System Security

Exec in Minix
(Lecture 5)
Roberto Guanciale

Exec

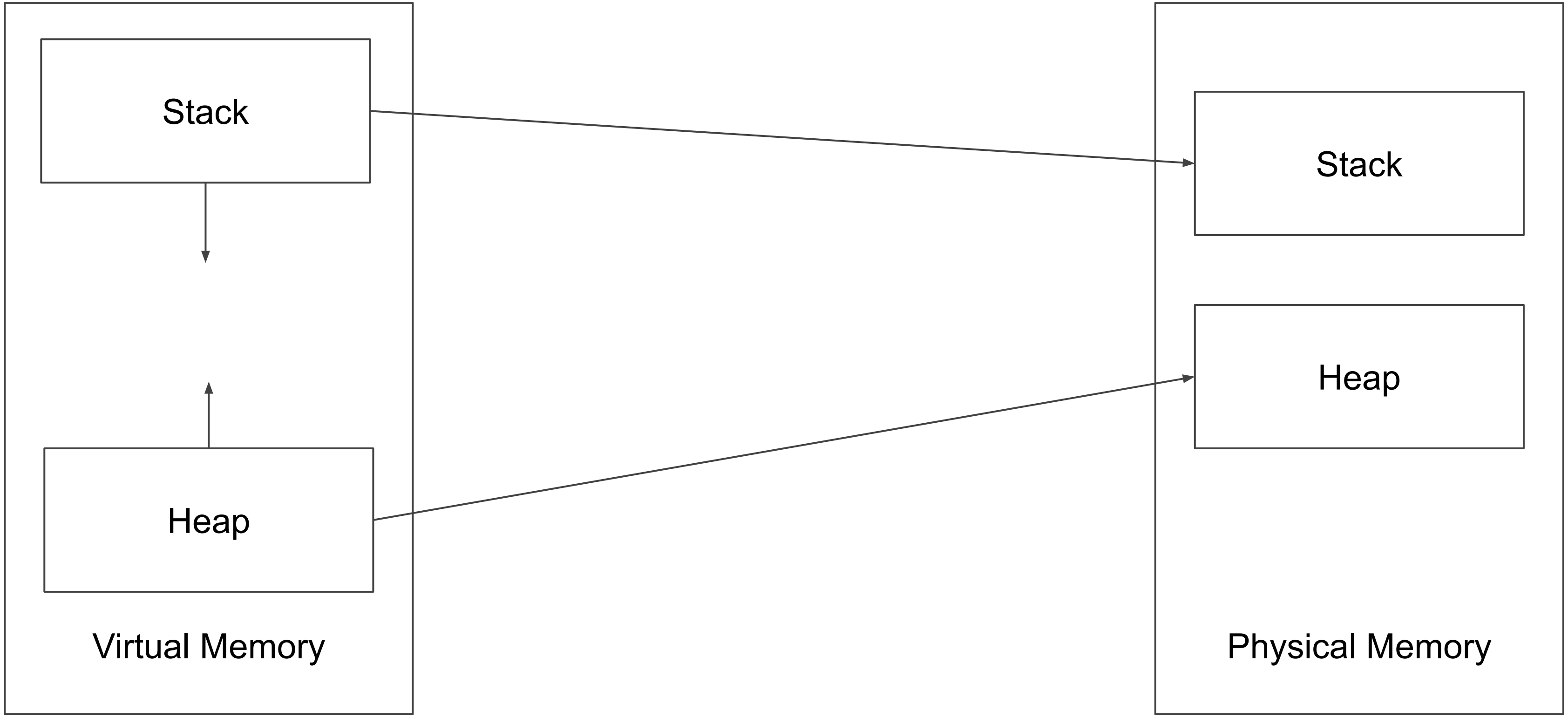
- POSIX syscall (different than MINIX kernel syscall)
- execl, execle, execlp, execv, execve, and execvp
- runs an executable file in the context of the existing process, replacing the previous executable (overlay).
  - PID does not change
  - the machine code, data, heap, and stack of the process are replaced by those of the new program
  - A file descriptor opened when an exec call is made will remain open in the new process image

http://cinnabar.sosdg.org/~qiyong/qxr/minix3/source

Exec implementation

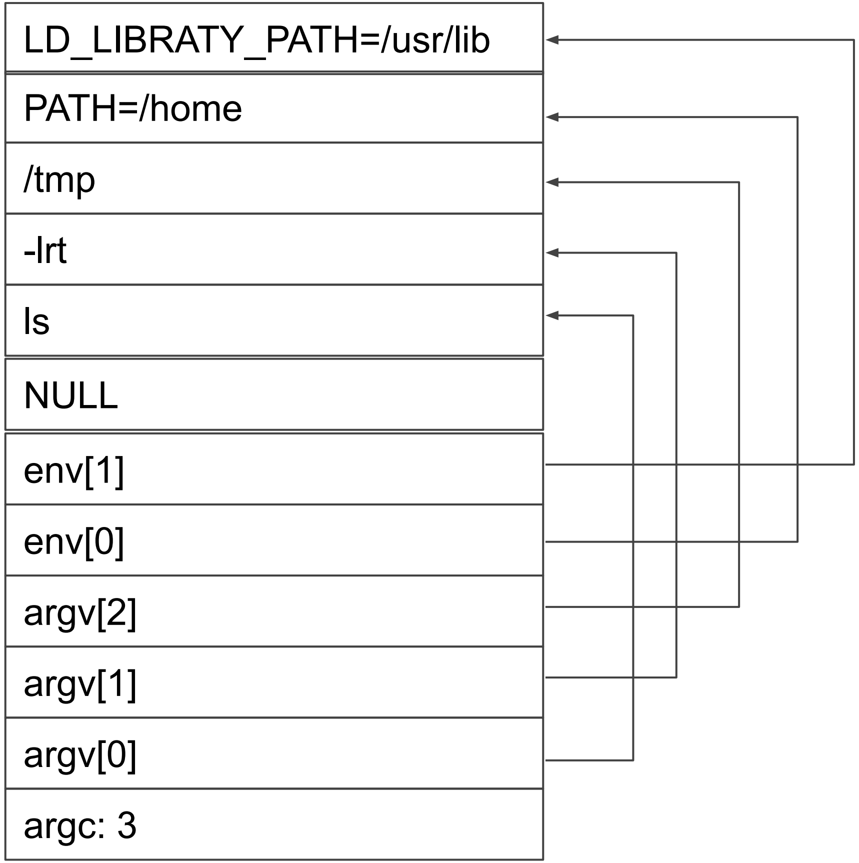
- lib/libc/gen/execl.c, execle.c, execlp.c, execv.c, execvp.c
    - User space wrappers
- minix/lib/libc/sys/execve.c
    - Process invokes PM_EXEC (14) syscall of process manager (PM_PROC_NR)
    - Done using ipc_sendrec via the kernel
- Before that, process prepares the initial stack for the new executable
    - The initial stack is used for argv (program arguments)
    - Notice that this stack is prepared by the user process

Creation of the new Stack



New Stack

```
main (argc, argv)      for       ls -lrt /tmp
```



LD_LIBRATY_PATH=/usr/lib

PATH=/home

/tmp

-lrt

ls

NULL

env[1]

env[0]

argv[2]

argv[1]

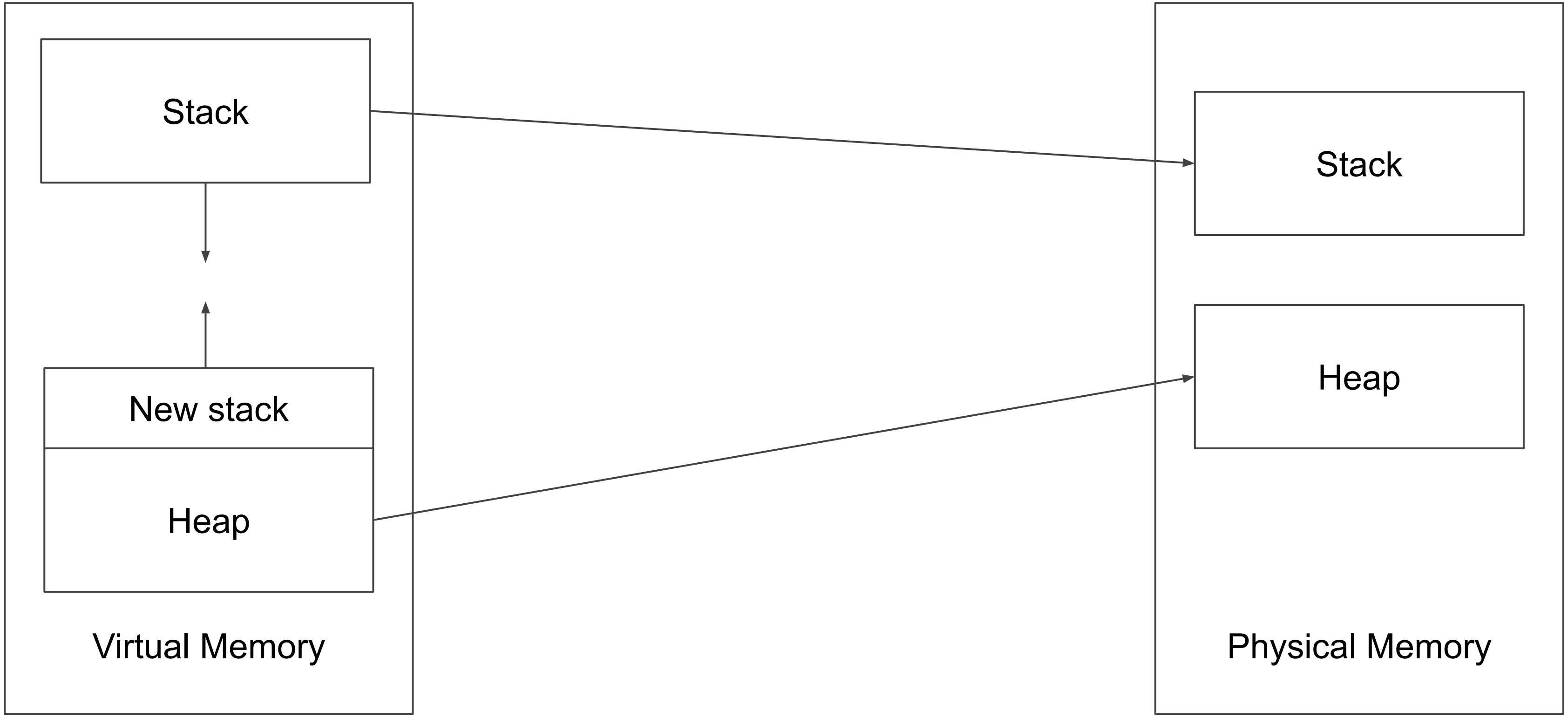
argv[0]

argc: 3

Libc exec

- minix_stack_params
  - minix/lib/libc/sys/stack_utils.c#L76
  - computes size of the new stack
    - 1 integer for argc +
    - (1 pointer + string) for arg
    - (1 pointer + string) for env
    - 1 pointer for NULL terminated env
- allocates space on the heap for the new stack
  - sbrk

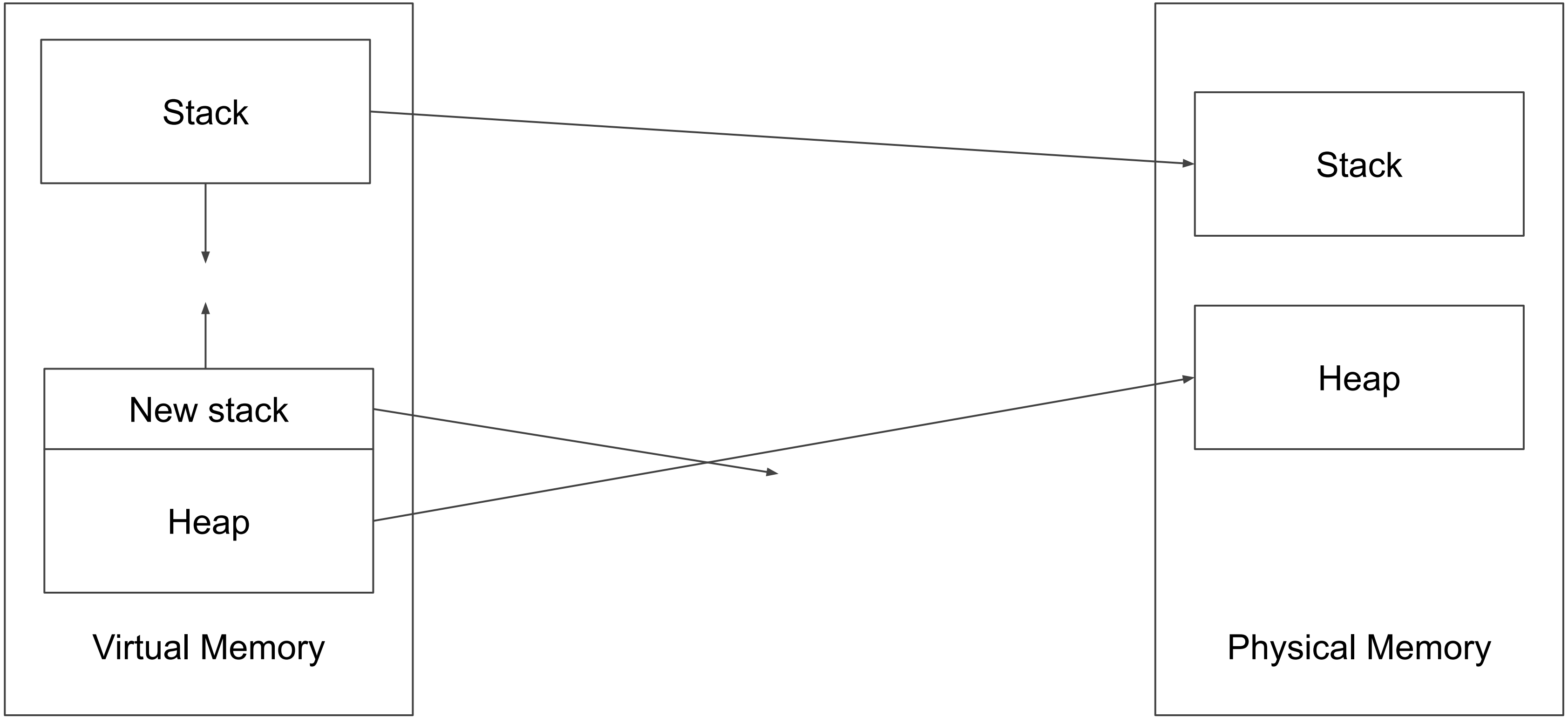
Creation of the new Stack



sbrk

- change the location of the program break, which defines the end of the process's data segment
    - (exec allocates on the heap bypassing malloc)
- minix/lib/libc/sys/sbrk.c
- minix/lib/libc/sys/brk.c
- _syscall(VM_PROC_NR, VM_BRK)
- minix/servers/vm/break.c#L63
    - map_region_extend_upto_v

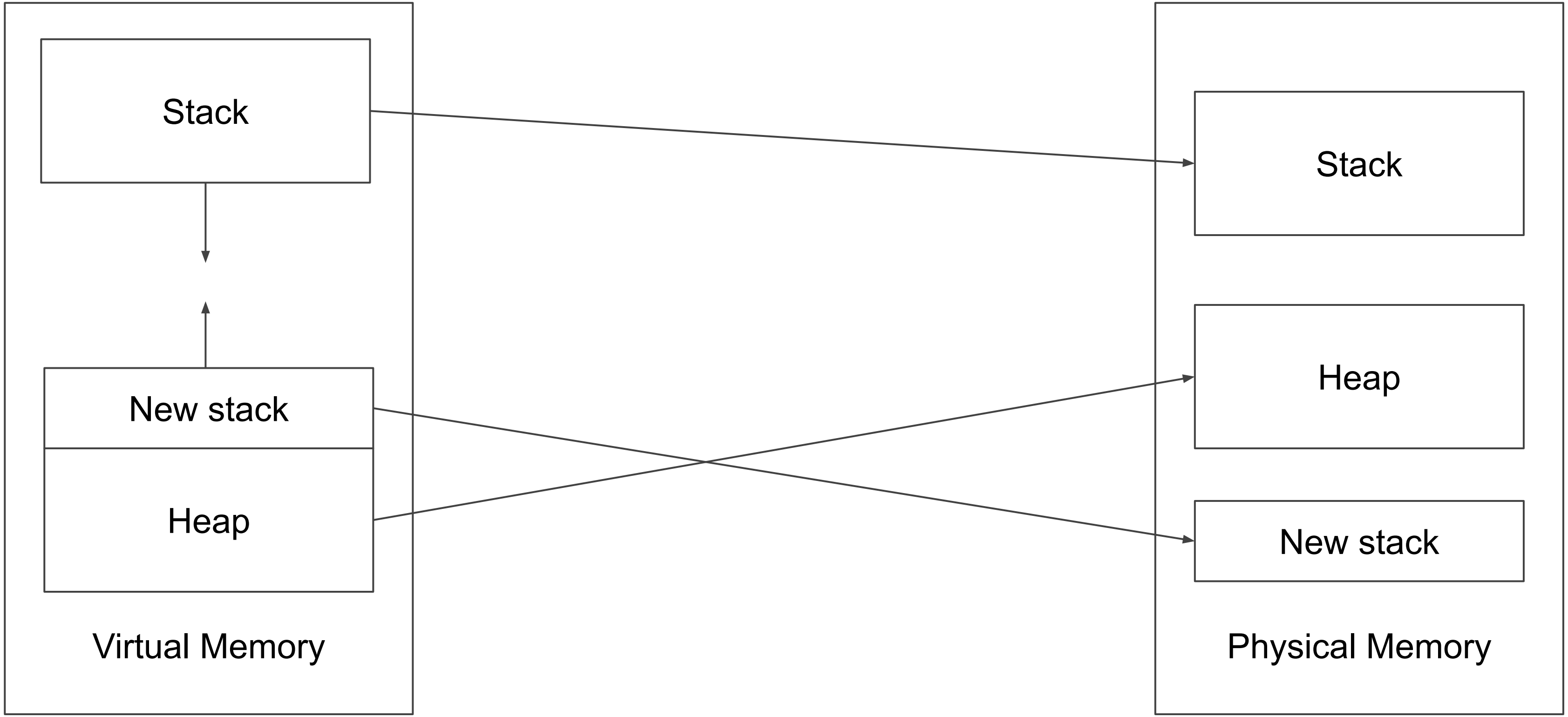
Creation of the new Stack



sbrk

- No need to allocate the new virtual memory in physical memory
- As soon as the process writes in the new heap region
  - Page fault
  - VM allocates physical memory and map virtual memory
  - Process writes into the heap
  - There can be multiple pages, which requires multiple page faults
- After writing the new stack.... (and possibly several page faults)

Creation of the new Stack

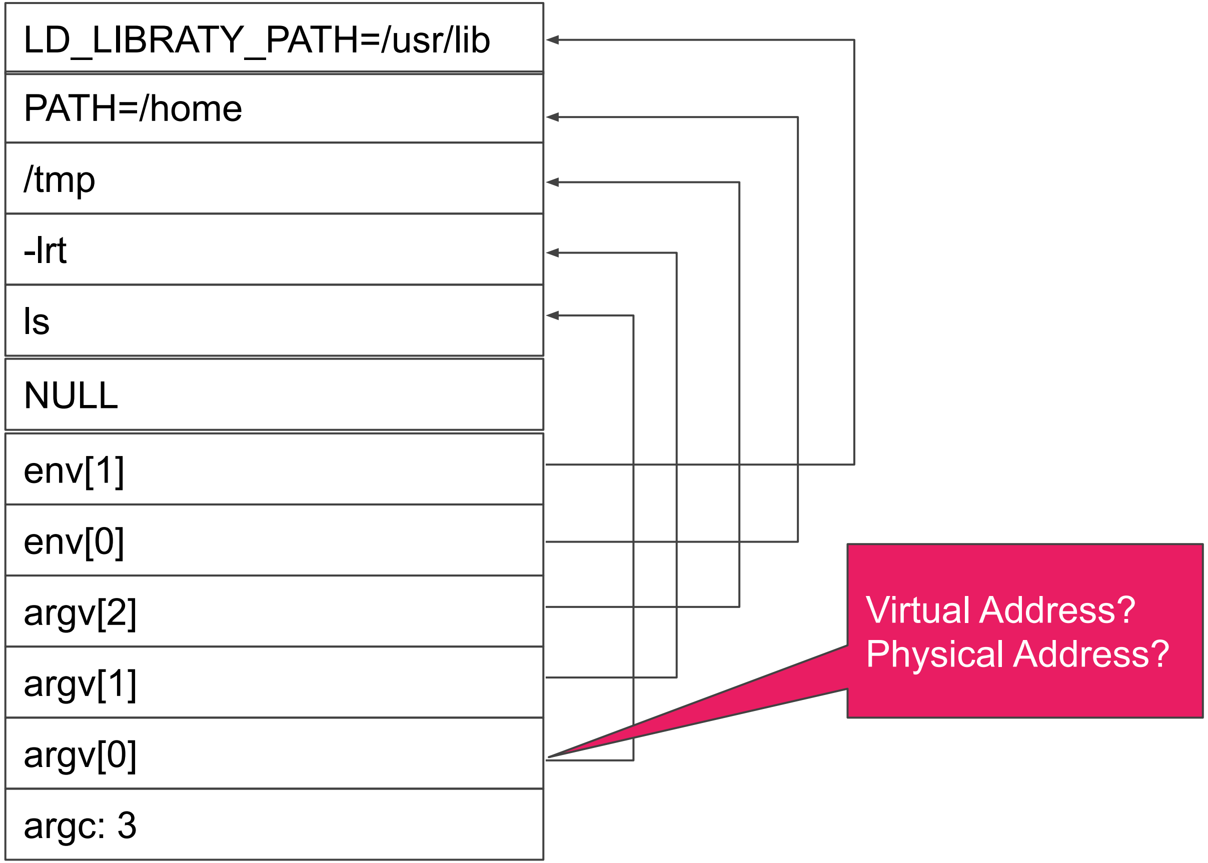


Libc exec

- minix_stack_fill writes the new stack
    - minix/lib/libc/sys/stack_utils.c#L119

New Stack

`main (argc, argv)      for      ls -lrt /tmp`



LD_LIBRATY_PATH=/usr/lib

PATH=/home

/tmp

-lrt

ls

NULL

env[1]

env[0]

argv[2]

argv[1]

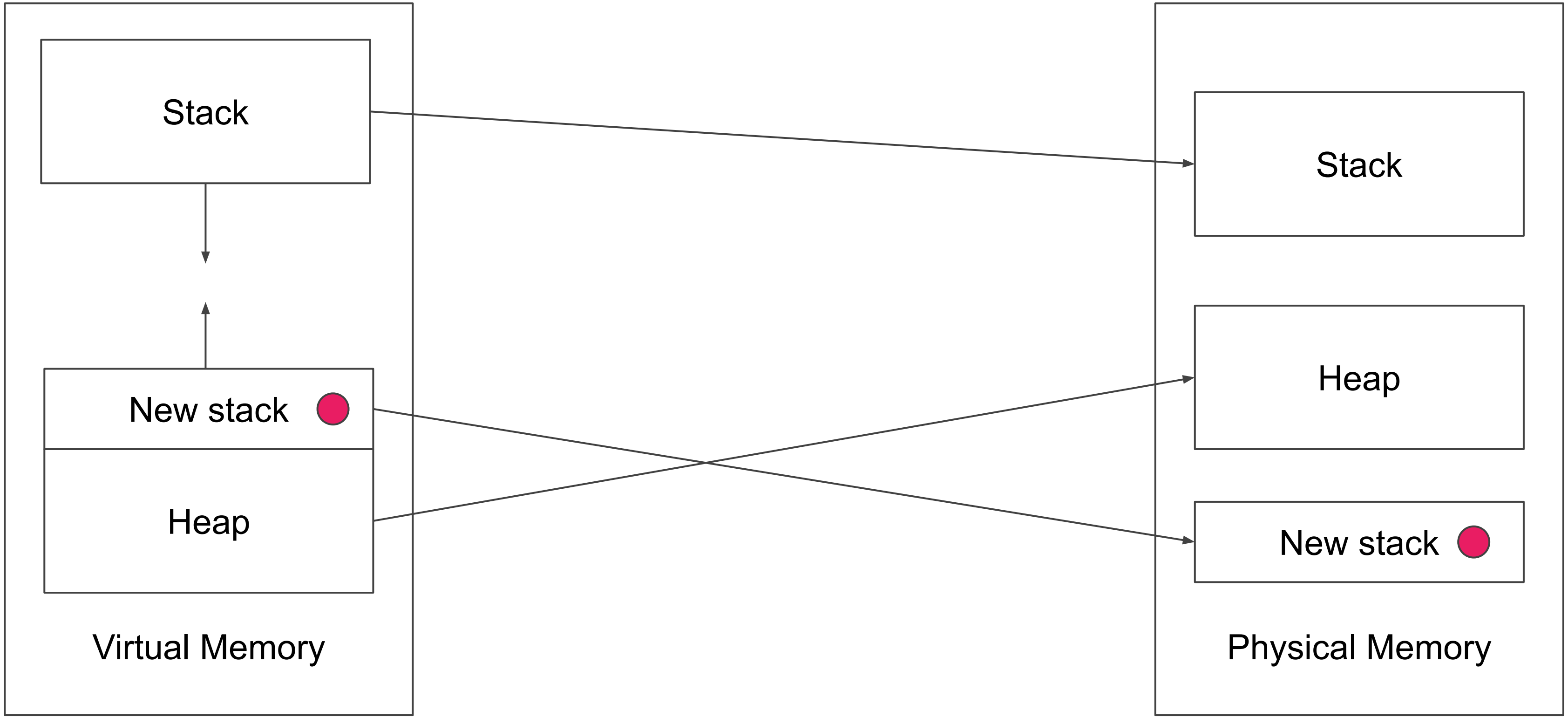
argv[0]

argc: 3

Virtual Address?
Physical Address?

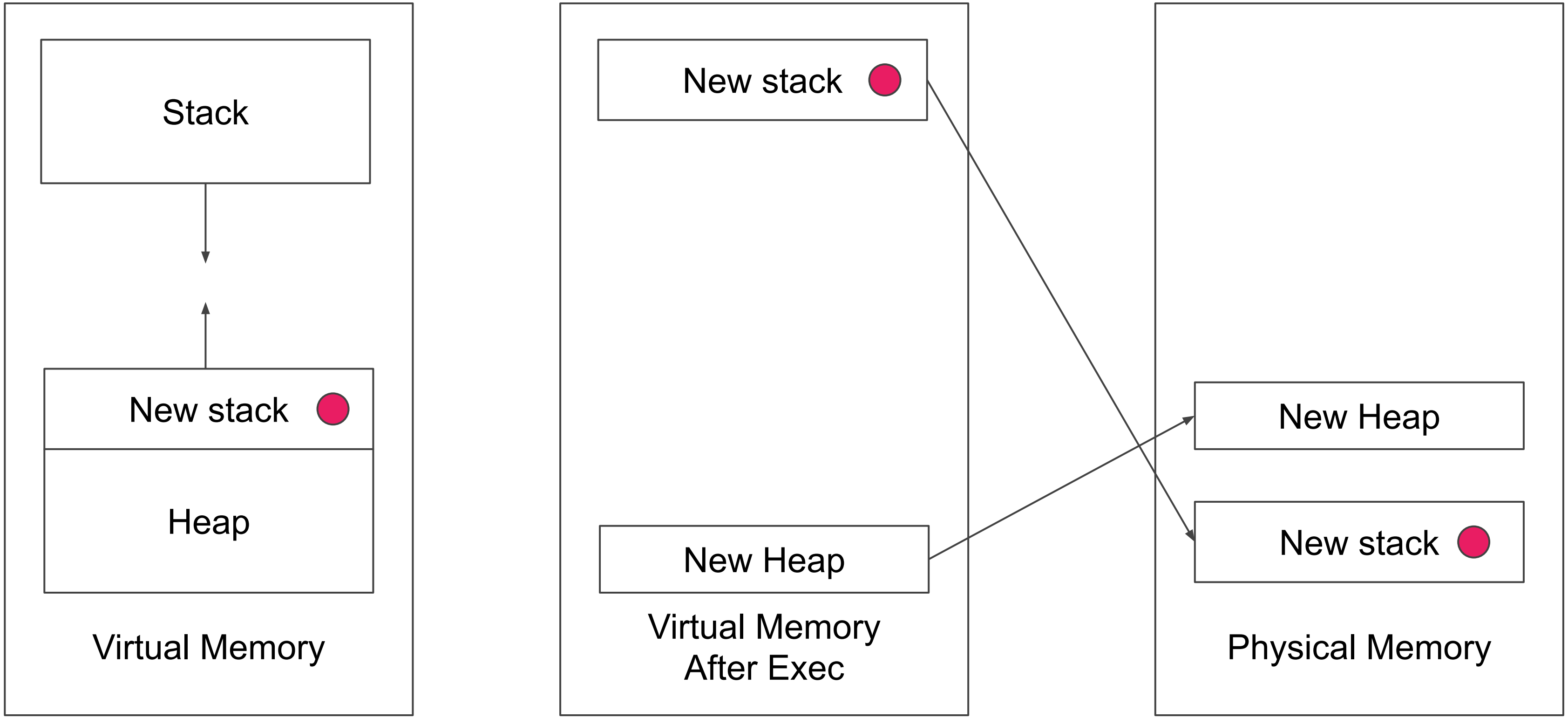
Creation of the new Stack

● argv[0] string



Creation of the new Stack

● argv[0] string

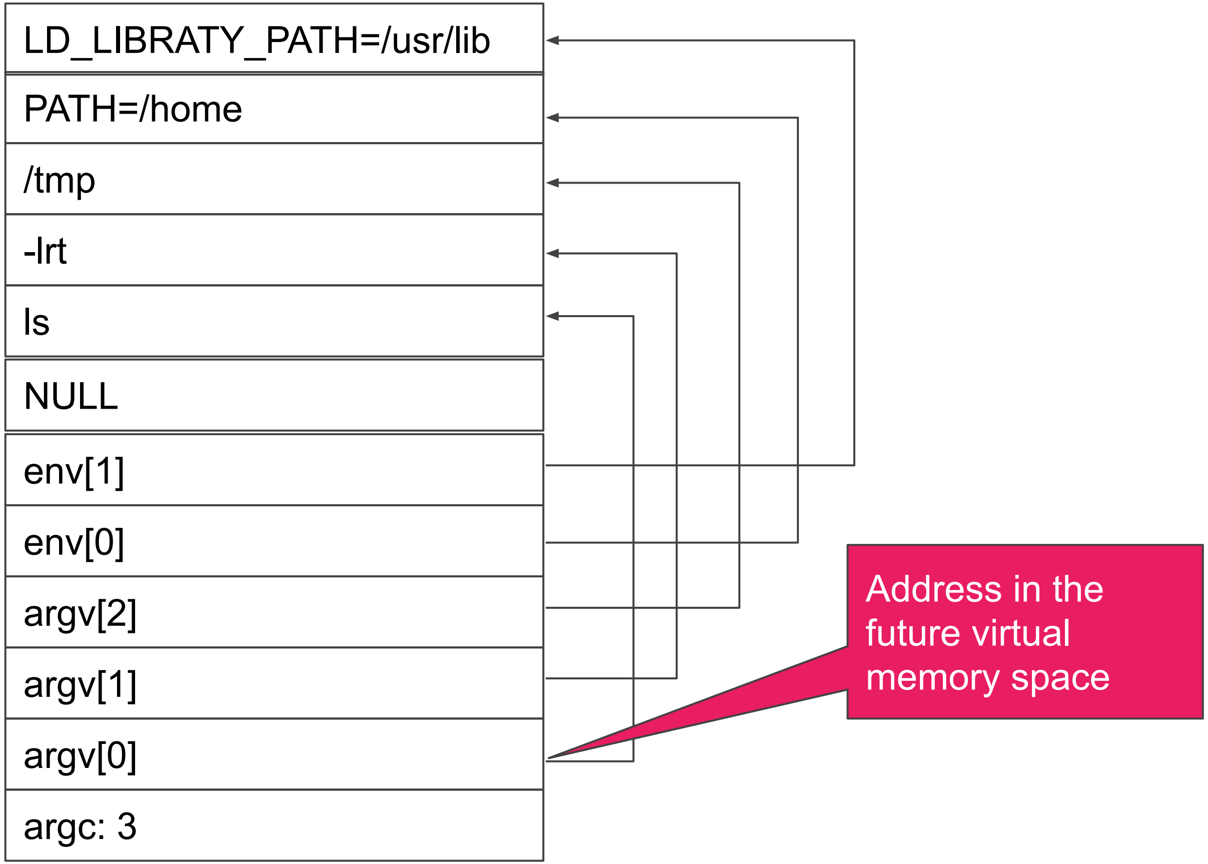


Libc exec

- minix_stack_fill writes the new stack
  - minix/lib/libc/sys/stack_utils.c#L119
- *vsp = minix_get_user_sp() - stack_size
  - minix/lib/libc/sys/stack_utils.c#L133
- minix_get_user_sp()
  - lib/libc/sys/kernel_utils.c#L40
  - Kernel info initialized when a program starts (exec) by libc, invoking ipc_minix_kerninfo syscall
  - user_sp is the same for every process, configured by kui_user_sp
- minix_stack_fill uses this information to fix the pointers in the new stack

New Stack

`main (argc, argv)      for      ls -lrt /tmp`



LD_LIBRATY_PATH=/usr/lib

PATH=/home

/tmp

-lrt

ls

NULL

env[1]

env[0]

argv[2]

argv[1]

argv[0]

argc: 3

Address in the future virtual memory space

Libc exec

- Finally exec invokes the syscall
  - _syscall(PM_PROC_NR, PM_EXEC)
- It specifies the size and position of the stack (even if m.m_lc_pm_exec.ps_str should be the same for all processes)
- Exec protocol (among servers) quite complicated
  - It is designed to avoid deadlocks

PM

- minix/servers/pm/exec.c consists of several steps:
  - do_exec: receives the request from the user process
    - Forwards the request to VFS using the VFS_PM_EXEC message
    - Async
  - do_newexec: handle PM part of exec call after VFS
    - e.g. setuid etc
  - exec_restart: finish a regular exec call
  - do_execrestart: finish the special exec call for RS

VFS

- pm_exec does the actual work
  - minix/servers/vfs/exec.c#L185
- interoperates with file systems (and disk drivers) to parse and load the elf
- communicates with VM to create the new virtual memory
  - map the executable
  - allocate stack
- replies to PM

VFS

- Get_read_vp
  - reads the header of the executable using map_header
  - invokes req_readwrite to communicate with the file system
    - minix/servers/vfs/exec.c#L754
  - Notice cpf_grant_magic
    - minix/servers/vfs/request.c#L836
    - It enables VFS to grant a real file system to write/read memory of a process
      - In general can be a process that requested a memory read (user processes cannot use cpf_grant)

VFS

- Elf load done by libexec_load_elf
  - Using callbacks in minix/servers/vfs/exec.c#L338
  - stack_size and stack_high have been identified by pm_exec
  - For every segment
    - If mmap is enabled, informs VM about the vfs_mmap
      - minix/servers/vfs/exec.c#L161
    - Otherwise
      - Asks VM for junk mmap
      - Copies the segment (via the filesystem process)
  - Asks VM to allocate free memory for the stack

VFS

- VFS informs PM that process has been loaded
- PM do_newexec
  - minix/servers/pm/exec.c#L62
  - Sets PM informations in PM table e.g. UID GID
- Stack_prepare_elf
  - Copies stack data into the new stack
- Sends VFS_PM_EXEC_REPLY to PM

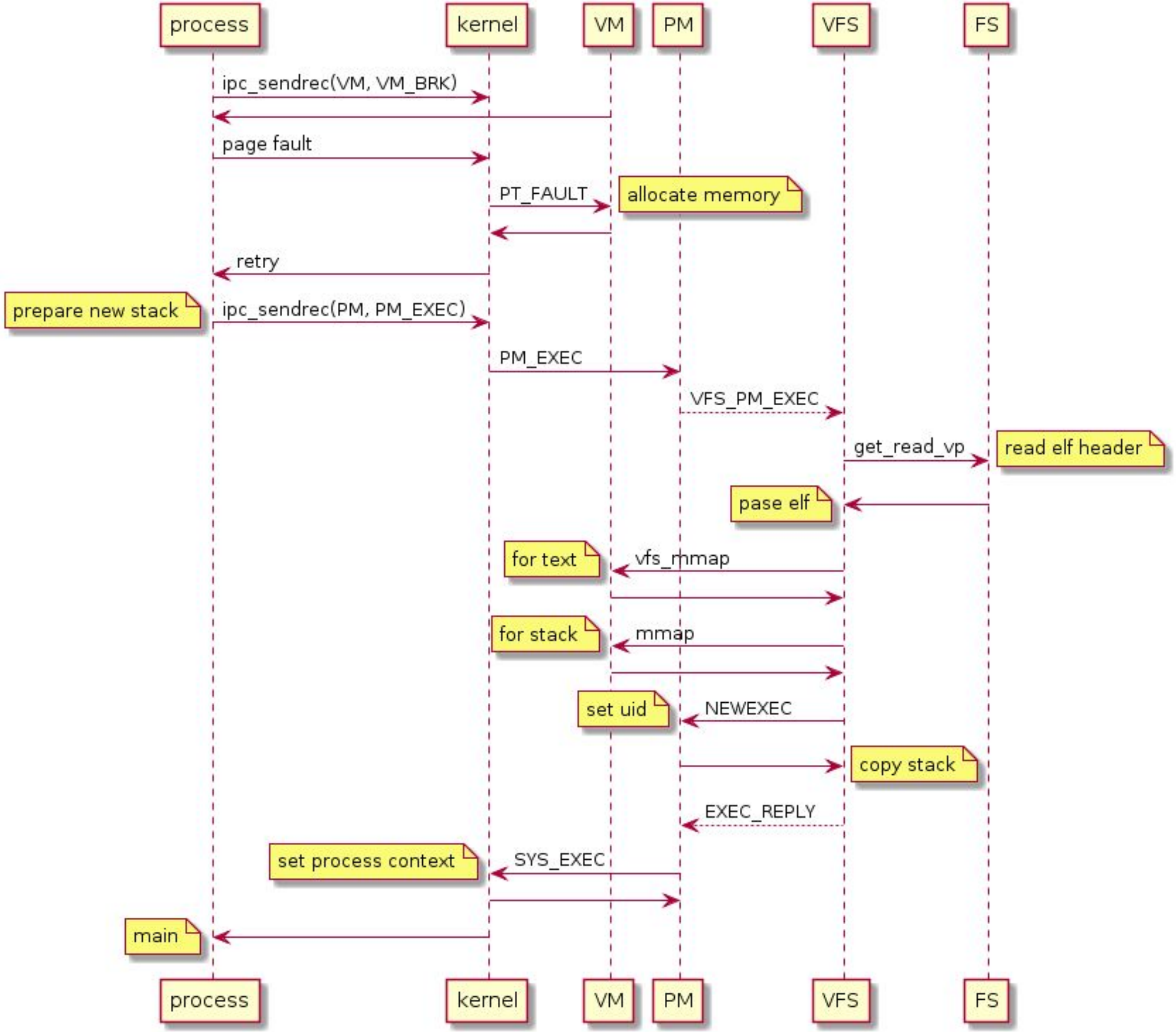
PM

- exec_restart
  - Completes exec
  - sys_exec informs the kernel the exec is done (informing about pc and sp)
- Does not reply to the process
  - Kernel will activate the process later

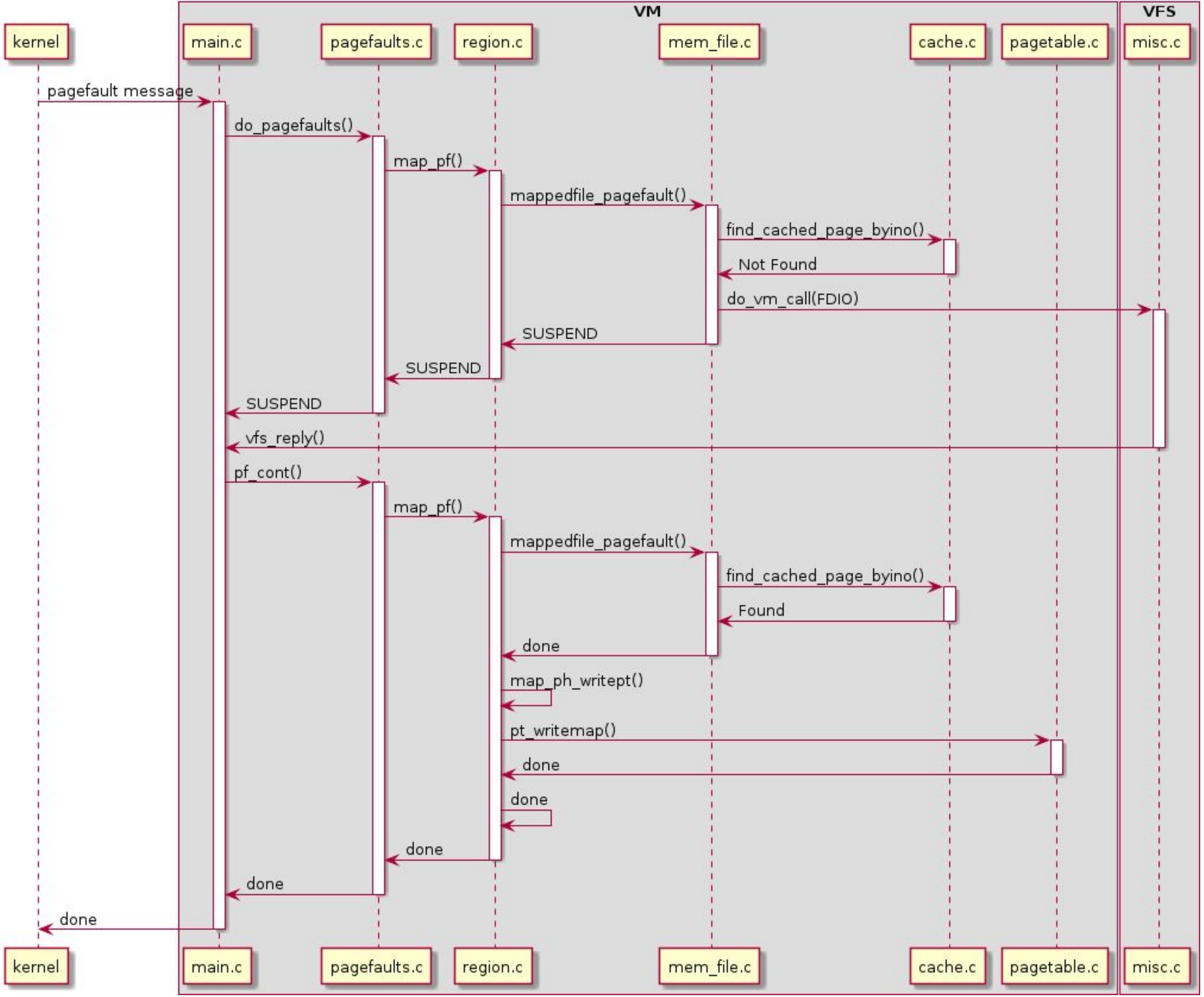
Kernel

- sys_exec
    - minix/kernel/system/do_exec.c#L20
    - Save command name for debugging, ps(1) output
    - Update process context
        - PC, SP
    - Unmark process as waiting a reply from PM so it is runnable

UML


process
kernel
VM
PM
VFS
FS
ipc_sendrec(VM, VM_BRK)
page fault
PT_FAULT
allocate memory
retry
prepare new stack
ipc_sendrec(PM, PM_EXEC)
PM_EXEC
VFS_PM_EXEC
get_read_vp
read elf header
pase elf
for text
vfs_mmap
for stack
mmap
set uid
NEWEXEC
copy stack
EXEC_REPLY
set process context
SYS_EXEC
main
process
kernel
VM
PM
VFS
FS

Page fault


VM
VFS
kernel
main.c
pagefaults.c
region.c
mem_file.c
cache.c
pagetable.c
misc.c
pagefault message
do_pagefaults()
map_pf()
mappedfile_pagefault()
find_cached_page_byino()
Not Found
do_vm_call(FDIO)
SUSPEND
SUSPEND
SUSPEND
vfs_reply()
pf_cont()
map_pf()
mappedfile_pagefault()
find_cached_page_byino()
Found
done
map_ph_writept()
pt_writemap()
done
done
done
done
done
kernel
main.c
pagefaults.c
region.c
mem_file.c
cache.c
pagetable.c
misc.c

Heap

- Stack built by the executing process
- Data and Text built by VFS
- Heap?
- lib/libc/stdlib/malloc.c
  - malloc_init initialize the structures needed for malloc
  - malloc uses sbrk to increase size of data memory, whose top contains the heap

Adding a new service to minix

minix/servers/myserver/ simple service with ping

- Makekfile
  - CPPFLAGS.myserver.c copied from DS service (black magic)
  - .include <minix.service.mk> it’s a service
- proto.h (prototypes), inc.h and myserve.h (dependencies)
- main.c
  - Uses sef, sef_receive to receive messages synchronously, ipc_send to send reply, dispatches MYSERVER_SYS1
- myserver.c
  - implementation of sef and syscall

Additional files to compile and deploy the service

- minix/include/minix/com.h
  - Define MYSERVER endpoint (fixed ID)
  - Defines MYSERVER_SYS1
- minix/servers/Makefile
  - includes compilation of myserver
- distrib/sets/lists/minix-base/mi
  - includes the binary
- etc/system.conf
  - enables myservice to interact with other services

Wrapper

- minix/include/minix/myserver.h
    - Wrapper to invoke the syscall
- minix/lib/libsys/myserver.c
    - Implementation of the wrapper
- minix/include/minix/Makefile
    - Adds the wrapper prototype to compile the kernel
- minix/lib/libsys/Makefile
    - Adds the wrapper to libsys
- distrib/sets/lists/minix-comp/mi
    - Adds the wrapper prototype to the file list

Service strart-up

- Service must be started when minix boot
  - with a fixed endpoint number
- releasetools/Makefile
  - Adds myservice to the initial image
- minix/kernel/table.c
  - Informs kernel about the new service
- minix/servers/rs/table.c
  - Informs RS about the new service
- distrib/sets/lists/minix-kernel/mi
  - Add the new service to the boot files

Invocation of myservice

- Services cannot be directly invoked by user-processes
- minix/drivers/mydriver/mydriver.c
  - Uses the syscall wrapper
- minix/drivers/mydriver/mydriver.conf
  - IPC access to all services

Questions