadline

Student Debates on A2 Designs

- In two weeks (04.-07.12.)
- Like the one from Assignment 1
- Compulsory attendance
- Bring 2 pieces of paper with your name
- Repeating the assignment specification is not enough!
- Your design should be complete by that time
- Instant feedback

Evaluations

- Tell us what was good and should remain the same
- Tell us what was bad and should be changed