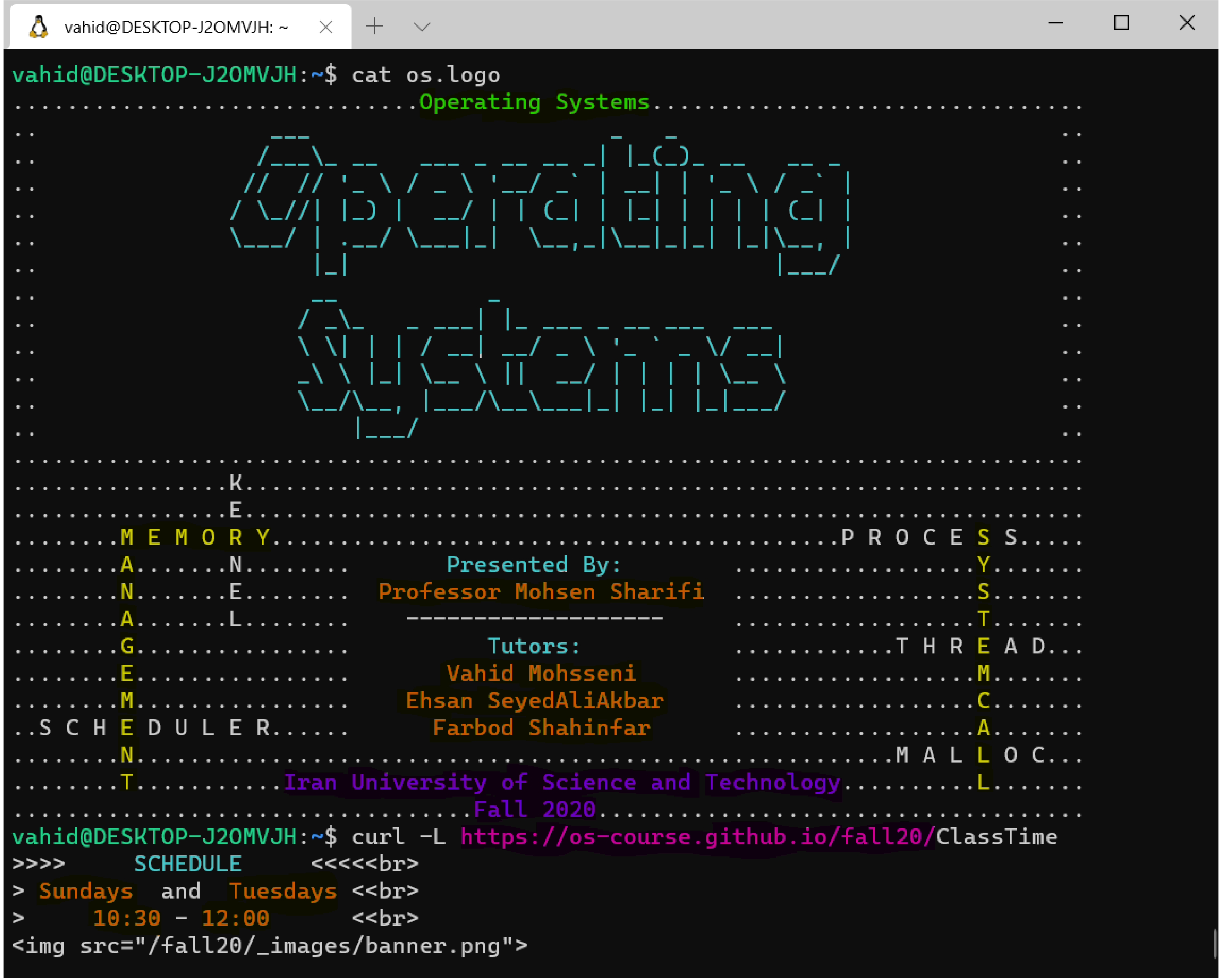
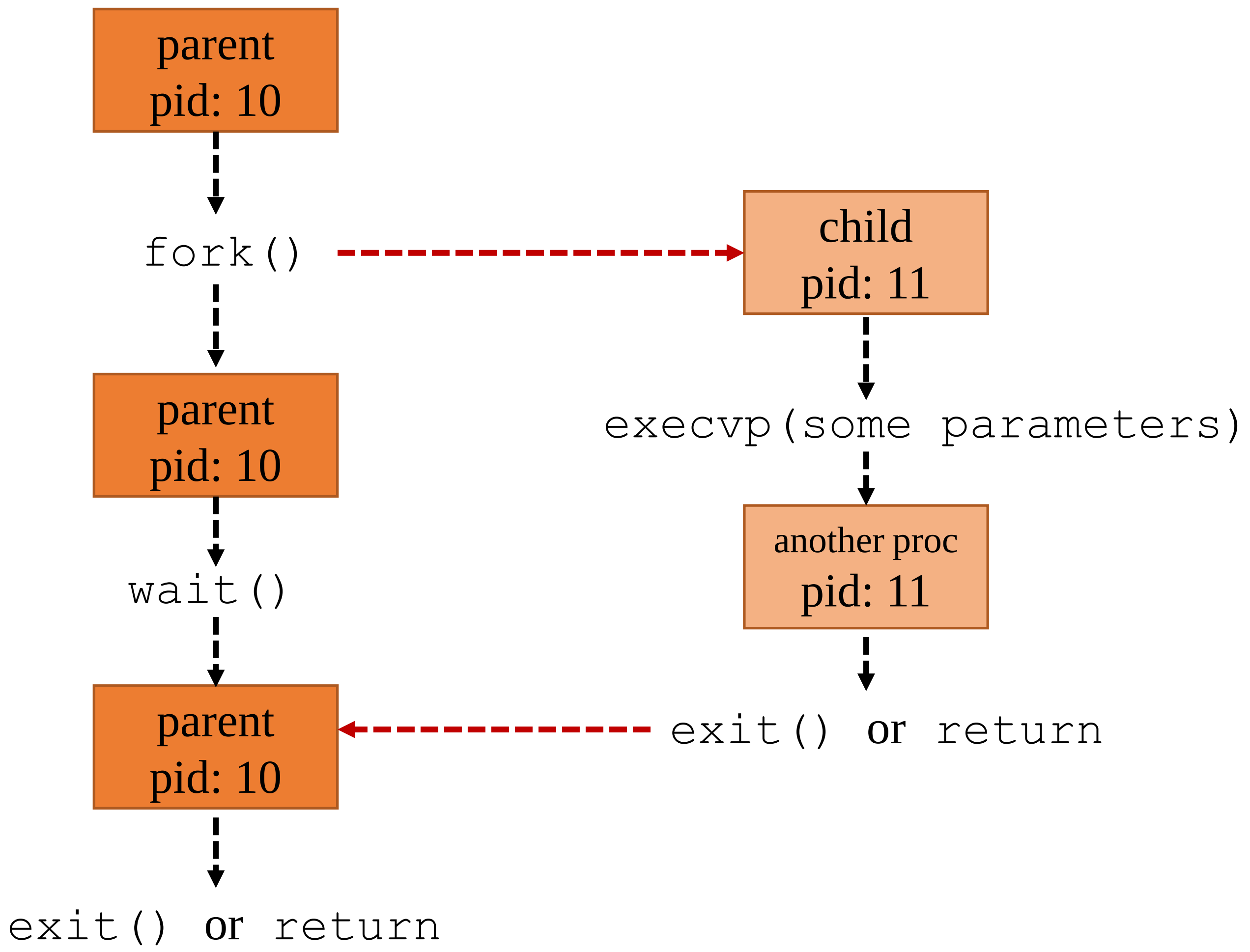
Operating Systems

Inter-Process Communication

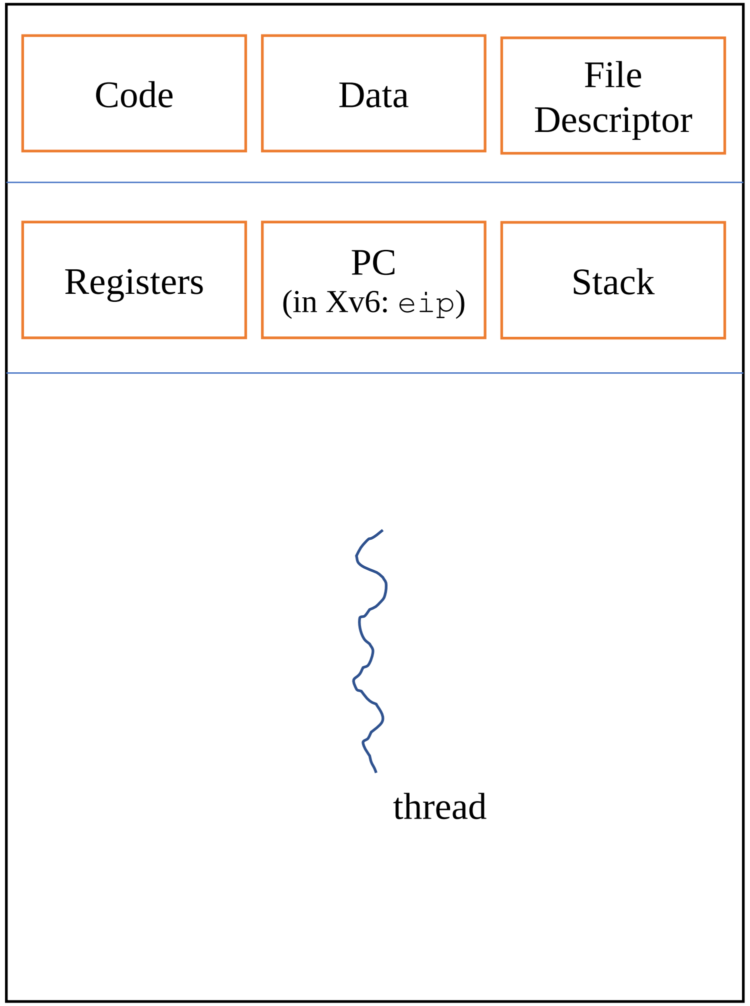
Fall 2020

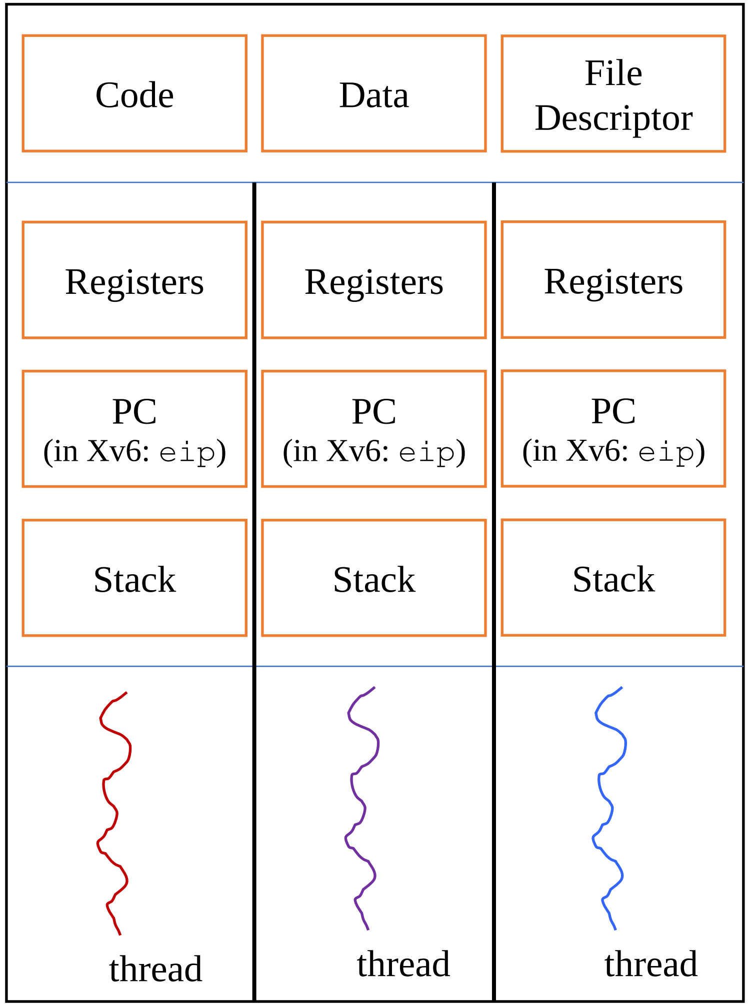

vahid@DESKTOP-J2OMVJH: ~
vahid@DESKTOP-J2OMVJH:~$ cat os.logo
Operating Systems
KERNEL
MEMORY MANAGEMENT
PROCESS
SYSTEM CALL
THREAD
MALLOC
SCHEDULER
Presented By:
Professor Mohsen Sharifi
Tutors:
Vahid Mohsseni
Ehsan SeyedAliAkbar
Farbod Shahinfar
Iran University of Science and Technology
Fall 2020
vahid@DESKTOP-J2OMVJH:~$ curl -L https://os-course.github.io/fall20/ClassTime
>>>> SCHEDULE <<<<
> Sundays and Tuesdays <
> 10:30 - 12:00 <
<img src="/fall20/_images/banner.png">

fork and exec



Process and Multi-Thread





Communication between threads

The code segment is shared between threads.

We can define shared variables.

All threads may access them.

The programmer should control the concurrent accesses using mutual exclusions.

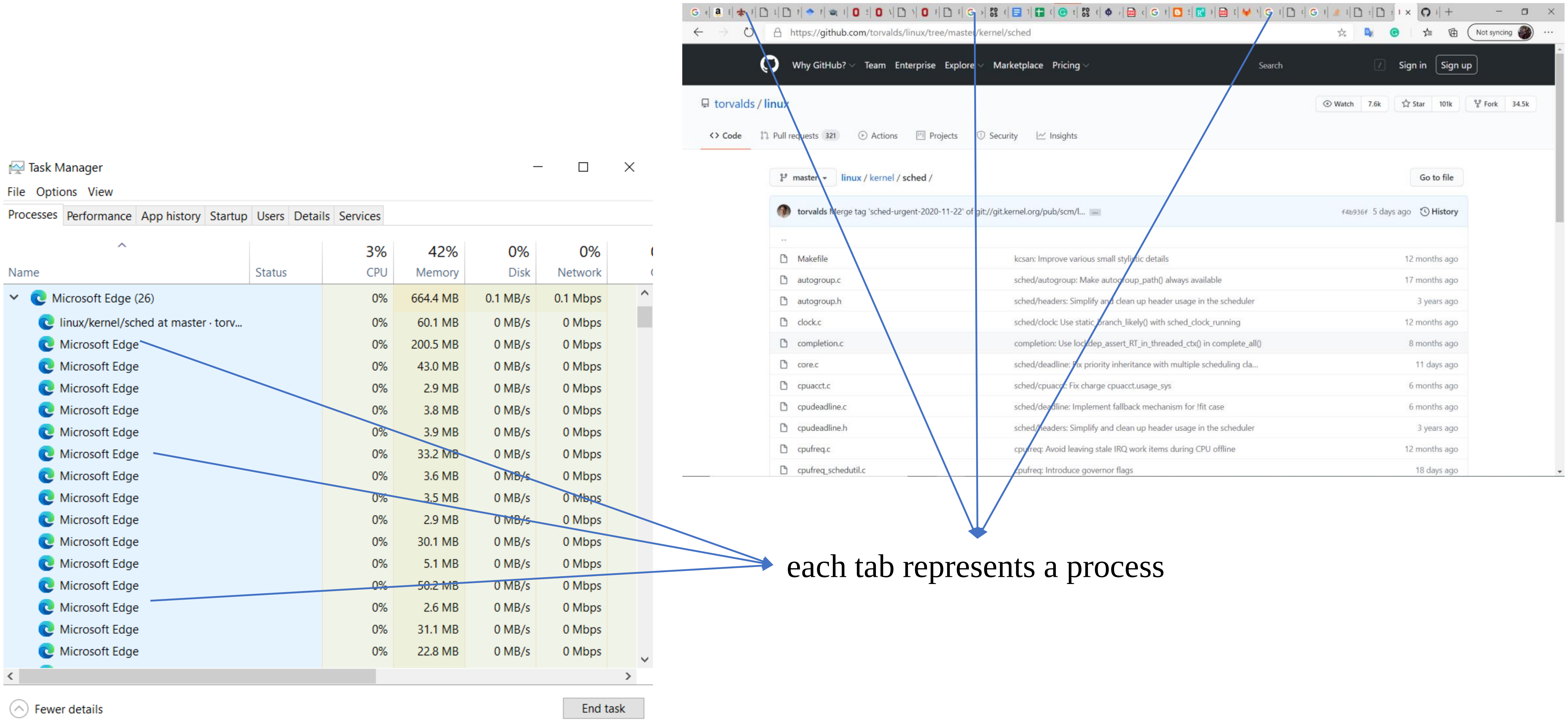
How processes Communicate with each other?

Inter-Process Communication (IPC) mechanism makes it possible for processes share data together.

Why?

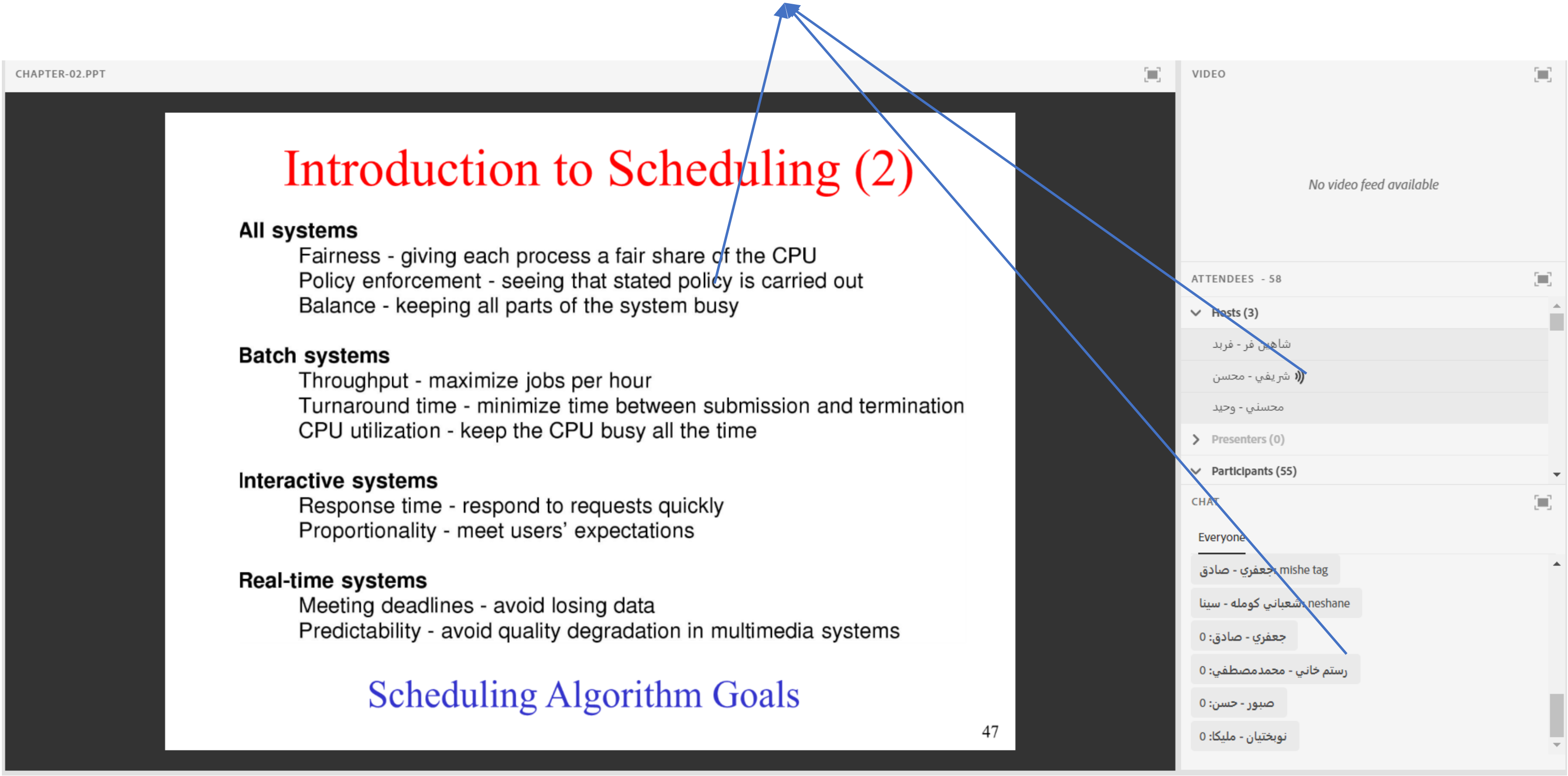
- Information Sharing
- Computation Speed up
- Modularity

Web browser example



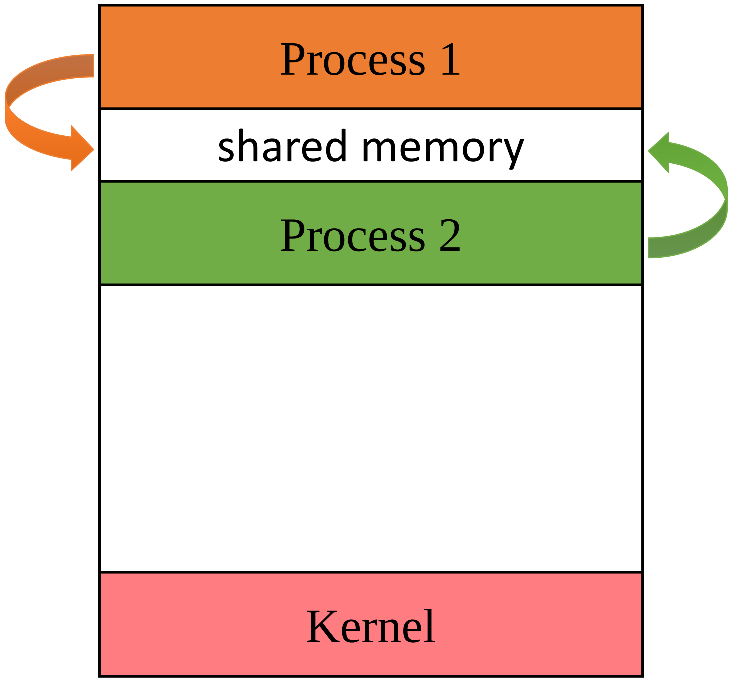
Web browser example

interactive elements within a tab could be a thread

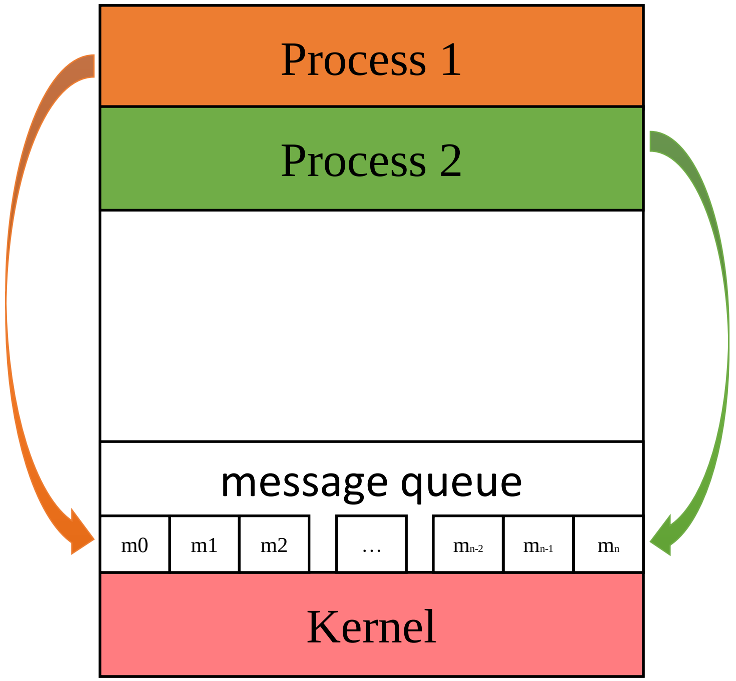


Fundamental Models

- Shared Memory



- Message Passing



Shared Memory

- Creates a region.
- This region typically resides in the address space of creator process.
- Other processes attach this segment into their address space.
- OS prevents two processes from accessing each other’s address space.
- Data exchange is not under OS control. The processes are responsible for ensuring that they are not writing data simultaneously on the same location.

Shared Memory - POSIX API

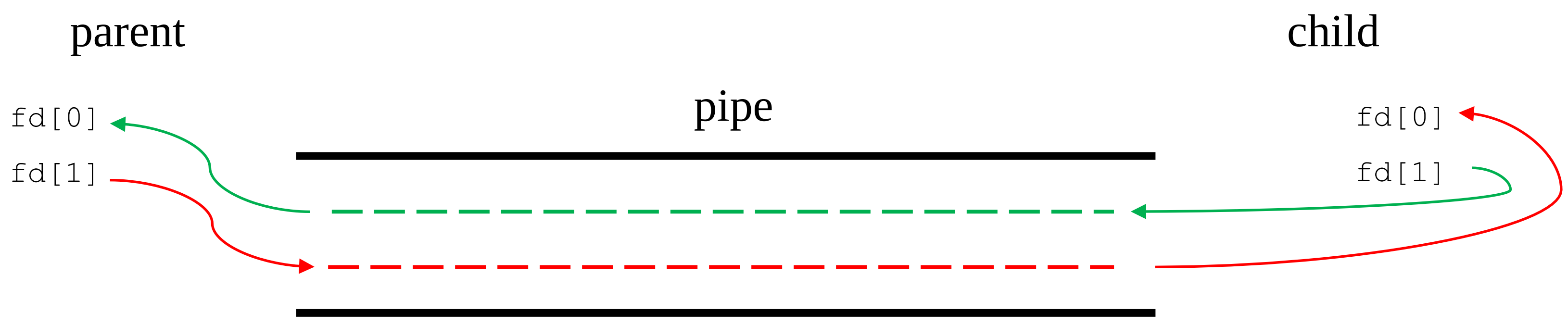
- Producer
  - create file descriptor of shared memory object:
    - `fd = shm_open(name, flag, mode)`
  - truncate the file to the given size:
    - `ftruncate(fd, size)`
  - map the FD to the memory of process
    - `ptr = mmap(addr, size, prot, flag, FD, offset)`
  - write the data on the memory
    - `sprintf(ptr, data)`
- Consumer
  - open the already created file by its name
    - `fd = shm_open(name, flag, mode)`
  - map the FD to the memory of process
    - `ptr = mmap(addr, size, prot, flag, FD, offset)`
  - read the data from the memory
    - `printf("%s\n", ptr)`
  - close the link
    - `shm_unlink(name)`

example: https://github.com/os-course/iustfall20/blob/master/07_inter-process_communication/shared_mem_shm_open.c

Message Passing

- OS intervenes in providing a mechanism for IPC.
- Two simple APIs.
  - `Send`
  - `Receive`
- We should consider three features/methods for such a mechanism:
  - Direct or Indirect Communication
    - direct send/recv or mailbox
  - Synchronous or Asynchronous Communication
    - blocking and non-blocking send/recv
  - Buffering
    - zero, bounded, and unbounded capacity

Message Passing – PIPE API



Shared Memory - POSIX API

- Define variables
  - `int fd[2];`
  - `char buff[80];`
- Initialize pipe
  - `pipe(fd);`
- Write to and Read from pipe file descriptor
  - `write(fd[1], “string”, strlen(“string”) + 1);`
  - `read(fd[0], buff, sizeof(buff));`

example: https://github.com/os-course/iustfall20/blob/master/07_inter-process_communication/msg_pass_pipe.c

Questions?

?