W4118: xv6 process operations

Instructor: Junfeng Yang

References: Modern Operating Systems (3rd edition), Operating Systems Concepts (8th edition), previous W4118, and OS at MIT, Stanford, and UWisc

Outline

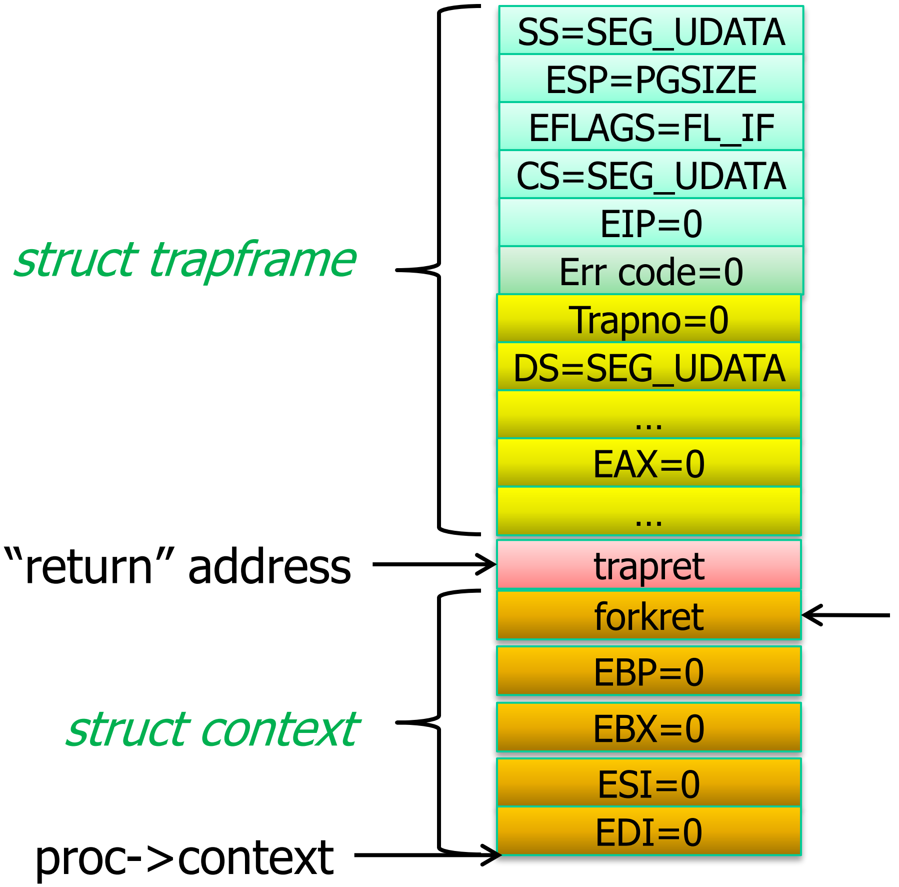
- How to create the first user process
- fork()
- exit()
- wait()
- kill()
- exec()
- sleep()
- wakeup()

Create the first user process

- Idea: create a fake trap frame, then reuse trap return mechanism

- userinit() in proc.c
  - allocproc() in vm.c allocates PCB, sets trap return address to trapret in trapasm.S, and sets "saved" kernel CPU context
  - inituvm() in vm.c sets up user space
    - Allocates a physical page for the process, sets up page table, and copies initcode
  - Set up fake trap frame
  - Set up current working directory

Init process's kernel stack



struct trapframe

SS=SEG_UDATA
ESP=PGSIZE
EFLAGS=FL_IF
CS=SEG_UDATA
EIP=0
Err code=0
Trapno=0
DS=SEG_UDATA
…
EAX=0
…

"return" address → trapret

forkret ← "saved" EIP

struct context

EBP=0
EBX=0
ESI=0
EDI=0

proc->context →

```
trapret: // trapasm.S
    popal
    …
    iret
```

```
forkret: // proc.c
    …
    ret
```

```
swtch:  // swtch.S
    …
    popl %edi
    popl %esi
    popl %ebx
    popl %ebp
    ret
```

initcode.S

```
// equivalent C code
char init[] = "/init\0";
char *argv = {init, 0};
exec(init, argv);
for(;;) exit();
```

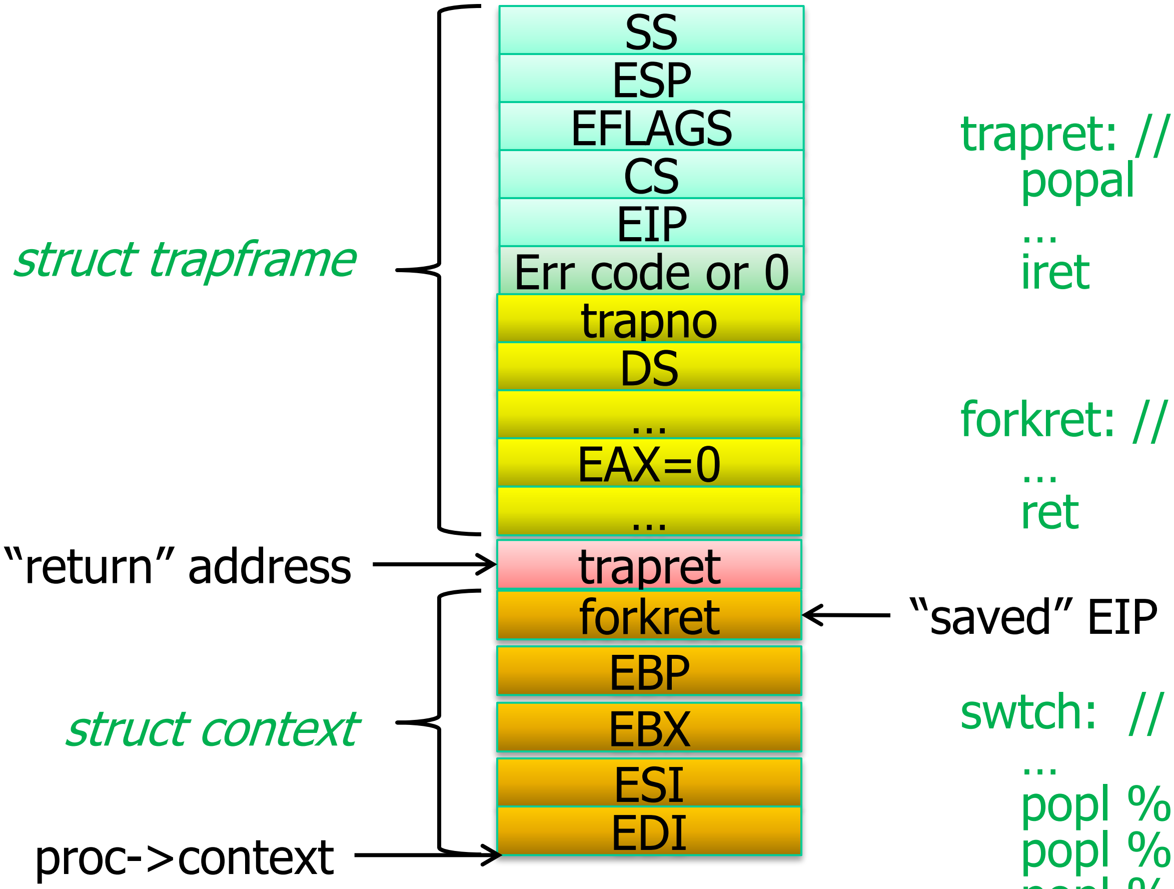
- Assembly code that
  - Sets up system call arguments
  - Moves SYS_exec to EAX
  - Traps into kernel via INT 64
- Execute init generated from init.c
- Compiled and linked into kernel
  - Makefile

fork()

- sysproc.c, proc.c

- Allocate new PCB and stack
  - Set up EIP of child to forkret ➔ trapret
- Copy address space
  - Copy both page tables and physical pages
  - Can you do better?
- Set parent pointer
- Copy parent's trap frame
- Change EAX in trap frame so that child returns 0
- Copy open file table

Child process's kernel stack



SS
ESP
EFLAGS
CS
EIP
Err code or 0
trapno
DS
…
EAX=0
…
trapret
forkret
EBP
EBX
ESI
EDI

struct trapframe

"return" address

"saved" EIP

struct context

proc->context

```
trapret: // trapasm.S
    popal
    …
    iret
```

```
forkret: // proc.c
    …
    ret
```

```
swtch:  // swtch.S
    …
    popl %edi
    popl %esi
    popl %ebx
    popl %ebp
    ret
```

exit()

- sysproc.c, proc.c

- Close open files
- Decrement reference count to current working directory
- Wake up waiting parents
- Re-parent children to init
- Set state to zombie
- Yield to scheduler

wait()

- sysproc.c, proc.c

- Find a zombie child by iterating process table
  - Can you do better?
- If there is one,
  - Free their PCB and other resources
  - Return child PID
- If no child or killed, return -1
- Repeat

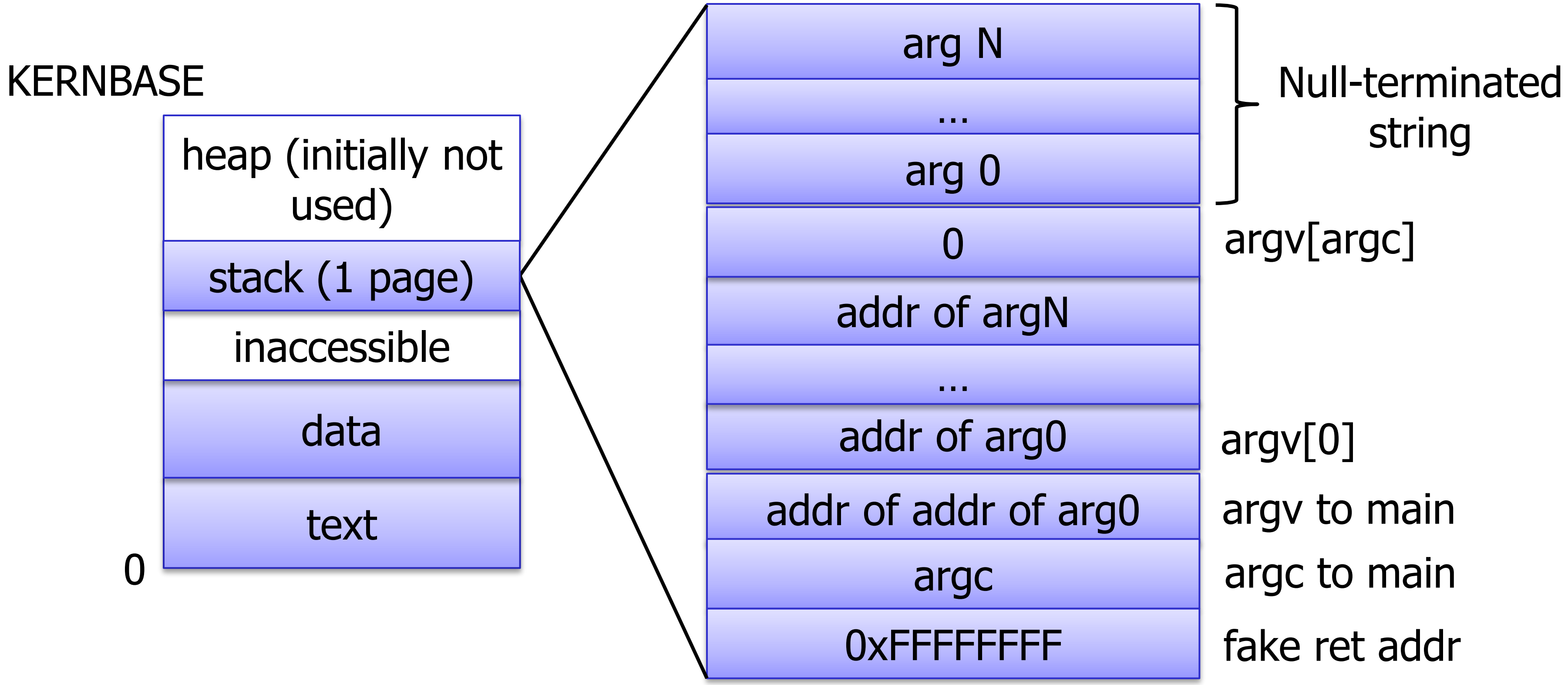
kill()

- sysproc.c, proc.c

- Set proc->killed to 1
- At various places in kernel, check this flag, and if process is killed, exit
  - trap() in trap.c
  - sys_sleep() in sysproc.c
  - piperead() & pipewrite() in pipe.c
  - proc.c

exec()

- sysfile.c, exec.c
- Set up user page table
- Load segments of the executable file into memory
- Set up stack and arguments to main(int argc, char* argv[])
- Jump to entry point (main) of the executable



sleep()

- proc.c

- Remember what we wait for (proc->chan)
- Set process state
- Yield to scheduler

wakeup()

- ❑ proc.c

- ❑ Scan through all processes
- ❑ Wake up those waiting on chan