Operating Systems

Assignment 1

Daniel Gruss

October 12, 2020

www.iaik.tugraz.at

Table of contents

Memory Management in SWEB

Virtual memory layout (x86-64)

Every process has its own virtual address space (256 TB)

- Userspace at 0 GiB (size: 128 TB)
- Kernelspace (kernel, video memory) at -2 GB (size: 1 GB)
  - -2 GB → `0xFFFF FFFF 8000 0000`
- Identity mapping at -16 TB (size: 1 GB)
  - -16 TB → `0xFFFF F000 0000 0000`

Page Map Level 4 (PML4)

- Every Process has a PML4
- Kernel has a PML4

Page Map Level 4 (PML4)

- Every Process has a PML4
- Kernel has a PML4
- Maps virtual to physical address space
- Syscall: Jump to kernel, but PML4 stays the same

Page Map Level 4 (PML4)

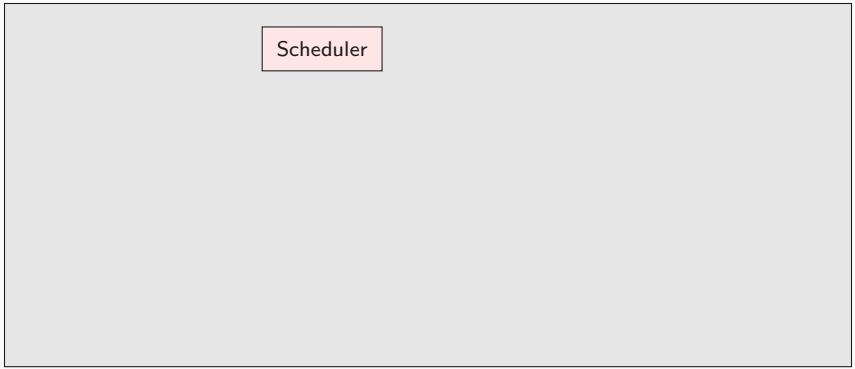
- Every Process has a PML4
- Kernel has a PML4
- Maps virtual to physical address space
- Syscall: Jump to kernel, but PML4 stays the same
- Identitiy Mapping for physical access

Page Map Level 4 (PML4)

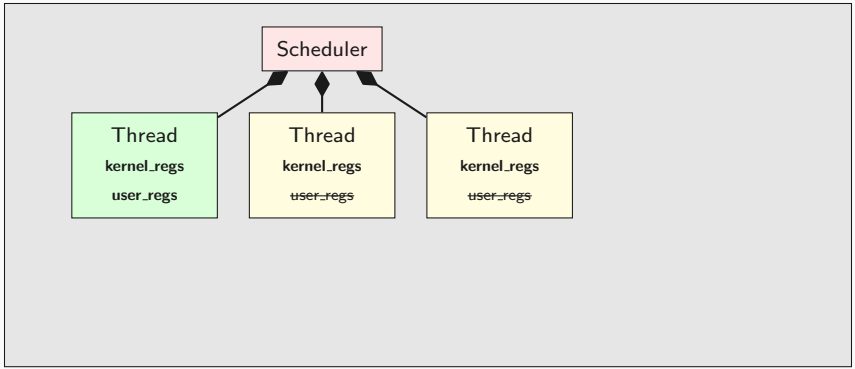
- Every Process has a PML4
- Kernel has a PML4
- Maps virtual to physical address space
- Syscall: Jump to kernel, but PML4 stays the same
- Identitiy Mapping for physical access

Processes and Threads

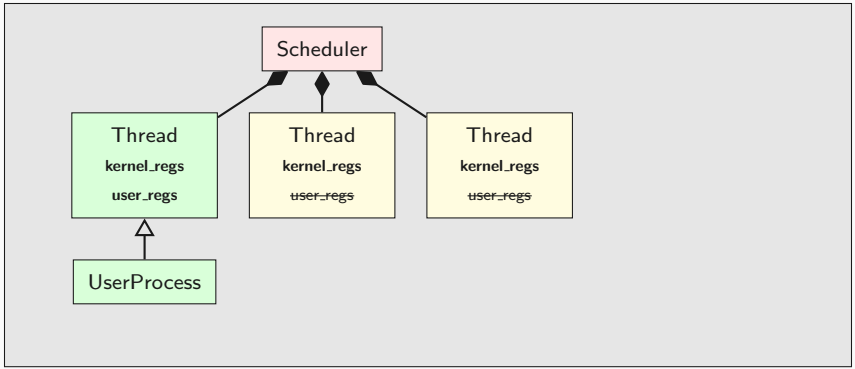
UserProcess Overview



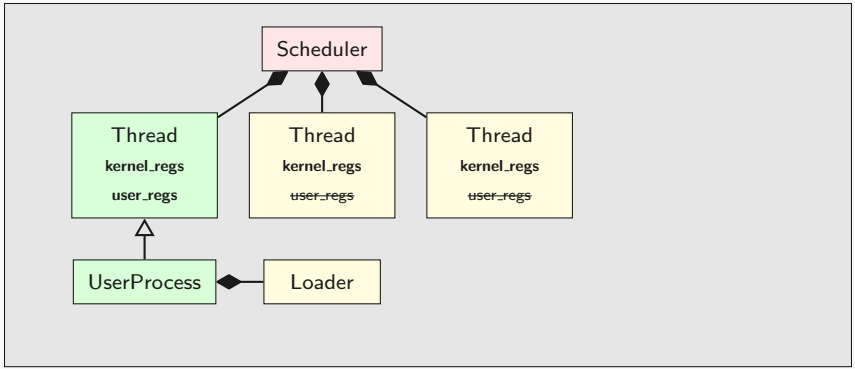
UserProcess Overview



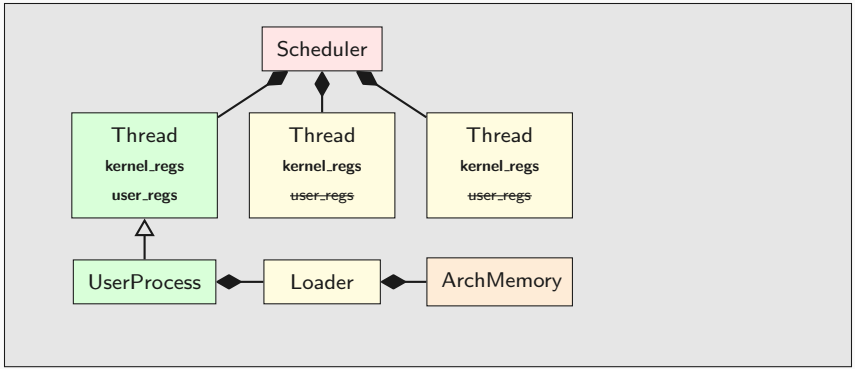
UserProcess Overview



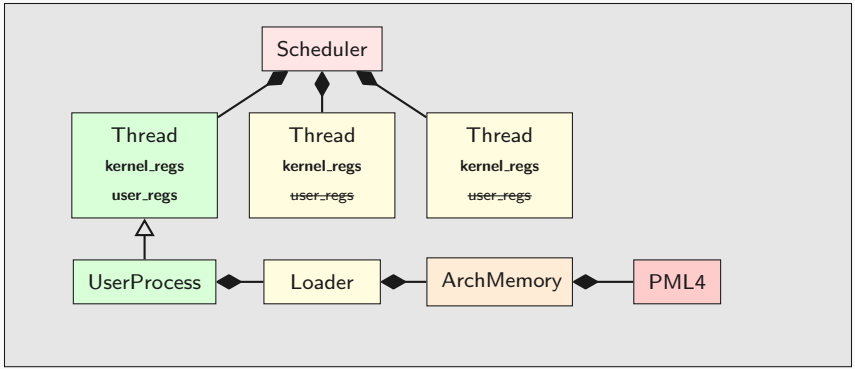
UserProcess Overview



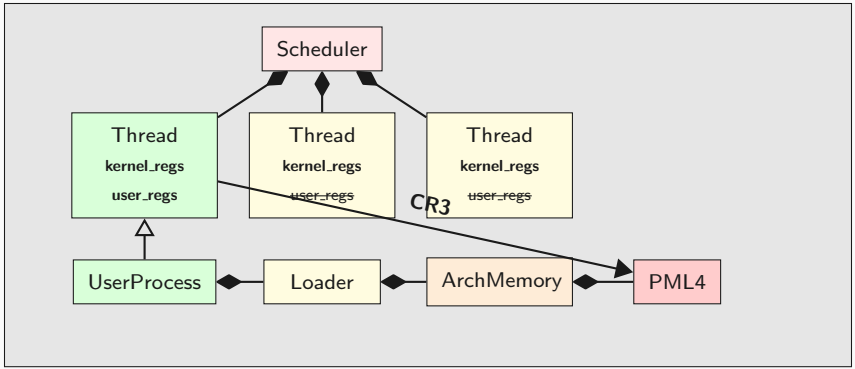
UserProcess Overview



UserProcess Overview



UserProcess Overview



Threads

- Scheduleable base unit
- Don't schedule processes!

Threads

- Scheduleable base unit
- Don't schedule processes!
- Thread status:
  - Running, Sleeping, ToBeDestroyed
  - Enough? ($\rightarrow$ Syscall sleep?)

- Scheduleable base unit
- Don't schedule processes!
- Thread status:
  - Running, Sleeping, ToBeDestroyed
  - Enough? ($\rightarrow$ Syscall sleep?)
- UserThreads need to be derived from Thread
- UserProcess should not be derived from Thread

Creating a User Process in SWEB

- `userspace/tests/example.c`
- automatically on VMDK: `/usr/example.sweb`
- start via shell,

- `userspace/tests/example.c`
- automatically on VMDK: `/usr/example.sweb`
- start via shell, or add to autostart:
  - autostart `user_progs[]` in `user_progs.h`

Creating a User Process in SWEB

- `userspace/tests/example.c`
- automatically on VMDK: `/usr/example.sweb`
- start via shell, or add to autostart:
  - autostart `user_progs[]` in `user_progs.h`
- prepare build: `mkdir -p /tmp/sweb; cd /tmp/sweb; cmake /path/to/sourcecode/of/sweb`
- build: `make -j`

Function Calls (cdecl)

How do function calls work?

C Program

```c
size_t add(size_t a, size_t b) {
  return a + b;
}

int main() {
  size_t a = 7;
  size_t b = 14;
  size_t c = 0;

  c = add(a, b);

  return c;
}
```

Assembler Program (gcc -S -m64)

```
main:
>pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	<rbp>
%rsp	0xfff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
>pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rsp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	<rbp>
%rsp	0xfff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
>movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rsp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	<rbp>
%rsp	0xfff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
>movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rsp, %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xfff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
>subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rsp, %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xfff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
>subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
>movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
>movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
>movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
>movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
>movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
>movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
>movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
>movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	14
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
>movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	??
%rdx	14
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
>movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
>movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	??
%rsi	??

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
>movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	??
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
>movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	??
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
>movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
>call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
>call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0		
0xffb8		
0xffb0		

%rbp	0xfff0
%rsp	0xffc8
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
>pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0		
0xffb8		
0xffb0		

%rbp	0xfff0
%rsp	0xffc8
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
>pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp
0xffb8		
0xffb0		

%rbp	0xfff0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
>movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp
0xffb8	
0xffb0	

%rbp	0xfff0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
>movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	
0xffb0	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)


```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
>movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	
0xffb0	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
>movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	7
0xffb0	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
>movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	7
0xffb0	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
>movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
>movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
>movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
>movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
>movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	14
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
>addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	14
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
>addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
>popq  %rbp
 ret
```

0xfff0	<rbp>	
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xfff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc0
%rsp	0xffc0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
>popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13> ← %rsp
0xffc0	0xfff0
0xffb8	7
0xffb0	14

%rbp	0xfff0
%rsp	0xffc8
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
>ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0	0xfff0	
0xffb8	7	
0xffb0	14	

%rbp	0xfff0
%rsp	0xffc8
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
>ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xfff0	
0xffb8	7	
0xffb0	14	

%rbp	0xfff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
>movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xfff0	
0xffb8	7	
0xffb0	14	

%rbp	0xfff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
>movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp>	← %rbp
0xffe8	21	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xfff0	
0xffb8	7	
0xffb0	14	

%rbp	0xfff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
>movq  -8(%rbp), %rax
 leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	<main+13>
0xffc0	0xfff0
0xffb8	7
0xffb0	14

%rbp	0xfff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
>leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	<main+13>
0xffc0	0xfff0
0xffb8	7
0xffb0	14

%rbp	0xfff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
>leave
 ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp,%rsp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0
0xffb8	7
0xffb0	14

%rbp	0xfff0
%rsp	0xfff0
%rax	21
%rdx	7
%rdi	7
%rsi	14

Assembler Program (gcc -S -m64)

```
main:
 pushq %rbp
 movq  %rsp, %rbp
 subq  $32, %rsp
 movq  $7, -24(%rbp)
 movq  $14, -16(%rbp)
 movq  $0, -8(%rbp)
 movq  -16(%rbp), %rdx
 movq  -24(%rbp), %rax
 movq  %rdx, %rsi
 movq  %rax, %rdi
 call  add
 movq  %rax, -8(%rbp)
 movq  -8(%rbp), %rax
 leave
>ret
```

```
add:
 pushq %rbp
 movq  %rsp, %rbp
 movq  %rdi, -8(%rbp)
 movq  %rsi, -16(%rbp)
 movq  -8(%rbp), %rdx
 movq  -16(%rbp), %rax
 addq  %rdx, %rax
 popq  %rbp
 ret
```

0xfff0	<rbp> ← %rbp,%rsp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xfff0
0xffb8	7
0xffb0	14

%rbp	0xfff0
%rsp	0xfff0
%rax	21
%rdx	7
%rdi	7
%rsi	14

System Calls in the Kernel

- Userspace: `__syscall(...)`
- Move arguments and syscall number into the registers (`rax`,`rbx`,...)
- Syscall numbers defined in:
  `common/include/kernel/syscall-definitions.h`
- Jump into kernel mode: IRQ 0x80

- Userspace: `__syscall(...)`
- Move arguments and syscall number into the registers (`rax`,`rbx`,...)
- Syscall numbers defined in:
  `common/include/kernel/syscall-definitions.h`
- Jump into kernel mode: IRQ 0x80
- Kernel low-level: Interrupt dispatching
- Kernel high-level: `Syscall::syscallException(...)`

- Processes and threads share resources
- Will be destroyed without locking

→ Synchronization necessary

- Why “critical section” is a good term?

- Why “critical section” is a good term?
- Why “critical section” is a bad term?

- Why “critical section” is a good term?
- Why “critical section” is a bad term?
- Identify accesses to a shared resource
- Protect all accesses using the same mechanism

- Can we just disable interrupts?

- Can we just disable interrupts? **No!**

- Can we just disable interrupts? **No!**
- Spinlock
- Mutex
- Semaphore
- Condition Variables

```
// return 0 if locking was successful
size_t lock(size_t* lock) {
  if (*lock == 0) // not locked
  {
    *lock = 1; // now locked
    return 0;
  }
  return 1;
}
```

POSIX: 0 means success!

```
size_t lock(size_t* lock) {
  if (*lock == 0) // not locked
  {
    *lock = 1; // now locked
    return 0;
  }
  return 1;
}
```

Any problems here?

```
size_t lock(size_t* lock) {
  size_t old_val = 1;
  asm("xchg %0,%1"
      : "=r" (old_val)
      : "m" (*lock), "0" (old_val)
      : "memory");
  return old_val;
}
```

Lock should spin until successful!

```
size_t lock(size_t* lock) {
  size_t old_val = 1;
  do
  {
    asm("xchg %0,%1"
        : "=r" (old_val)
        : "m" (*lock), "0" (old_val)
        : "memory");
  } while (old_val);
  return old_val;
}
```

Upon return: `*lock == 1` and return value is `0`

```
size_t lock(size_t* lock) {
  size_t old_val = 1;
  do
  {
    asm("xchg %0,%1"
        : "=r" (old_val)
        : "m" (*lock), "0" (old_val)
        : "memory");
  } while(old_val && !sched_yield());
  return old_val;
}
```

Single core: Yield instead of busy wait

- Use spinlock to protect:

Mutex

- Use spinlock to protect:
  - Mutex lock variable

- Use spinlock to protect:
  - Mutex lock variable
  - “Held by” thread pointer

- Use spinlock to protect:
  - Mutex lock variable
  - “Held by” thread pointer
  - List of threads (Sleepers List)
    - Want to acquire the Mutex
    - Wait for Mutex to be free

- Use spinlock to protect:

- Use spinlock to protect:
  - Counter variable

Semaphore

- Use spinlock to protect:
  - Counter variable
  - **No** “held by” thread pointer

- Use spinlock to protect:
  - Counter variable
  - **No** "held by" thread pointer
  - List of threads (Sleepers List)

Semaphore

- Use spinlock to protect:
  - Counter variable
  - **No** "held by" thread pointer
  - List of threads (Sleepers List)
- Increase/Decrease counter
- Counter is 0 $\rightarrow$ block $\rightarrow$ sleepers list

- A sleepers list

- A sleepers list
- Needs a Mutex to protect the sleeper list

- A sleepers list
- Needs a Mutex to protect the sleeper list
- “Wait” on a CV
- “Signal” wakes up 1 thread
- “Broadcast” wakes up all threads

Design Decisions

- Every process has its own virtual address space
- Multiple threads per process (number only limited by hardware)
- Managed by the kernel (easier than user space managed)

- What happens when the first thread finishes?

- What happens when the first thread finishes?
- When to clean-up user space and loader?

- What happens when the first thread finishes?
- When to clean-up user space and loader?
- How to detect that a function running as a thread returns?

- What happens when the first thread finishes?
- When to clean-up user space and loader?
- How to detect that a function running as a thread returns?
- Where shall the stacks be placed?

- What happens when the first thread finishes?
- When to clean-up user space and loader?
- How to detect that a function running as a thread returns?
- Where shall the stacks be placed?
- What happens if the process calls `fork/exec`?

- One thread per `UserProcess`
- We want multi threading
- Threads share page dir, `Loader`, etc.

- One thread per `UserProcess`
- We want multi threading
- Threads share page dir, `Loader`, etc.
- Do we need a `UserThread`?

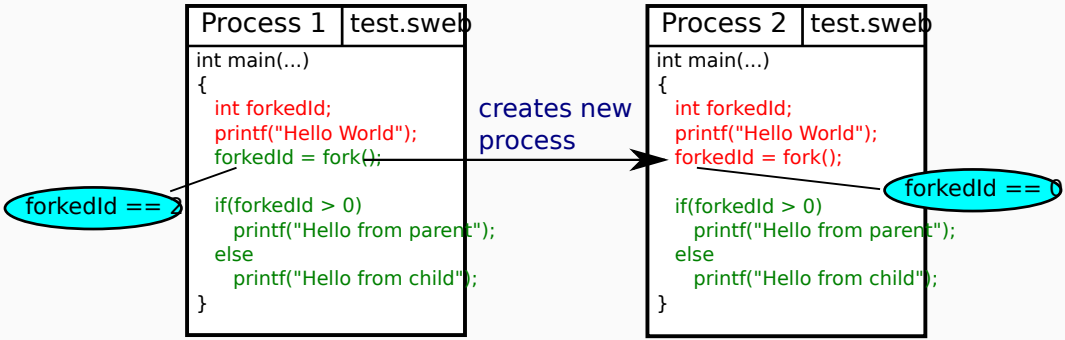
- POSIX Thread Library
- `man pthreads` or search for “opengroup pthreads”
- At least:

- POSIX Thread Library
- `man pthreads` or search for “opengroup pthreads”
- At least:
  - `pthread_create`

- POSIX Thread Library
- `man pthreads` or search for “opengroup pthreads”
- At least:
  - `pthread_create`
  - `pthread_exit`

- POSIX Thread Library
- `man pthreads` or search for “opengroup pthreads”
- At least:
  - `pthread_create`
  - `pthread_exit`
  - `pthread_cancel`

- POSIX Thread Library
- `man pthreads` or search for “opengroup pthreads”
- At least:
  - `pthread_create`
  - `pthread_exit`
  - `pthread_cancel`
  - `pthread_join`
- Additional syscalls as you like


Process 1
test.sweb
int main(...)
{
int forkedId;
printf("Hello World");
forkedId = fork();
if(forkedId > 0)
printf("Hello from parent");
else
printf("Hello from child");
}
creates new process
forkedId == 2
Process 2
test.sweb
int main(...)
{
int forkedId;
printf("Hello World");
forkedId = fork();
if(forkedId > 0)
printf("Hello from parent");
else
printf("Hello from child");
}
forkedId == 0

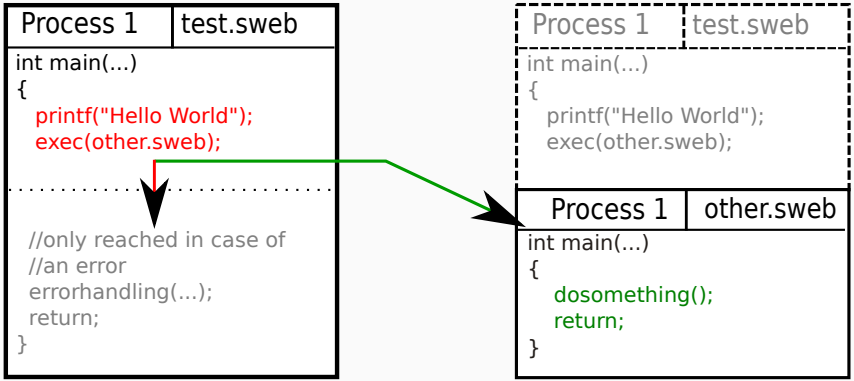
Fork Design Decisions

- What happens if the process has several threads?

- What happens if the process has several threads?
- What happens with the used ressources? (files, mutexes, ...)?

- What happens if the process has several threads?
- What happens with the used ressources? (files, mutexes, ...)?
- How to ensure that the copy procedure is “atomic”?

- What happens if the process has several threads?
- What happens with the used ressources? (files, mutexes, ...)?
- How to ensure that the copy procedure is “atomic”?
- Advanced: Is it possible that processes share pages (especially code)?


Process 1
test.sweb
int main(...)
{
printf("Hello World");
exec(other.sweb);
//only reached in case of
//an error
errorhandling(...);
return;
}
Process 1
test.sweb
int main(...)
{
printf("Hello World");
exec(other.sweb);
Process 1
other.sweb
int main(...)
{
dosomething();
return;
}

Execv Example

```
// int execv(const char *path, char *const argv[]);

int child_status;

int pid = fork();

if(pid == 0) { // child process
    execv("programm.sweb",0);
} else if(pid > 0) { // parent process
    waitpid(pid, &child_status, WEXITED);
```

Execv Design Decisions

- `argv` should be handed over to the `main`-function of the new process

Execv Design Decisions

- `argv` should be handed over to the `main`-function of the new process
- Where shall the parameters be placed at?

Execv Design Decisions

- `argv` should be handed over to the `main`-function of the new process
- Where shall the parameters be placed at?
- How to give them to `main`?

Waitpid

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- Wait until the child process with the given PID terminates
- Requirement for a working shell
- The return value of the process is stored at the status address

Sleep

```
unsigned sleep(unsigned seconds);
```

- The calling thread is put to sleep for a specified time
- The time is given in seconds
- The time shall be as precise as possible (!)
- “yield once” is not enough

```
unsigned sleep(unsigned seconds);
```

- The calling thread is put to sleep for a specified time
- The time is given in seconds
- The time shall be as precise as possible (!)
- “yield once” is not enough
- Can be realized using timestamp counter (or real time clock)
- Optional: implement `usleep` too

Clock

```
clock_t clock(void);
```

- Returns the CPU time of a process

→ Total amount of time it was scheduled

- Time in clocks (see man-page)
- As precise as possible

Clock

```
clock_t clock(void);
```

- Returns the CPU time of a process

→ Total amount of time it was scheduled

- Time in clocks (see man-page)
- As precise as possible

→ Not possible without timestamp counter

- Global file descriptors, shared memory, etc.
- Protect it: local file descriptors, etc.

```
int pipe(int filedes[2]);
```

- Generates two file descriptors
- The data written to `fildes[1]` can be read out from `fildes[0]`
- Used for shell I/O redirection (and several other things)
- Related: How can processes share file descriptors (if they want to)? (!)

- Controlled shutdown of the system
- Terminate all user processes
- Unmount Minix file system
- Turn off the PC (ACPI)

- Spinlocks in userspace (1 point)
  - no points if using the gnu-build-in atomic functions

- Spinlocks in userspace (1 point)
  - no points if using the gnu-build-in atomic functions
- More points: Mutexes, CVs, Semaphores in userspace
- POSIX interface

- Spinlocks in userspace (1 point)
  - no points if using the gnu-build-in atomic functions
- More points: Mutexes, CVs, Semaphores in userspace
- POSIX interface
- It is not allowed to use kernel locks via syscall
- Write tests!

- Spinlocks in userspace (1 point)
  - no points if using the gnu-build-in atomic functions
- More points: Mutexes, CVs, Semaphores in userspace
- POSIX interface
- It is not allowed to use kernel locks via syscall
- Write tests!
- Hint: See stripped-down Lock Examples in the Wiki!

- You can do basically anything OS related

- You can do basically anything OS related
- Talk to your tutor whether something is actually OS related

- Systems that are very slows
- User programs that crash the kernel
- Disabling interrupts/scheduler is very bad (except it has a quite good reason and has been discussed with a tutor)!
- Ignoring the given interfaces (test system won't compile then)

Submissions

- Points resulting from automated tests
- Tag: `SubmissionD1`
- Deadline: 30.10.2020

- Points resulting from automated tests
- Tag: `SubmissionD1`
- Deadline: 30.10.2020
- `git push`
- `git tag SubmissionD1`
- `git push --tags`
- Check whether the test system found your tag!

- Tag: `SubmissionI1`
- Deadline: 20.11.2020

- Tag: `SubmissionI1`
- Deadline: 20.11.2020
- `git push`
- `git tag SubmissionI1`
- `git push --tags`
- Check whether the test system found your tag!

- Next week
- Compulsory attendance

- Next week
- Compulsory attendance
- Say something helpful → tutor changes your tag

- Next week
- Compulsory attendance
- Say something helpful → tutor changes your tag
- tag changed?→ You've said enough. Give others the chance to speak!

- Next week
- Compulsory attendance
- Say something helpful → tutor changes your tag
- tag changed?→ You've said enough. Give others the chance to speak!
- Instant feedback

- Not prepared $\rightarrow$ 0 points

Design Discussion (2)

- Not prepared $\rightarrow$ 0 points
- Not a single word in the discussion $\rightarrow$ 0 points

- Not prepared $\rightarrow$ 0 points
- Not a single word in the discussion $\rightarrow$ 0 points
- Only repeating the task specification $\rightarrow$ 0 points

- Not prepared $\rightarrow$ 0 points
- Not a single word in the discussion $\rightarrow$ 0 points
- Only repeating the task specification $\rightarrow$ 0 points
- No design for Process/Thread separation $\rightarrow$ 0 points

Design Discussion (2)

- Not prepared $\rightarrow$ 0 points
- Not a single word in the discussion $\rightarrow$ 0 points
- Only repeating the task specification $\rightarrow$ 0 points
- No design for Process/Thread separation $\rightarrow$ 0 points
- No design for `pthread_create` $\rightarrow$ 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points
- No design for `pthread_join` → 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points
- No design for `pthread_join` → 0 points
- No design for `pthread_exit` → 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points
- No design for `pthread_join` → 0 points
- No design for `pthread_exit` → 0 points
- No design for `fork` → 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points
- No design for `pthread_join` → 0 points
- No design for `pthread_exit` → 0 points
- No design for `fork` → 0 points
- No design for `exec` → 0 points

Design Discussion (2)

- Not prepared → 0 points
- Not a single word in the discussion → 0 points
- Only repeating the task specification → 0 points
- No design for Process/Thread separation → 0 points
- No design for `pthread_create` → 0 points
- No design for `pthread_cancel` → 0 points
- No design for `pthread_join` → 0 points
- No design for `pthread_exit` → 0 points
- No design for `fork` → 0 points
- No design for `exec` → 0 points
- No design for `exit` → 0 points

Design Discussion (3)

- Try to destroy other designs
- Try to defend your own design

- Try to destroy other designs
- Try to defend your own design
- Afterwards: all groups will have a better design
- Additionally: tutor will give a few hints

- High drop-out rates
- Compulsory attendance
- Everyone needs to know everything!

- High drop-out rates
- Compulsory attendance
- Everyone needs to know everything!
- Every group member has to be able to change the parts other members have developed or implement new features that are not too difficult!
- Example: "You have not implemented exec? Okay, 30 minutes left, go implement it!"
- Timeslots will be published by the tutors