CS5600 -Fall 2025
Homework 1
Zongwang Wang

1. Run process-run.py with the following flags: -l 5:100,5:100. What should the CPU utilization be (e.g., the percent of time the CPU is in use?) Why do you know this? Use the -c and -p flags to see if you were right.

The CPU utilisation should be 100%, as 2 processes took turns using the CPU to run each process in an automatic way (running to completion). The result matches my assumption, as CPU bust is at 100%.

```
Time        PID: 0      PID: 1        CPU        IOs
  1       RUN:cpu       READY          1
  2       RUN:cpu       READY          1
  3       RUN:cpu       READY          1
  4       RUN:cpu       READY          1
  5       RUN:cpu       READY          1
  6          DONE     RUN:cpu          1
  7          DONE     RUN:cpu          1
  8          DONE     RUN:cpu          1
  9          DONE     RUN:cpu          1
 10          DONE     RUN:cpu          1
Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy  0 (0.00%)
```

2. Now run with these flags: ./process-run.py -l 4:100,1:0. These flags specify one process with 4 instructions (all to use the CPU), and one that simply issues an I/O and waits for it to be done. How long does it take to complete both processes? Use -c and -p to find out if you were right.

total completion time = 11 time units.
Process 0 took 4 time units, while process 1 took 7 units. This simulation is mimicking the real-world system I/O calls, which are very slow with respect to the running speed of a CPU. Hence, it took 1 unit of 1 for the kernel making the system call doing the I/O operation, 5 units for the process waiting for the I/O operation to be finished, and 1 unit to wrap up once the I/O is done.

```
Time        PID: 0      PID: 1        CPU        IOs
  1       RUN:cpu       READY          1
  2       RUN:cpu       READY          1
  3       RUN:cpu       READY          1
  4       RUN:cpu       READY          1
  5          DONE       RUN:io         1
  6          DONE      BLOCKED                    1
  7          DONE      BLOCKED                    1
  8          DONE      BLOCKED                    1
  9          DONE      BLOCKED                    1
 10          DONE      BLOCKED                    1
 11*         DONE   RUN:io_done           1
Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)
Stats: IO Busy  5 (45.45%)
```

CS5600 -Fall 2025
Homework 1
Zongwang Wang

3. Switch the order of the processes: -l 1:0,4:100. What happens now? Does switching the order matter? Why? (As always, use -c and -p to see if you were right)

Now total completion time = 7 time units.
The switching order does matter, and the order running the 2 processes here is more optimized than before.

At the beginning, process 0's I/O call was made first, while process 1 is ready to run. Then, while process 0 is waiting for the I/O call to be completed, the status of process 0 is now blocked. The scheduler then could start running process 1 while process 0 is waiting. After 4 time units, process 1 is done, but process 0 is still running its I/O operation. Finally, at time 7, process 0's I/O operation is finally done, and 2 processes are now completed. This is an example of how concurrency can actually increase the CPU utilisation while minimizing the down time.

Time	PID: 0	PID: 1	CPU	IOs
1	RUN:io	READY	1	
2	BLOCKED	RUN:cpu	1	1
3	BLOCKED	RUN:cpu	1	1
4	BLOCKED	RUN:cpu	1	1
5	BLOCKED	RUN:cpu	1	1
6	BLOCKED	DONE		1
7*	RUN:io_done	DONE	1	

Stats: Total Time 7
Stats: CPU Busy 6 (85.71%)
Stats: IO Busy 5 (71.43%)

4. We'll now explore some of the other flags. One important flag is -S, which determines how the system reacts when a process issues an I/O. With the flag set to SWITCH ON END, the system will NOT switch to another process while one is doing I/O, instead waiting until the process is completely finished. What happens when you run the following two processes (-l 1:0,4:100 -c -S SWITCH_ON_END), one doing I/O and the other doing CPU work?

Now, as we set SWITCH_ON_END, the order of calling 2 processes no longer affects the overall efficiency, and the completion time slows down to 11 units once again. While process 0 was blocked and waiting for I/O operation, the CPU was idle. Although, it could have better utilized this CPU idle time running process 1.

Time	PID: 0	PID: 1	CPU	IOs
1	RUN:io	READY	1	
2	BLOCKED	READY		1
3	BLOCKED	READY		1
4	BLOCKED	READY		1
5	BLOCKED	READY		1
6	BLOCKED	READY		1
7*	RUN:io_done	READY	1	
8	DONE	RUN:cpu	1	
9	DONE	RUN:cpu	1	
10	DONE	RUN:cpu	1	
11	DONE	RUN:cpu	1	

Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)

CS5600 -Fall 2025
Homework 1
Zongwang Wang
Stats: IO Busy 5 (45.45%)

5. Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O (-l 1:0,4:100 -c -S SWITCH_ON_IO). What happens now? Use -c and -p to confirm that you are right.

The total time is back to 7 again. Now we can really see, the system should better utilize the time while waiting for I/O operation, as I/O operations are taking a significant amount of time until its completion. For example, I/O operations involve hard drive, disks, even SSD and RAM, all of which are much slower compared to the CPU. Instead of letting the CPU stay idle during this waiting period, the operating system can schedule other processes to run, thus improving overall system performance and reducing waiting time.

Time	PID: 0	PID: 1	CPU	IOs
1	RUN:io	READY	1	
2	BLOCKED	RUN:cpu	1	1
3	BLOCKED	RUN:cpu	1	1
4	BLOCKED	RUN:cpu	1	1
5	BLOCKED	RUN:cpu	1	1
6	BLOCKED	DONE		1
7*	RUN:io_done	DONE	1	

Stats: Total Time 7
Stats: CPU Busy 6 (85.71%)
Stats: IO Busy 5 (71.43%)

6. One other important behavior is what to do when an I/O completes. With -l IO RUN LATER, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (Run ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH ON IO -l IO_RUN_LATER -c -p) Are system resources being effectively utilized?

No, the system resources are poorly utilized.The CPU and I/O devices are not kept as busy as they could be, and the total completion time (31 units) is much longer than it would be if I/O completions triggered an immediate switch back to the I/O waiting process. This way, we ensure that each I/O request is issued promptly with RUN:io, and while the I/O operation is in progress, the CPU remains busy executing other processes.

Time	PID: 0	PID: 1	PID: 2	PID: 3	CPU	IOs
1	RUN:io	READY	READY	READY	1	
2	BLOCKED	RUN:cpu	READY	READY	1	1
3	BLOCKED	RUN:cpu	READY	READY	1	1
4	BLOCKED	RUN:cpu	READY	READY	1	1
5	BLOCKED	RUN:cpu	READY	READY	1	1
6	BLOCKED	RUN:cpu	READY	READY	1	1
7*	READY	DONE	RUN:cpu	READY	1	
8	READY	DONE	RUN:cpu	READY	1	
9	READY	DONE	RUN:cpu	READY	1	
10	READY	DONE	RUN:cpu	READY	1	
11	READY	DONE	RUN:cpu	READY	1	
12	READY	DONE	DONE	RUN:cpu	1	

```
 13       READY       DONE       DONE     RUN:cpu          1
 14       READY       DONE       DONE     RUN:cpu          1
 15       READY       DONE       DONE     RUN:cpu          1
 16       READY       DONE       DONE     RUN:cpu          1
 17  RUN:io_done       DONE       DONE        DONE          1
 18      RUN:io       DONE       DONE        DONE          1
 19     BLOCKED       DONE       DONE        DONE                  1
 20     BLOCKED       DONE       DONE        DONE                  1
 21     BLOCKED       DONE       DONE        DONE                  1
 22     BLOCKED       DONE       DONE        DONE                  1
 23     BLOCKED       DONE       DONE        DONE                  1
 24*  RUN:io_done       DONE       DONE        DONE          1
 25      RUN:io       DONE       DONE        DONE          1
 26     BLOCKED       DONE       DONE        DONE                  1
 27     BLOCKED       DONE       DONE        DONE                  1
 28     BLOCKED       DONE       DONE        DONE                  1
 29     BLOCKED       DONE       DONE        DONE                  1
 30     BLOCKED       DONE       DONE        DONE                  1
 31*  RUN:io_done       DONE       DONE        DONE          1
Stats: Total Time 31
Stats: CPU Busy 21 (67.74%)
Stats: IO Busy  15 (48.39%)
```

7. Now run the same processes, but with -I IO RUN IMMEDIATE set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

With -I IO_RUN_IMMEDIATE, as soon as a process finishes its I/O, it gets to run right away instead of waiting its turn. This makes the whole system faster because the CPU never sits idle and the I/O device stays busy too. We can see it in the numbers: everything finishes in 21 time units instead of 31, and the CPU is working 100% of the time. Running the process immediately after its I/O finishes is beneficial because such processes are usually I/O-bound and ready to issue another I/O request quickly.

```
Time        PID: 0       PID: 1       PID: 2       PID: 3          CPU          IOs
  1         RUN:io       READY        READY        READY            1
  2        BLOCKED       RUN:cpu      READY        READY            1            1
  3        BLOCKED       RUN:cpu      READY        READY            1            1
  4        BLOCKED       RUN:cpu      READY        READY            1            1
  5        BLOCKED       RUN:cpu      READY        READY            1            1
  6        BLOCKED       RUN:cpu      READY        READY            1            1
  7*    RUN:io_done        DONE        READY        READY            1
  8         RUN:io        DONE        READY        READY            1
  9        BLOCKED        DONE        RUN:cpu      READY            1            1
 10        BLOCKED        DONE        RUN:cpu      READY            1            1
 11        BLOCKED        DONE        RUN:cpu      READY            1            1
 12        BLOCKED        DONE        RUN:cpu      READY            1            1
 13        BLOCKED        DONE        RUN:cpu      READY            1            1
 14*    RUN:io_done        DONE        DONE         READY            1
 15         RUN:io        DONE        DONE         READY            1
 16        BLOCKED        DONE        DONE         RUN:cpu          1            1
 17        BLOCKED        DONE        DONE         RUN:cpu          1            1
 18        BLOCKED        DONE        DONE         RUN:cpu          1            1
 19        BLOCKED        DONE        DONE         RUN:cpu          1            1
 20        BLOCKED        DONE        DONE         RUN:cpu          1            1
 21*    RUN:io_done        DONE        DONE         DONE             1
Stats: Total Time 21
```

CS5600 -Fall 2025
Homework 1
Zongwang Wang
Stats: CPU Busy 21 (100.00%)
Stats: IO Busy 15 (71.43%)

8. Now run with some randomly generated processes: -s 1 -l 3:50,3:50 or -s 2 -l 3:50,3:50 or -s 3 -l 3:50,3:50. See if you can predict how the trace will turn out. What happens when you use the flag -l IO_RUN_IMMEDIATE vs. -l IO_RUN_LATER? What happens when you use -S SWITCH_ON_IO vs. -S SWITCH_ON_END?

When we run the same workload with different seeds, the seed is used to make the random process generation deterministic and reproducible. Without it, every run could behave differently. By default, -s 0 sets the random seed to 0.

The choice between IO_RUN_IMMEDIATE and IO_RUN_LATER changes how quickly a process that finishes an I/O can return to the CPU. With IO_RUN_IMMEDIATE the process resumes right away, which improves responsiveness but can reduce fairness because it slows other jobs. With IO_RUN_LATER the process goes back into the ready queue and waits its turn, which makes the system fairer but sometimes less responsive.

The switch policies have a bigger effect. SWITCH_ON_IO allows the CPU to be reassigned as soon as a process blocks on I/O, so the CPU stays busy while I/O is happening. SWITCH_ON_END waits until the process's CPU finishes before switching, even if the process is already blocked on I/O. This wastes CPU cycles and leads to longer overall completion times.

Overall the simulation shows that immediate scheduling decisions, both after I/O and after blocking events, tend to keep resources busy and improve responsiveness. In contrast, delaying switches or forcing processes to wait in the ready queue makes the system fairer but less efficient, and it often extends the total time to finish the workload. The bottomline is there is no one single scheduling strategy that maximizes the efficiency in all scenarios, and balancing the trade-off between one strategy with another is what a system programmer like us should be focusing on.

process-run.py -s 1 -l 3:50,3:50 -c -p

Time	PID: 0	PID: 1	CPU	IOs
1	RUN:cpu	READY	1	
2	RUN:io	READY	1	
3	BLOCKED	RUN:cpu	1	1
4	BLOCKED	RUN:cpu	1	1
5	BLOCKED	RUN:cpu	1	1
6	BLOCKED	DONE		1
7	BLOCKED	DONE		1
8*	RUN:io_done	DONE	1	
9	RUN:io	DONE	1	
10	BLOCKED	DONE		1
11	BLOCKED	DONE		1
12	BLOCKED	DONE		1
13	BLOCKED	DONE		1
14	BLOCKED	DONE		1

```
 15*   RUN:io_done          DONE             1
Stats: Total Time 15
Stats: CPU Busy 8 (53.33%)
Stats: IO Busy  10 (66.67%)

 process-run.py -s 2 -l 3:50,3:50 -c -p
Time        PID: 0        PID: 1          CPU          IOs
 1         RUN:io         READY             1
 2        BLOCKED       RUN:cpu             1            1
 3        BLOCKED        RUN:io             1            1
 4        BLOCKED       BLOCKED                           2
 5        BLOCKED       BLOCKED                           2
 6        BLOCKED       BLOCKED                           2
 7*    RUN:io_done       BLOCKED             1            1
 8         RUN:io       BLOCKED             1            1
 9*       BLOCKED    RUN:io_done            1            1
10        BLOCKED        RUN:io             1            1
11        BLOCKED       BLOCKED                           2
12        BLOCKED       BLOCKED                           2
13        BLOCKED       BLOCKED                           2
14*    RUN:io_done       BLOCKED             1            1
15        RUN:cpu       BLOCKED             1            1
16*          DONE    RUN:io_done             1
Stats: Total Time 16
Stats: CPU Busy 10 (62.50%)
Stats: IO Busy  14 (87.50%)

python3 process-run.py -s 3 -l 3:50,3:50 -c -p
Time        PID: 0        PID: 1          CPU          IOs
 1        RUN:cpu         READY             1
 2         RUN:io         READY             1
 3        BLOCKED        RUN:io             1            1
 4        BLOCKED       BLOCKED                           2
 5        BLOCKED       BLOCKED                           2
 6        BLOCKED       BLOCKED                           2
 7        BLOCKED       BLOCKED                           2
 8*    RUN:io_done       BLOCKED             1            1
 9*       RUN:cpu         READY             1
10           DONE    RUN:io_done            1
11           DONE        RUN:io             1
12           DONE       BLOCKED                          1
13           DONE       BLOCKED                          1
14           DONE       BLOCKED                          1
15           DONE       BLOCKED                          1
16           DONE       BLOCKED                          1
17*          DONE    RUN:io_done            1
18           DONE       RUN:cpu             1
Stats: Total Time 18
Stats: CPU Busy 9 (50.00%)
Stats: IO Busy  11 (61.11%)

process-run.py -s 1 -l 3:50,3:50 -l IO_RUN_IMMEDIATE -c -p
Time        PID: 0        PID: 1          CPU          IOs
 1        RUN:cpu         READY             1
 2         RUN:io         READY             1
 3        BLOCKED       RUN:cpu             1            1
 4        BLOCKED       RUN:cpu             1            1
 5        BLOCKED       RUN:cpu             1            1
 6        BLOCKED          DONE                          1
 7        BLOCKED          DONE                          1
 8*    RUN:io_done          DONE             1
```

```
 9        RUN:io         DONE            1
 10       BLOCKED          DONE                    1
 11       BLOCKED          DONE                    1
 12       BLOCKED          DONE                    1
 13       BLOCKED          DONE                    1
 14       BLOCKED          DONE                    1
 15*   RUN:io_done          DONE            1
Stats: Total Time 15
Stats: CPU Busy 8 (53.33%)
Stats: IO Busy  10 (66.67%)
```

process-run.py -s 1 -l 3:50,3:50 -l IO_RUN_LATER -c -p

```
Time        PID: 0        PID: 1          CPU           IOs
 1        RUN:cpu         READY            1
 2         RUN:io         READY            1
 3        BLOCKED       RUN:cpu            1            1
 4        BLOCKED       RUN:cpu            1            1
 5        BLOCKED       RUN:cpu            1            1
 6        BLOCKED          DONE                       1
 7        BLOCKED          DONE                       1
 8*    RUN:io_done          DONE            1
 9         RUN:io         DONE            1
 10       BLOCKED          DONE                    1
 11       BLOCKED          DONE                    1
 12       BLOCKED          DONE                    1
 13       BLOCKED          DONE                    1
 14       BLOCKED          DONE                    1
 15*   RUN:io_done          DONE            1
Stats: Total Time 15
Stats: CPU Busy 8 (53.33%)
Stats: IO Busy  10 (66.67%)
```

process-run.py -s 2 -l 3:50,3:50 -l IO_RUN_LATER -c -p

```
Time        PID: 0        PID: 1          CPU           IOs
 1         RUN:io         READY            1
 2        BLOCKED       RUN:cpu            1            1
 3        BLOCKED        RUN:io            1            1
 4        BLOCKED       BLOCKED                         2
 5        BLOCKED       BLOCKED                         2
 6        BLOCKED       BLOCKED                         2
 7*    RUN:io_done       BLOCKED            1            1
 8         RUN:io       BLOCKED            1            1
 9*       BLOCKED    RUN:io_done            1            1
 10       BLOCKED        RUN:io            1            1
 11       BLOCKED       BLOCKED                         2
 12       BLOCKED       BLOCKED                         2
 13       BLOCKED       BLOCKED                         2
 14*   RUN:io_done       BLOCKED            1            1
 15        RUN:cpu       BLOCKED            1            1
 16*          DONE    RUN:io_done            1
Stats: Total Time 16
Stats: CPU Busy 10 (62.50%)
Stats: IO Busy  14 (87.50%)
```

process-run.py -s 2 -l 3:50,3:50 -l IO_RUN_IMMEDIATE -c -p

CS5600 -Fall 2025
Homework 1
Zongwang Wang

```
Time        PID: 0        PID: 1          CPU          IOs
 1         RUN:io         READY            1
 2        BLOCKED       RUN:cpu            1            1
 3        BLOCKED        RUN:io            1            1
 4        BLOCKED       BLOCKED                         2
 5        BLOCKED       BLOCKED                         2
 6        BLOCKED       BLOCKED                         2
 7*   RUN:io_done       BLOCKED            1            1
 8         RUN:io       BLOCKED            1            1
 9*       BLOCKED   RUN:io_done            1            1
10        BLOCKED        RUN:io            1            1
11        BLOCKED       BLOCKED                         2
12        BLOCKED       BLOCKED                         2
13        BLOCKED       BLOCKED                         2
14*   RUN:io_done       BLOCKED            1            1
15        RUN:cpu       BLOCKED            1            1
16*          DONE   RUN:io_done            1
Stats: Total Time 16
Stats: CPU Busy 10 (62.50%)
Stats: IO Busy  14 (87.50%)
```

```
process-run.py -s 2 -l 3:50,3:50 -S SWITCH_ON_IO -c -p
Time        PID: 0        PID: 1          CPU          IOs
 1         RUN:io         READY            1
 2        BLOCKED       RUN:cpu            1            1
 3        BLOCKED        RUN:io            1            1
 4        BLOCKED       BLOCKED                         2
 5        BLOCKED       BLOCKED                         2
 6        BLOCKED       BLOCKED                         2
 7*   RUN:io_done       BLOCKED            1            1
 8         RUN:io       BLOCKED            1            1
 9*       BLOCKED   RUN:io_done            1            1
10        BLOCKED        RUN:io            1            1
11        BLOCKED       BLOCKED                         2
12        BLOCKED       BLOCKED                         2
13        BLOCKED       BLOCKED                         2
14*   RUN:io_done       BLOCKED            1            1
15        RUN:cpu       BLOCKED            1            1
16*          DONE   RUN:io_done            1
Stats: Total Time 16
Stats: CPU Busy 10 (62.50%)
Stats: IO Busy  14 (87.50%)
```

```
process-run.py -s 2 -l 3:50,3:50 -S SWITCH_ON_END -c -p
Time        PID: 0        PID: 1          CPU          IOs
 1         RUN:io         READY            1
 2        BLOCKED          READY                       1
 3        BLOCKED          READY                       1
 4        BLOCKED          READY                       1
 5        BLOCKED          READY                       1
 6        BLOCKED          READY                       1
 7*   RUN:io_done          READY            1
 8         RUN:io         READY            1
 9        BLOCKED          READY                       1
10        BLOCKED          READY                       1
11        BLOCKED          READY                       1
12        BLOCKED          READY                       1
13        BLOCKED          READY                       1
14*   RUN:io_done          READY            1
15        RUN:cpu         READY            1
```

```
 16        DONE      RUN:cpu          1
 17        DONE       RUN:io          1
 18        DONE      BLOCKED                  1
 19        DONE      BLOCKED                  1
 20        DONE      BLOCKED                  1
 21        DONE      BLOCKED                  1
 22        DONE      BLOCKED                  1
 23*       DONE   RUN:io_done          1
 24        DONE       RUN:io          1
 25        DONE      BLOCKED                  1
 26        DONE      BLOCKED                  1
 27        DONE      BLOCKED                  1
 28        DONE      BLOCKED                  1
 29        DONE      BLOCKED                  1
 30*       DONE   RUN:io_done          1
Stats: Total Time 30
Stats: CPU Busy 10 (33.33%)
Stats: IO Busy  20 (66.67%)
```

9. The Unix Time-Sharing System Download The Unix Time-Sharing Systemby D. M. Ritchie and K. Thompson

Provenance

- Authors: Dennis M. Ritchie and Kenneth L. Thompson.
    - Ritchie is best known for creating the C programming language.
    - Thompson is known for designing Unix and for contributions to the B programming language, which preceded C.
- Discipline: Computer science, specifically operating systems.
- Employer at the time: Bell Labs, part of AT&T.
- Publisher: The paper appeared in Communications of the ACM (CACM), a leading peer-reviewed computer science journal.
- Status of venue: CACM is one of the top venues in CS.

Relevant Literature

- At the time, operating systems research was shaped by Multics (a large, complex timesharing OS).
- The paper cites Unix as a simpler, more practical alternative.
- Cited works mostly discuss timesharing, file systems, and programming environments.

CS5600 -Fall 2025
Homework 1
Zongwang Wang

- Gap identified: There was no simple, efficient, portable OS that could run on modest hardware. Multics was too complex; batch systems too restrictive. Unix filled this gap.

- Significance: Demonstrated that simplicity + powerful abstractions could outperform heavyweight systems.

Research Question

- Implicit question: Can a small, efficient operating system provide powerful functionality (timesharing, hierarchical file system, tools, programming environment) while being portable and widely useful?

- Importance: Addresses how to make computing resources accessible, interactive, and flexible to programmers.

- Evidence needed: Demonstrating a working OS with innovative features, performance measurements, and adoption by users.

- Evidence they present: Architectural design, system calls, file system model, and practical usage examples.

Methodology

- Method: System design and implementation.

- They describe the architecture of Unix, focusing on the file system, processes, and shell.

- They provide concrete system calls and examples of user interaction.

- Methodological strength: Real system, running code, not just a theoretical design.

- Limitations: Few quantitative benchmarks — the paper emphasizes conceptual elegance and practicality more than rigorous performance evaluation.

Analysis

- Techniques: Mostly descriptive system analysis rather than statistics.

- They highlight design choices:

    - Everything is a file (uniform interface).

    - Hierarchical file system.

    - Small modular utilities combined through pipes and redirection.

- Strengths: Clarity of exposition, showing how the design supports extensibility and portability.

- Weakness: Lack of detailed performance metrics compared to other OSs.

Conclusions

- Unix succeeded in being simple, powerful, and portable.

- Key conclusion: By emphasizing simplicity, modularity, and reuse, a small team can build an OS that rivals or surpasses more complex projects like Multics.

- Gaps remaining: Didn't fully address multiprocessor systems, security models, or large-scale distributed environments.

- Future work (implied): Expanding portability (eventually to C), evolving file system, broader adoption.

Critique

- Shortcomings:

    - Minimal quantitative analysis of efficiency compared to contemporary systems.
    - Research question not always explicitly stated.

    - Limited discussion of security and fault tolerance.

- Importance: Despite those shortcomings, the system's actual impact validated its claims.

- Next steps: Later Unix versions, Linux, macOS, etc., show the paper's core ideas are still relevant until today.