

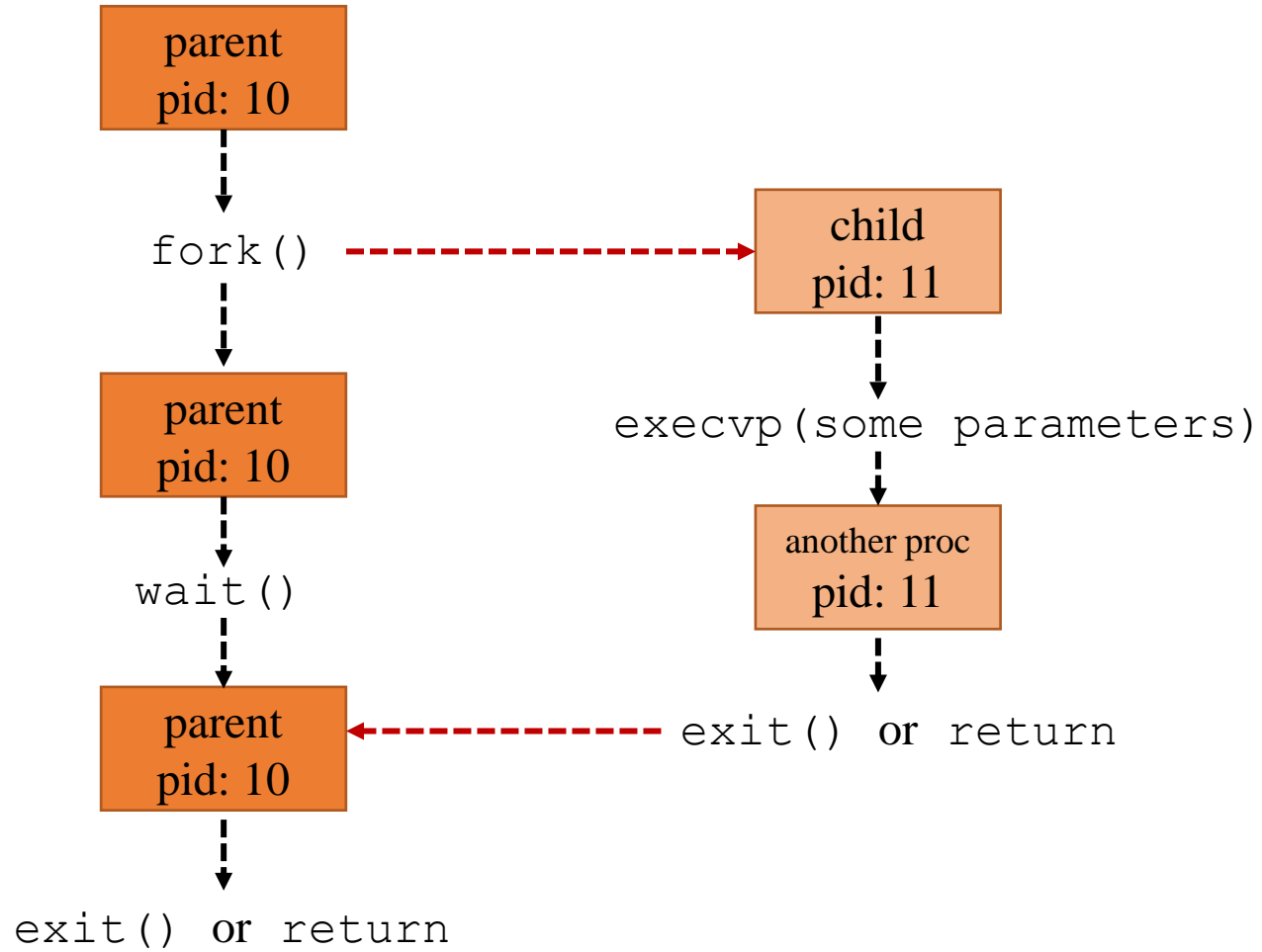


Operating Systems

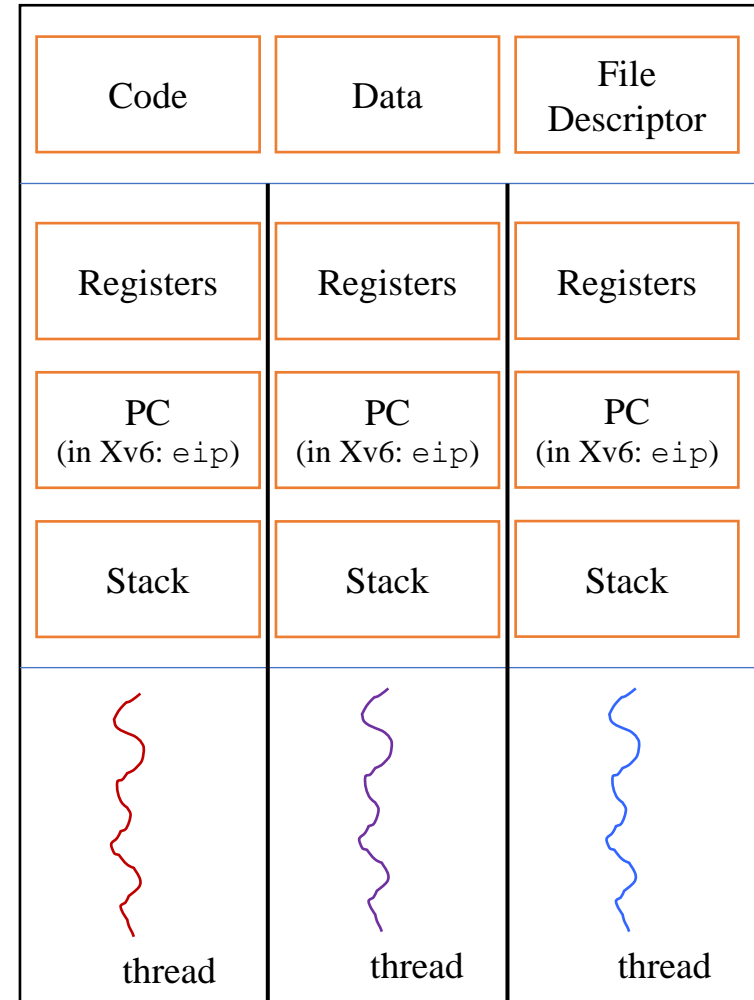
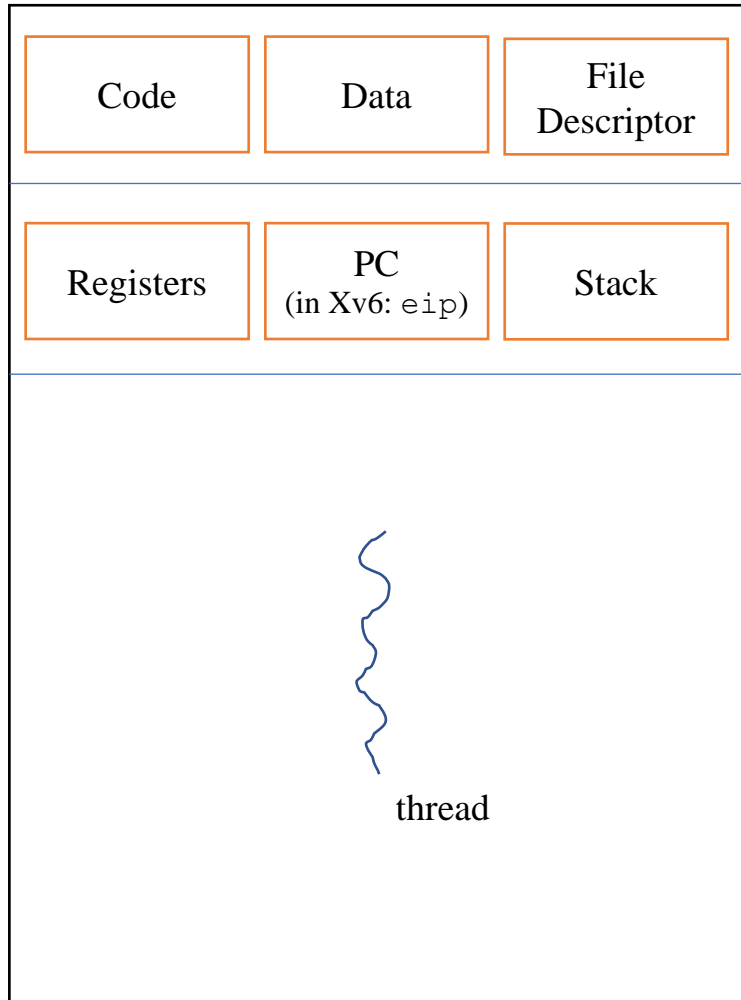
Inter-Process Communication

Fall 2020

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

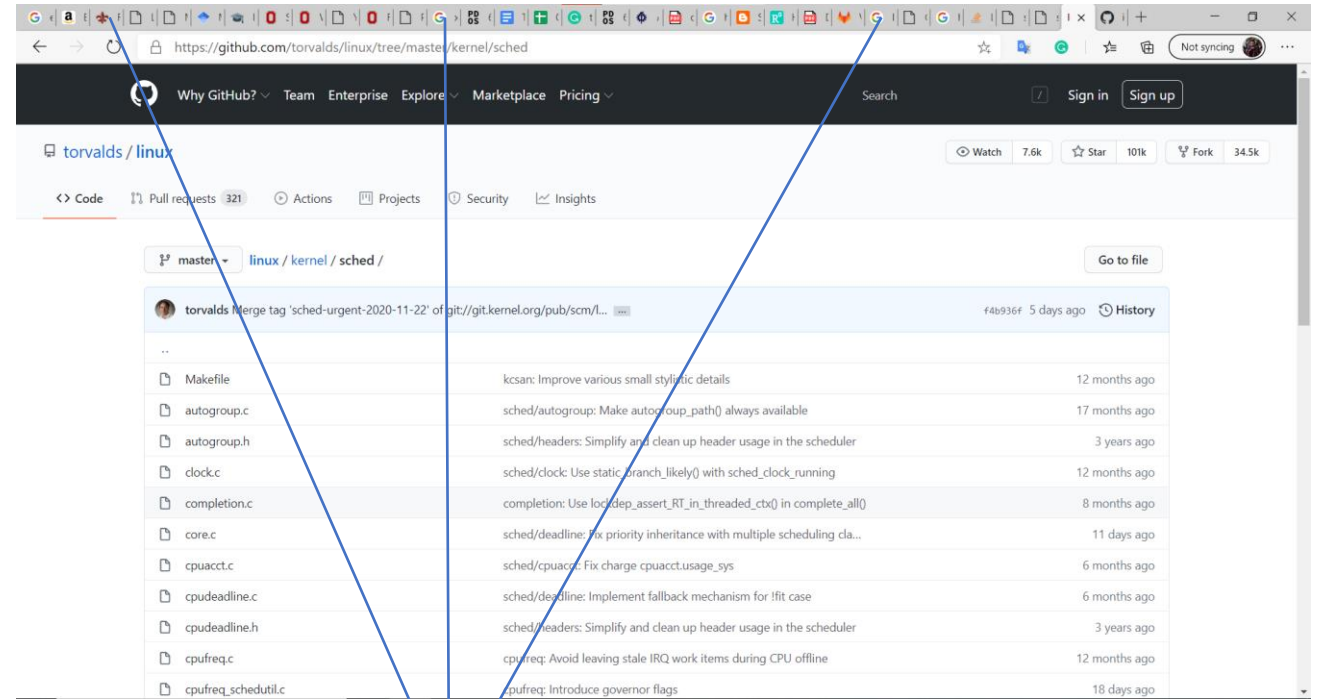
Task Manager

File Options View

Processes Performance App history Startup Users Details Services

Name	Status	3% CPU	42% Memory	0% Disk	0% Network
Microsoft Edge (26)		0%	664.4 MB	0.1 MB/s	0.1 Mbps
linux/kernel/sched at master · torv...		0%	60.1 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	200.5 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	43.0 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	2.9 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	3.8 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	3.9 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	33.2 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	3.6 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	3.5 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	2.9 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	30.1 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	5.1 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	50.2 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	2.6 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	31.1 MB	0 MB/s	0 Mbps
Microsoft Edge		0%	22.8 MB	0 MB/s	0 Mbps

Fewer details End task



each tab represents a process

Web browser example

interactive elements within a
tab could be a thread

CHAPTER-02.PPT

Introduction to Scheduling (2)

All systems
Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems
Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems
Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Scheduling Algorithm Goals

47

VIDEO
No video feed available

ATTENDEES - 58

Posts (3)

- شاهین فر - فرید
- شرفی - محسن
- محسنی - وحید

Presenters (0)

Participants (55)

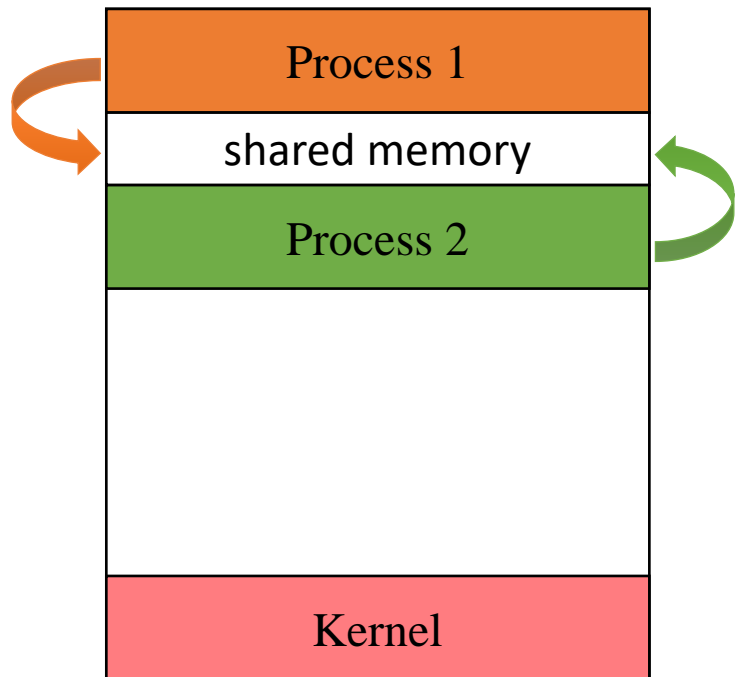
CHAT

Everyone

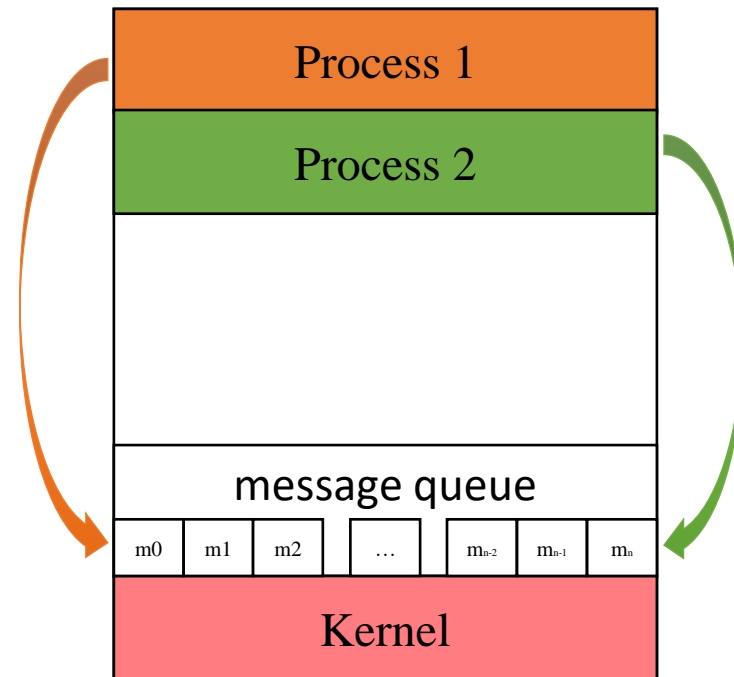
- صادق جعفری - mishe tag
- شعبانی کومله - neshane
- جعفری - صادق: 0
- رستم خانی - محمدمصطفی: 0
- صبور - حسن: 0
- نویختیان - ملیکا: 0

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

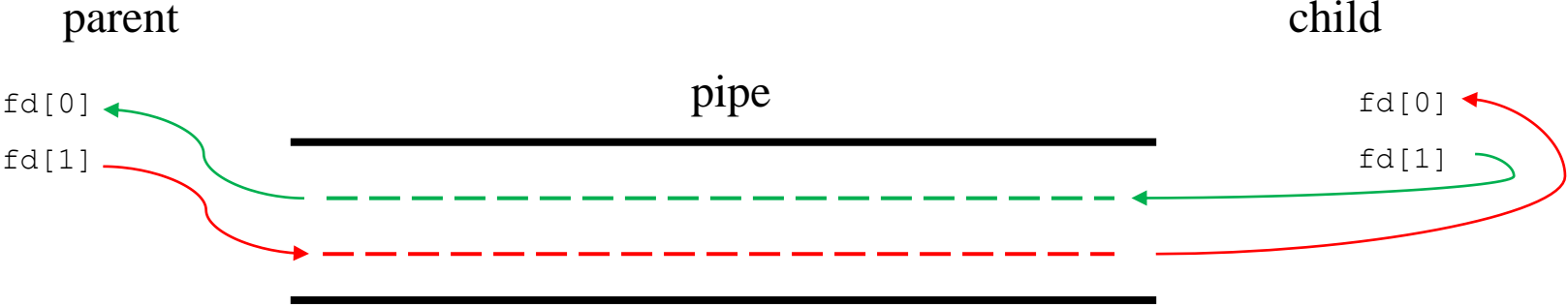
- open the already created file by its name
 - `fd = shm_open(name, flag, mode)`
- map the FD to the memory of process
 - `ptr = mmap(addr, size, prot, flag, FD, offset)`
- read the data from the memory
 - `printf("%s\n", ptr)`
- close the link
 - `shm_unlink(name)`

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/rcv` or mailbox
 - Synchronous or Asynchronous Communication
 - blocking and non-blocking `send/rcv`
 - 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?

?