

Operating Systems

Assignment 1

Daniel Gruss

2023-10-08



1. Memory Management in SWEB
2. Processes and Threads
3. Design Decisions
4. Submissions

Memory Management in SWEB

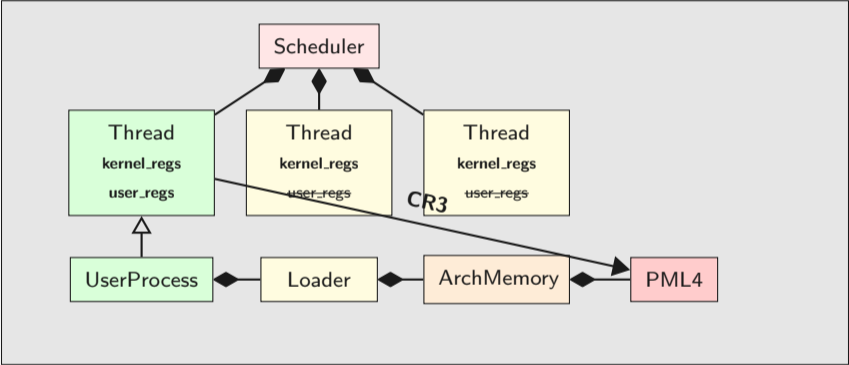
Every process has its own virtual address space (256 TB)

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



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

Processes and Threads



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

- `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`

How do function calls work?

```
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

0xffff0	<rbp> ← %rsp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	<rbp>
%rsp	0xffff0
%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

0xffff0	<rbp> ← %rsp, %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	<rbp>	0xffff0
%rsp		0xffff0
%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

0xffff0	<rbp> ← %rsp, %rbp
0xffe8	
0xffe0	
0xffd8	
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp		0xf
%rsp	0xffff0	0xff
%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

0xffff0	<rbp> ← %rbp
0xffe8	
0xffe0	
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xffff0
%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
```

0xffff0	<rbp> ← %rbp
0xffe8	
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xffff0
%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

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%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
```

0xffff0	<rbp> ← %rbp
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	
0xffc0	
0xffb8	
0xffb0	

%rbp	0xffff0
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%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

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	← %rsp
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd00xffc8
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	← %rsp
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffc80xffc0
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xffff0	← %rsp,%rbp
0xffb8		
0xffb0		

%rbp	0xffff00xffc0
%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
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %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
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %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
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	147
%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
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	714
%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
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	1421
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc00xffff0
%rsp	0xffc00xffc8
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffc80xffd0
%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

0xffff0	<rbp>	← %rbp
0xffe8	021	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%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
```

0xffff0	<rbp>	← %rbp
0xffe8	21	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%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
```

0xffff0	<rbp> ← %rbp,%rsp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	← %rsp
0xffc8	<main+13>
0xffc0	0xffff0
0xffb8	7
0xffb0	14

%rbp	0xffff0
%rsp	0xffd00xffff0
%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
```

0xffff0	<rbp> ← %rbp,%rsp
0xffe8	21
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0
0xffb8	7
0xffb0	14

%rbp	0xffff0
%rsp	0xffff0
%rax	21
%rdx	7
%rdi	7
%rsi	14

- 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

- 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 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:
 - Counter variable
 - **No** “held by” thread pointer
 - List of threads (Sleepers List)
- Increase/Decrease counter
- Counter is 0 → block → sleepers 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



ich

Process

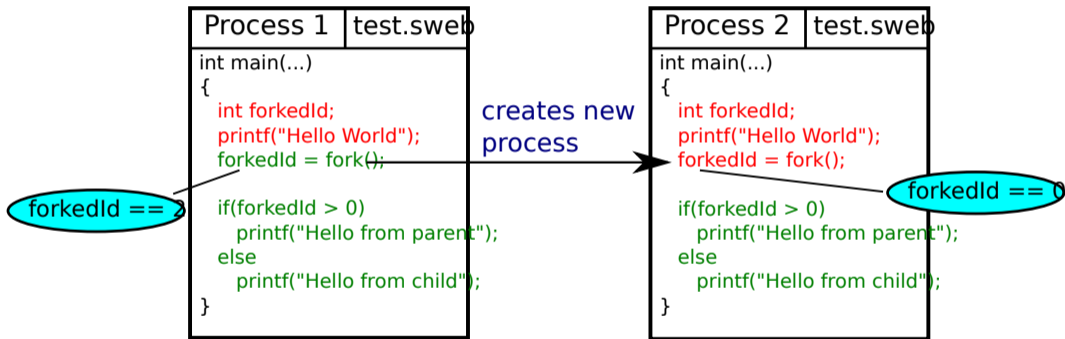
Thread

- 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)

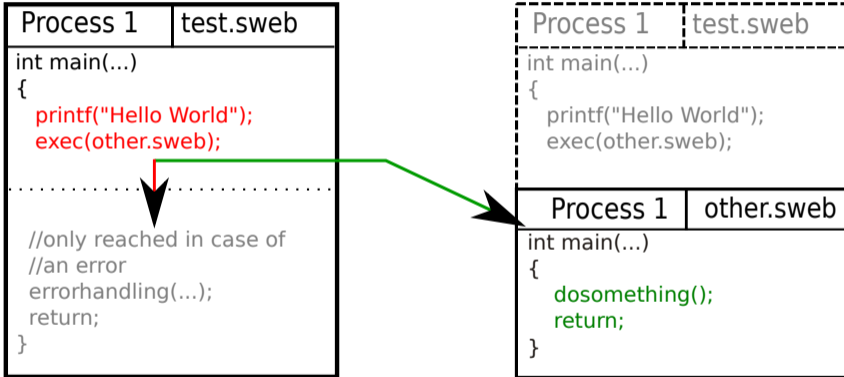
- What happens when the first thread finishes?
- When to clean-up user space and loader?
- How to detect that a function running as a thread returns?
- Where shall the stacks be placed?
- What happens if the process calls `fork/exec`?

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

- 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



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




```
// int execv(const char *path, char *const argv[]);  
  
int child_status;  
  
int pid = fork();  
  
if(pid == 0) { // child process  
    execv("program.swab",0);  
} else if(pid > 0) { // parent process  
    waitpid(pid, &child_status, WEXITED);
```

- `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`?

```
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

```
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_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 fildes[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
 - 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
- Talk to your tutor whether something is actually OS related



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

Republik Österreich

Ich will nicht mehr,
ich kann nicht mehr,
ich halte das alles
nicht mehr **OS**



Submissions

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

- Tag: `Submission11`
- `git push`
- `git tag Submission11`
- `git push --tags`
- Check whether the test system found your tag!



Circle of chairs in the seminar room

- Next week
- Compulsory attendance
- 2 pieces of paper with your name
- Get rid of at least 1 one them
- No more pieces of paper? → You've said enough. Give others the chance to speak!
- Instant feedback (don't forget paper and pencil / laptop!)

- Not prepared → 0 points
- Not a single word in the debate → 0 points
- Only repeating the task specification → 0 points



- Try to find the flaws in other's designs and ideas
- Try to defend your own design and ideas
- Afterwards: all teams will have a better design
- Additionally: tutor will give a few hints

- 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

