Operating Systems
5dv171 vt23

Tutorial-I

Monowar Bhuyan
Slides: Sourasekhar Banerjee

UMEÅ UNIVERSITY

Umeå

24 Jan 2023

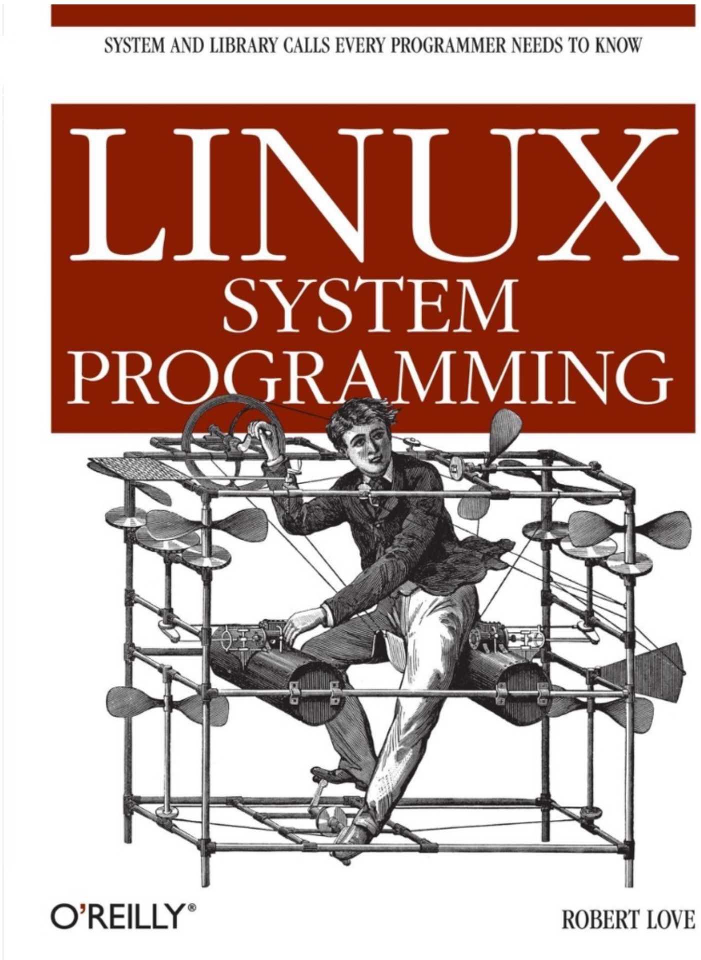
Agenda

- Basic information
- Assignments 1 & 2
- Finalize groups (hopefully)

Tutoring

- Working hours - ask for a time to meet
- Assignment questions
  - Follow assignment-specific discussion pages
- Private/feedback questions: 5dv171vt23-handl@cs.umu.se
- Any information I send out via email will be sent to your cs-email addresses

Useful (Non-mandatory book)



Assignments and Project

- Mandatory
  - Assignment 1: CPU/IO Scheduling
  - Assignment 4: Project LKM
- Bonus-1
  - Assignment 2: Scatter-gather
- Bonus-2
  - Assignment 3: Syscall

Deadlines

Examination/ Assignment(s)/ Project	Deadline 1	Deadline 2	Deadline 3
Written examination	17/3 13:00-17:00	3/5 08:00-12:00	23/8 08:00-12:00
Assignment 1 (mandatory) [Specification]	10/2 23:59	3/3 23:59	9/6 23:59
Assignment 2 (bonus) [Specification]	17/2 23:59	24/3 23:59	9/6 23:59
Assignment 3 (bonus) [Specification]	24/2 23:59	31/3 23:59	9/6 23:59
Project Specification			
Project follow-up [Specification]	10/3 23:59	N/A	N/A
Project final submission [Specification]	21/3 23:59	12/5 23:59	9/6 23:59

Grades

- Grade 3
  - Complete the mandatory assignments
    - Pass assignment 1 and assignment 4 (project)
  - Pass the written exam
- Grade 4
  - Same as grade 3 + complete **one** bonus assignment.
- Grade 5
  - Same as grade 3 + complete **two** bonus assignments.

Groups

- For the mandatory assignment & project
  - Groups of **2 (two)**
- If there is an uneven number of students, then possible to be three in a group
- Preferably all done by today

Hardware and Configuration

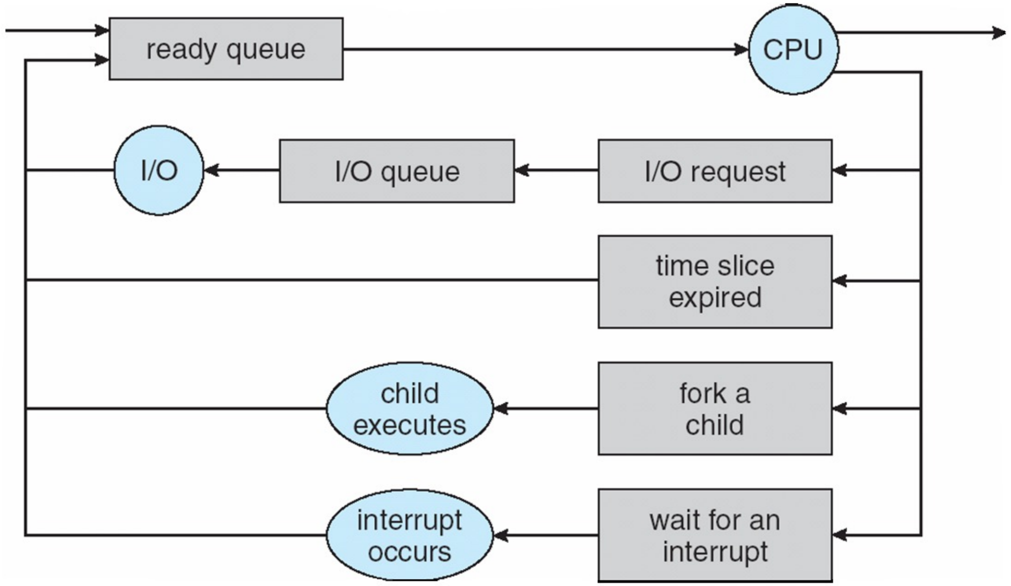
- Use balin.cs.umu.se server (you can find access instructions in Canvas)
- Use VirtualBox in the Linux lab
- Instructions are given in Canvas

- **Note**: You can use VirtualBox outside the Lab (it's up to you)
  - Download the recommended Linux version from the link given in Canvas
  - Contact support for any unfortunate mishandling of system

Assignment 1: CPU/IO Scheduling

- Purpose:
  - Familiarize yourself with the Linux kernel and the scheduler theory
  - Experimentally evaluating how different scheduling algorithms perform in the Linux kernel context
  - How I/O operations impact the system performance?

CPU Scheduling

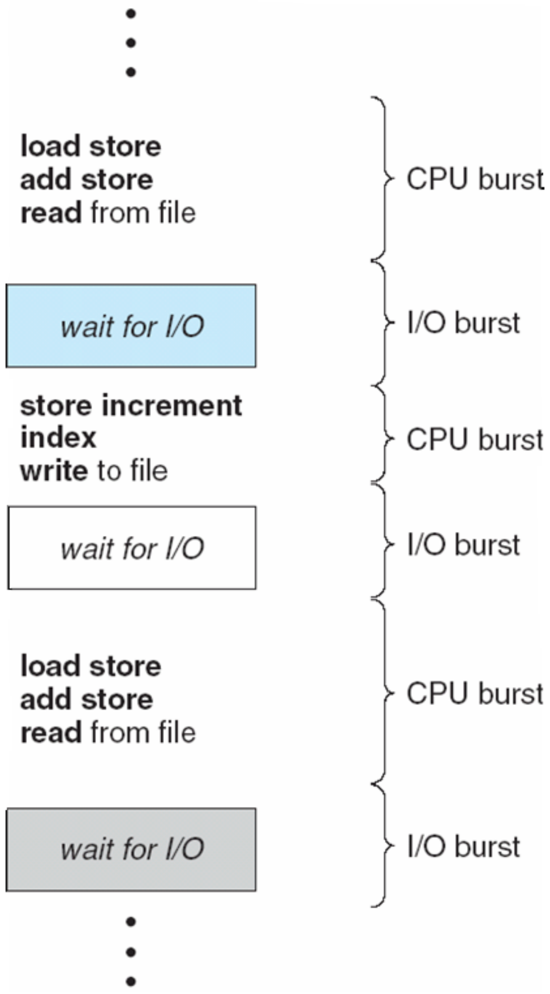


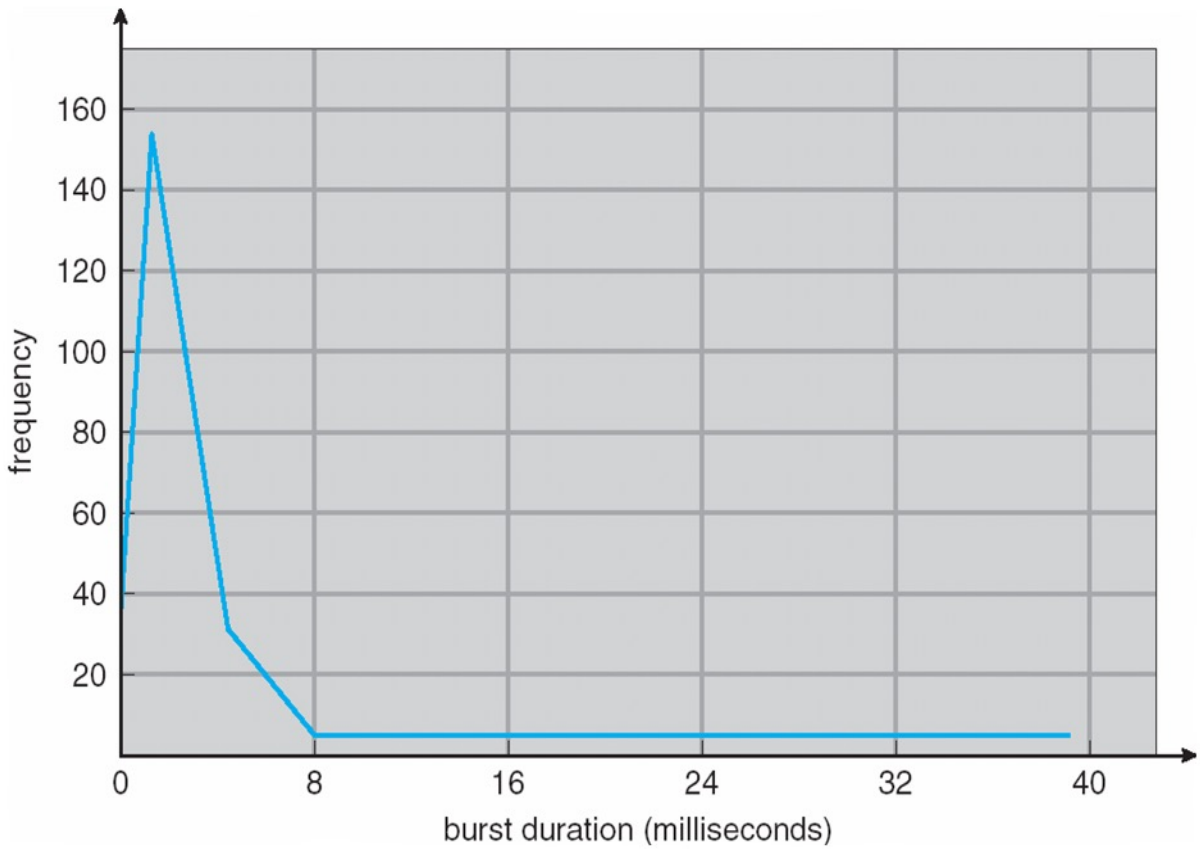
- How is the OS to decide which of several tasks to take off a queue?
- Scheduling: deciding which threads are given access to resources from moment to moment.

CPU Scheduling – Assumptions

- Many implicit assumptions for CPU scheduling:
  - One program per user
  - One thread per program
  - Programs are independent
- These are unrealistic but simplify the problem
- Does "fair" mean fairness among users or programs?
  - If I run one compilation job and you run five, do you get five times as much CPU?
    - Often times, yes!
- Goal: dole out CPU time to optimize some desired parameters of the system.
  - What parameters?

CPU Scheduling – CPU Bursts



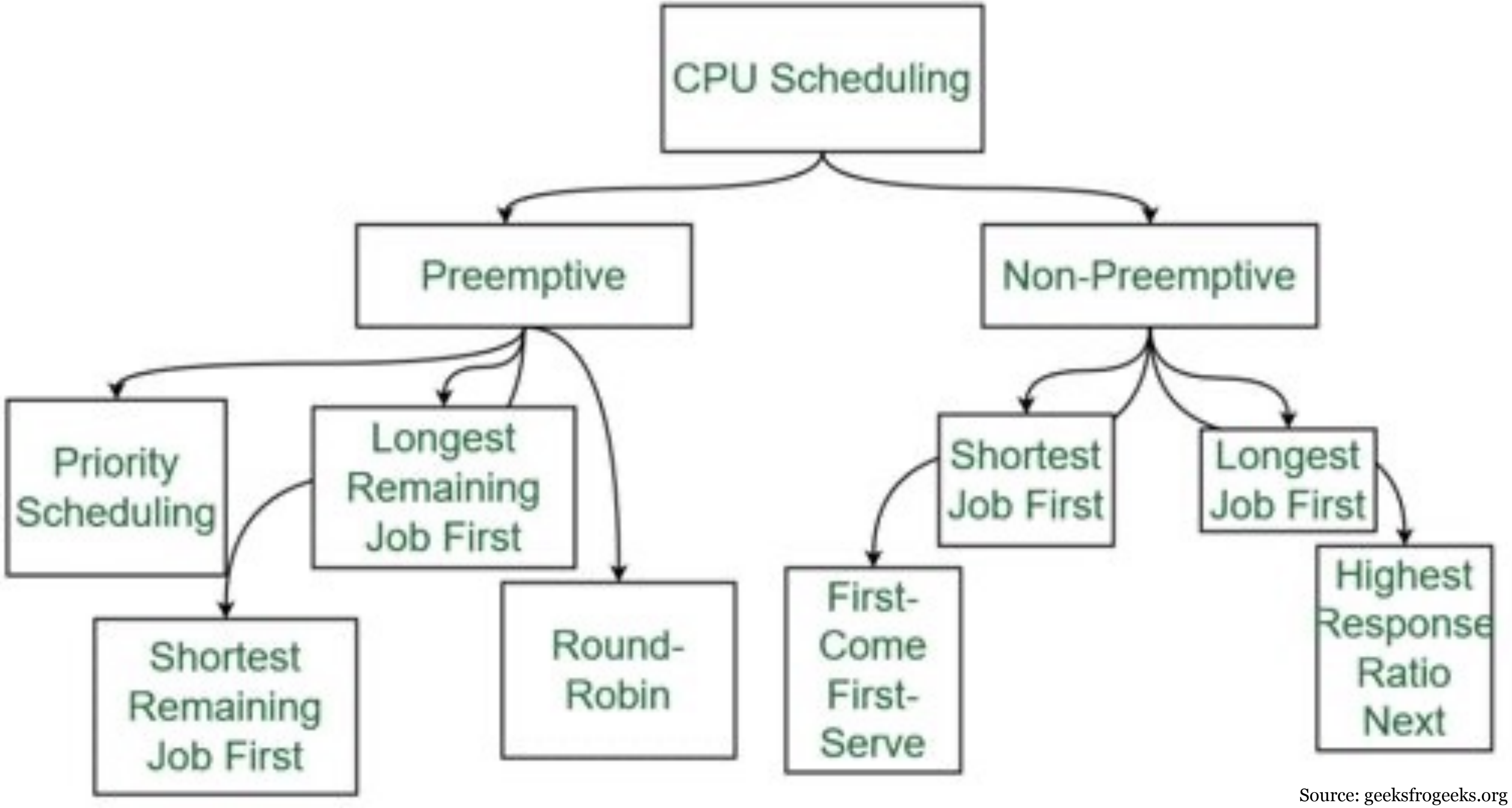


Execution model: programs alternate between bursts of CPU and I/O

CPU Scheduling

- Minimize Response Time
  - Elapsed time to do an operation (job)
  - Response time is what the user sees
    - Time to echo keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines imposed by World
- Maximize Throughput
  - Jobs per second
  - Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
  - Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

CPU Scheduling



Source: geeksfrogeeks.org

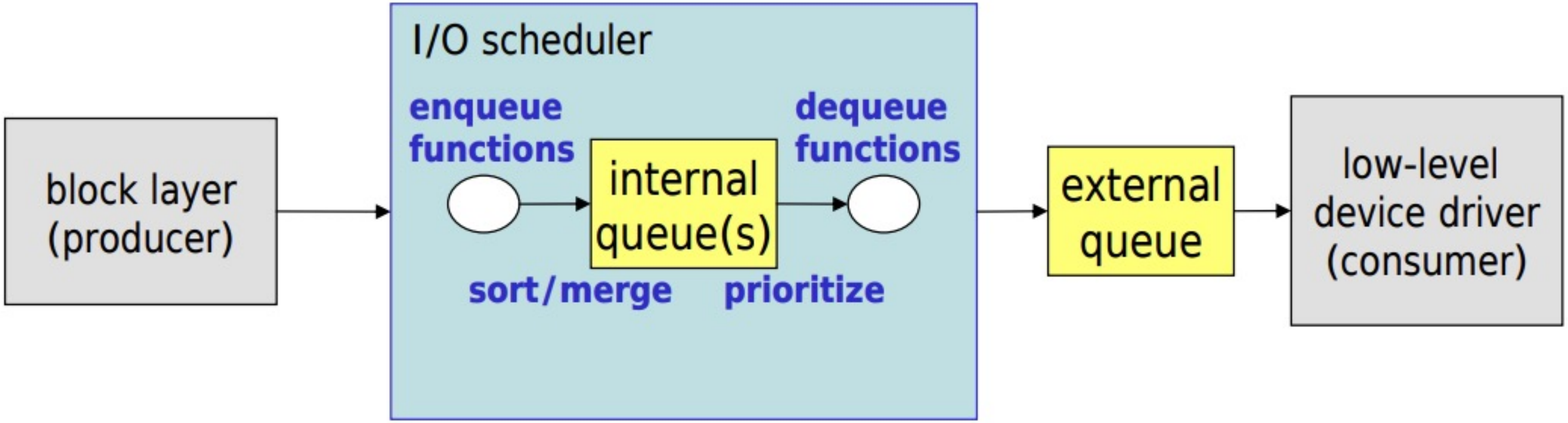
I/O Scheduling

- Disk seek is the slowest operation in a computer
  - A system would perform horribly without a suitable I/O scheduler
- I/O scheduler arranges the disk head to move in a single direction to minimize seeks
  - Like the way elevators moves between floors
  - Achieve greater global throughput at the expense of fairness to some requests

I/O Scheduling

- Improve overall disk throughput by
  - Reorder requests to reduce the disk seek time
  - Merge requests to reduce the number of requests
  - Prevent starvation
  - Submit requests before deadline
  - Avoid read starvation by write
- Provide fairness among different processes

I/O Scheduling



Types of I/O Scheduling

- Noop
  - Suitable for truly random-access device, like flash memory card
  - Requests in the queue are kept in FIFO order
- Deadline
  - Reorder requests to improve I/O performance while simultaneously ensuring that no I/O request is being starved
  - Favor reads over writes
- CFQ (Complete Fairness Queuing)
  - Provide fair allocation of I/O bandwidth among all the initiators of I/O requests
- Anticipatory
  - Disk schedulers reorder available disk requests for
    - performance by seek optimization
    - proportional resource allocation, etc.

Buffering and Caching

- Buffers are temporary storage for raw disk blocks
    - Cache data write to disks, usually not very large (about 20MB).
    - Kernel centralizes scattered writes and optimize disk writes uniformly.
        - For example, multiple small writes can be merged into a single large write, etc.

- Cache is a page cache for reading files from disk, which is used to cache data read from files.
    - Quick fetch the files directly from memory without having to access the slow disk again.

Assignment 1: CPU/IO Scheduling

- Performance evaluation
  - Throughput: How many units of threads/processes finish per unit of time?
  - Latency: On average, how long does it take for an application to finish?
  - Tail Latency: Sorting applications by how long they take, how long does it take for, e.g., the application that is slower than 95% of all other applications to finish?
  - Waiting time: How long does an application stay in the waiting queue on average?
  - You are free to choose more evaluation metrics relevant to the problem
  - Distributions: Plotting the distributions of the above numbers
    - Use box plots - MATLAB/ Python

Assignment 1: CPU/IO Scheduling

- Task 1: CPU scheduling
  - Write a multi-threaded program to test and compare two or more of the Linux CPU scheduling policies available in the main Linux kernel or by others.
  - Your program should run in user space. You should include performance metrics that make sense for your choice of benchmarked algorithms.
  - Your program should be adequate to what you are trying to test and be capable of running for sufficient time to draw conclusions.
  - **sched.h**

- Task 2: I/O synchronization/caching
  - Write a program to test and compare how I/O operations are impacted by the presence or absence of synchronization and caching.
  - You should include performance metrics that make sense for the task.
  - Your program should run in user space and be adequate to what you are trying to test, and should be capable of running for a time sufficient to draw conclusions.
  - **Open flags**
  - **dd**

Assignment 1: CPU/IO Scheduling

- Task 3: I/O scheduling
  - Describe three of the different Linux I/O schedulers.
  - For at least two of the three policies, describe how you think we can improve the policy and optimize the performance even more.
  - Ensure to provide a specific use case where such an improved policy would be valuable.
  - Run the test suite in **Task 2** using at least two different I/O schedulers.
  - Explain any observed differences or reasons for the lack thereof. Describe scenarios where a certain scheduler may be preferable over another.

Example (Assignment 1)

- **Task 1**
  - timespace
    - struct timespace tp {
      time_t tv_sec,
      long tv_nsec };
    - int clock_gettime(clockid_t clk_id, struct timespace *tp);
      - CLOCK_REALTIME : reports actual wall clock-time,
      - CLOCK_PROCESS_CPUTIME_ID : reports amount of CPU consumed by the process.
      - CLOCK_THREAD_CPUTIME_ID : Amount of CPU consumed by the threads.

  - For more informations http://man7.org/linux/man-pages/man2/clock_gettime.2.html

Example (Assignment 1) contd.

- sched_setscheduler(pid_t pid, int policy, const struct sched_param *param)
  - policy is 1 for SCHED_FIFO and 2 for SCHED_RR
  - Follow link : https://man7.org/linux/man-pages/man2/sched_setscheduler.2.html
- Create child process
  - pid_t fork(void)
    - fork() creates a new process by duplicating the calling process. The new process is referred to as the **child process,** and the calling process as the **parent** process.
    - return value: On success, the PID of the child process is returned in the parent, and 0 is returned in the child.
    - On failure -1 is returned in the parent, no child is created, and errno is set to indicate the error.
    - Follow link : https://man7.org/linux/man-pages/man2/fork.2.html
    - Follow link : https://man7.org/linux/man-pages/man3/errno.3.html

Example (Assignment 1) contd.

- You can use pthreads()
  - Useful functions
    - pthread_create() - function starts a new thread in the calling process.
    - pthread_join() - wait for a thread to end.

Example (Assignment 1) contd.

1. Task 1 (Summarized manner)
    1. Create a get_time function which will contain clock_gettime, timespec, etc.
    2. Function for setting new scheduler.
    3. Create a function for computing (e.g., counting n prime numbers, Fibonacci series, matrix multiplication, etc.)
    4. Main function
        1. processes/threads will be created using fork.
        2. The scheduler will be assigned using the set new scheduler function.
        3. Latency will be calculated based on the get_time function.
        4. From there, throughput will be calculated.
        5. From there, the waiting time will be calculated.
        6. From there, the turn-around time will be calculated.

Example (Assignment 1) Contd.

- How to calculate throughput, waiting time, latency, turnaround time
  - Example
    - **Burst time:** CLOCK_THREAD_CPUTIME_ID
    - **Waiting time:** CLOCK_REALTIME - CLOCK_THREAD_CPUTIME_ID
    - **Throughput : (Burst_time_thread1 + … + Burst_time_thread_N)** / N

Example (Assignment 1) Contd.

- **Task 2 & 3**
  - Use dd command
    - dd is a command-line utility for Unix, and Unix-like operating systems' whole primary purpose is to convert and copy files.
  - Options
    - of=FILE (write to a file instead of stdout)
    - bs = BYTES (read and write up to BYTES bytes at a time (default: 512)
    - count = N (copy only N input blocks)
    - iflag = FLAGS (read as per the comma-separated symbol list)
      - fullblock = accumulate full blocks of input
    - oflag =FLAGS (direct/sync/ direct, sync)
      - direct = use direct I/O for data
      - dsync = use synchronized I/O for data
      - nocache, requests to drop cache (oflag=sync)

Example (Assignment 1) Contd.

- **Task 2 & 3**
  - get_io_scheduler()
    - example: cat /sys/block/sda/queue/scheduler
  - set_io_scheduler()
    - example: sudo echo noop > /sys/block/had/queue/scheduler
  - dd command
    - dd if=/dev/urandom of=<FILE> bs=<BYTES> count=<N> iflag=fullblock oflag=<direct/sync/direct,sync>

Note: writing a shell script is enough.
Plot the latency difference between none, direct, sync, and direct+sync and your job is done.

- For Task 2, you don't need to change the I/O scheduler.
- For task 3, you have to apply task 2 in multiple I/O schedulers.

Deliverables

- Deliver a well written report with the following information:
  - Theory.
  - Hypothesis.
  - Method.
  - Results.

Informations

- These sections should contain information covering the solution of all tasks.
- Make sure to include information about what hardware you ran the tests on.
- Submit your code via departments Gitlab or other Git hosting services such as GitHub.
- Include a link in your report, and make sure to grant me read permissions on the repository.
- The assignment is to be done *in pairs of two*.
- Use Canvas for submission.

Assignment 2 (Bonus): Scatter-Gather

- Purpose:
  - Practice programming with respect to
    - concurrency
    - synchronization
    - inter-process communication
    - memory management.

    The focus of the implementation is to understand the components available in the operating system that enables these properties.
- The assignment must be addressed *individually*.
- A passing grade on the assignment will result in a higher grade.

Concurrency

In computer science, concurrency is a property of systems which consist of computations that execute overlapped in time, and which may permit the sharing of common resources between those overlapped computations. (Wikipedia page on concurrency)

- On multiprocessor machines, several threads can execute simultaneously, one on each processor (true parallelism).
- On uniprocessor machines, only one thread executes at a given time. However, because of preemption and timesharing, threads appear to run simultaneously (fake parallelism).

⇒ Concurrency is an issue even on uniprocessor machines!

Synchronization

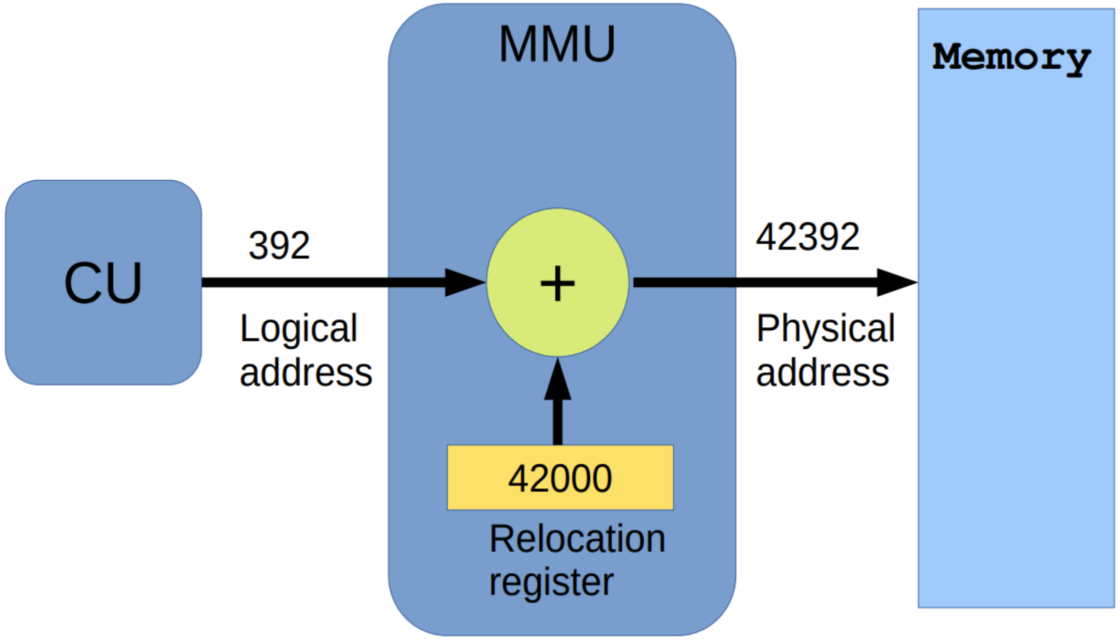
- Concurrent threads can interact with each other in a variety of ways:
  - Threads share access to system devices (through OS).
  - Threads in the same process share access to data (program variables) in their process' address space.
- Common solution when multiple threads access the same data structures
  - Mutual exclusion (MutEx): The shared resource is accessed by at most one thread at any given time.
- The part(s) of the program in which the shared object is accessed is called “critical section”.

IPC

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message) – message size fixed or variable
  - receive(message)
- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

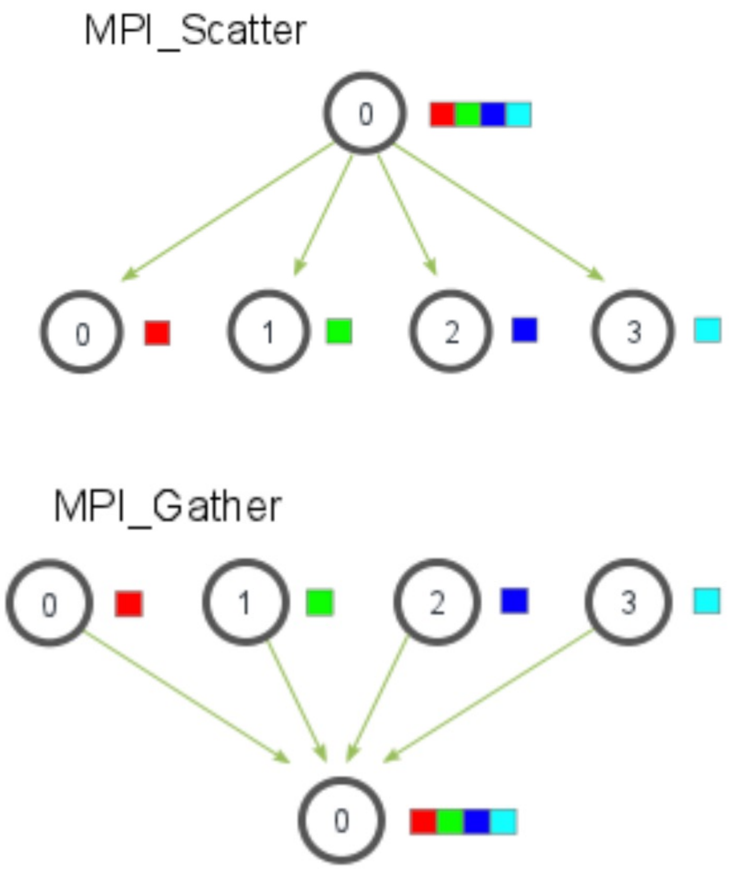
Memory Management

- Process memory separation
- Dynamic creation of processes
- Dynamic memory allocation



What is MPI-Scatter-Gather?

- MPI is a communication protocol for programming parallel computers.
  - It enables communication among processes running on a distributed memory system.
- **MPI_Scatter :** It is a function that splits a piece of data located at a root process into smaller segments and distributes these among a set of processes tasked to operate on the data in parallel.
- **MPI_Gather:** It reverses the action and gathers smaller segments from a set of processes in a way that assembles the entire piece of data at a root process.



Task

- Implement your own scatter and gather functionality.
- Make a couple of design decisions such as, e.g.,
  - Transfer data,
  - Synchronization
  - how processes will maintain references to the other processes in a group
- Tools
  - fork, pipe, socket, read, write….
- Note: You are **not allowed** to solve this assignment using POSIX Threads.

Report

- The report should include an overview of the assignment, and a cover sheet containing (at least):
  - Your name,
  - Username at the CS department,
  - Course code and assignment number.
- Describe the solution, design decisions, assumptions and limitations.
- Extra attention to describing how concurrency, synchronization, inter-process communication and memory management relate to the solution.
- Plot how running times for at least two different computational problems vary with an increasing amount of processes using your implemented functions.
- Any positive, neutral or negative critique regarding the assignment should be included in an enclosing discussion.
- The full report should be no longer than 4 pages, excluding figures (including the cover sheet).
- Use canvas for submission.

Useful information

- Introduction to MPI_Scatter and MPI_Gather: MPI Scatter, Gather, and Allgather
- The Microsoft docs describe the functions very well: MPI Scatter MPI_Gather
- The following Medium article provides some good examples of ways to realize inter-process communication.
- There are many existing algorithms for collective communications that are defined using only the write and read primitives that may interest you. Modifying a one-to-all reduction algorithm, for example, may or may not be useful.
- The technologies you learned during the C-programming and Unix should suffice to solve the assignment.

Reminders

- General assignment questions: Canvas discussion page (link on Canvas)
- Private/feedback questions: 5dv171vt23-handl@cs.umu.se
- Remember to check your cs mail regularly
- All important links regarding the assignments can be found on Canvas

Thank you!!

sourasb@cs.umu.se