Teaching Operating Systems Scheduling

Shimon Cohen
MLA Academic Learning Center, Israel

shamon51@gmail.com

Abstract

The *Operating System* is a very complex program that runs on our computer (probably the most complex …). It is difficult to comprehend the diversity of its operations, let alone – teach it. Second/Third year students with beginner programming skills are overwhelmed by the OS size and its multiple tasks.

This paper is focused on how-to better teach the operation of the OS ***Scheduler*** that manages the user's processes. The scheduler tasks are: create the process, load it into memory, allocate CPU time-slices for its execution, handle keyboard clicks and menu selection, swap the process in and out of memory, and instructs the devices to satisfy IO requests.

The **question** is: "how can we give the students hands-on experience on the scheduler operation? " Obviously, a few slides and a state-machine diagram, is not enough.

The **answer** consists of: explanation (slides), simulation and an exercise.

The **simulation** package includes two simulators (two levels of complexity) each running user-designed scenarios. The 1st simulator (A) is a simple Round-Robin scheduler that is easy to follow. The 2nd simulator (B) comes much closer to a real OS scheduler (with all its complexity); as such it is very difficult for the student to understand, unless they play with Simulator-A first.

The **exercise** goal is to write a "scheduler" that allocates the "CPU" time among ten "Processes" each doing a "long" task. The exercise can be accomplished by 2nd-3rd year students.

Keywords: Operating System, Simulator, Teaching

Introduction

Operating System

The computer is inherently a **parallel** machine; it consists of several CPUs, Memory units, IO Controllers. Each component is in fact a ***basic-computer*** (my definition) with a CPU, logic (program), some memory and IO connectors. For example: a Disk-Controller is made of a CPU (called *Controller*), programmable logic usually stored in EEPROM, memory for data buffers and IO connectors that enables it to control the spinning disk, move the read/write head and communicate with other devices or the mother board. Today these components are fabricated on one chip (the so called "*computer on a chip*")

On top of the computer parallel/real-time hardware architecture, there are

multiple software processes that compete on the use of the computer resources (components). The goal of the Operating System (OS) scheduler is to divide/allocate/arbitrate the use of these resources in the most efficient possible way.

The CPU is the only component that can execute programs; as such it must "wear two hats": (1) executing the multiple users programs, (2) managing the computer using the OS programs (Kernel). To do that, the CPU is constantly doing "context switch" among all its "obligations".

IO controllers are also required to do multiple tasks. For example: a Disk controller is instructed to look for several blocks, on different tracks, for several processes. Or, the network controller must send/receive different packets for multiple browser windows.

Problem / Question / Answer

The students have a **problem** grasping the full complexity of the computer operations. Being 2nd/3rd year students, they write simple programs (C, C++) that run, presumably alone, on the computer without parallelism or threads. The students are not fully aware of the computer real-time nature, and the BIG roll and many services provided by the operating system. They struggle with the idea that the CPU is very fast but also very "dumb", and that it needs very detailed instructions on how and what to execute.

As for the lecturer: presenting slides, writing on the board and being a better lecturer are the first necessary step, BUT usually the students quickly "lose contact" with the material. There are too many new ideas / concepts and they fail to "see" through it.

The **question** is how we can let them "see inside" the Scheduler and "feel" the operation of the OS, CPU, and IO Controllers.

The **answer** is: lecture (in two steps), a pair of simulators (in increasing levels of complexity) and an exercise in writing a small "scheduler". I suggest the following time line:

1. Introduction to OS, Interrupts, Processes States
2. Show the 1st simulator
3. Explain System Calls, Timing and Scheduling
4. Show the 2nd simulator
5. Present the Exercise

Introduction to the OS and the Scheduler

This paper is not about Operating Systems; it is about how to teach OS (in particular the Scheduler). In order to run (and understand …) the two simulators, and do the exercise, the lecturer must cover the following topics:

- Computer architecture: CPU, Clock, BUS, Memory, IO controllers
- CPU: very* [1] quick (speed of light) ←→ very* simple: four arithmetic operations
- Interrupts and System Calls
- Process structure, running, terminating, and Process Control Block (PCB)
- Resource, IO devices, and Resource Control Block (RCB)

[1] Very, very very

- Process States (seven): **new, ready, run, blocked, suspend ready/blocked, exit**
- Scheduling, Timing, Round-Robin algorithm, Queues
- OS tasks:
    - Process creation (linker ➔ "load module")
    - Process loading (loader ➔ memory, DLLs)
    - Process context-switch in/out of the CPU
    - Instructing IO devices
    - Handling devices interrupts
    - Suspending a Process out of memory
    - Reloading suspended process into memory
    - Process termination and cleanup

Simulators (two levels)

The 1st level simulator is a simple scheduler that manages several processes. The processes may need the CPU or service from one of four different IO devices. The simulator shows how the processes are kept in the READY and four IO queues. The scheduler allocates the CPU time among them using the "Round Robin" algorithm. Processes that need service from IO devices are BLOCKED in the IO queues and are serviced according to the First-Come-First-Serve method.

The 2nd level simulator is an elaborate "scheduler" that shows the details of the OS execution, how it loads a process from the disk to memory, performs context switching, schedules IO devices, communicates with IO devices, handles interrupts, receives blocks, swaps process out of memory (suspend state), swaps them in later and finally "finish" the process. The 2nd simulator shows how the IO controller are working in parallel with the |CPU.

Exercise

We define ten vectors (length=1000), and fill them with one thousand random numbers. There are 10 processes that should sort the ten vectors using Bubble sort.

Bubble sort consists of repeated **rounds** on the vector; adjacent elements are compared and swapped if not in order. The algorithm counts the number of swaps in each round, and if the number is zero then it means that the vector is sorted. Using 1000 elements vector with random numbers results on the average with about 120 rounds (although in the worst case scenario it might be 999)

The goal of the project is to write an OS that creates 10 processes (doing bubble sort) and schedules the CPU (execution) among them. Using the Round-Robin algorithm, the OS allocates one sort-round per process to simulate "time-slice".

To simulate using IO we define the following situation: when the process makes more than 500 swaps on the vector it "wants" to "print" it status. In this case the process is taken out of the CPU-queue and placed in the PRINT-queue. When a process is in the first place in the PRINT-queue then it can "print" but it takes 20 time-cycles.

Simulators

This section describes the two simulators that run on top of EXCEL.

Simulator A

	CPU	IO1	IO2	IO3	IO4	P1	P2	P3	P4	P5	P6	P7	P8	P9
Clock	20		CYCLE						RESET					
Work	5	1	1	1	0	2	3	1	2	1	2	1	2	1
IDLE	0	14	15	11	20	17	17	19	17	18	18	18	18	18
CPU	20					3	3	1	3	2	2	2	2	2
IO		6	5	9	0	3	6	0	0	0	6	0	5	0
FINISH						0	0	0	0	0	0	0	0	0

0

Simulator A - by Shimon Cohen

Queues													
P3 1	P6 2	P8 3	P1 4		CPU 3	CPU 2	CPU 2	CPU 3	CPU 4	CPU 2	CPU 6	CPU 2	CPU 3
P5 2	P4 2				IO3 7	IO3 6	IO1 7	IO1 2	IO1 2	IO1 8	IO2 9	IO2 8	IO4 8
P7 4					CPU 2	CPU 4	CPU 3	IO3 8	CPU 1	CPU 1	CPU 1	CPU 2	CPU 2
P9 1					IO2 7	IO2 7	IO2 7	IO2 7	IO2 7	IO2 7	IO1 7	IO4 7	IO2 7
P2 3					CPU 1	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1
					IO3 9	IO3 9	IO3 9	IO3 9	IO3 9	IO3 9	IO3 9	IO3 8	IO3 9
					CPU 2	CPU 2	CPU 2	CPU 2	CPU 2	CPU 2	CPU 2	CPU 2	CPU 2

Line 1 ➔ header with names: one CPU, four IO and nine Processes

Line 2 ➔ The clock (Yellow) and two buttons:

- RESET – to start (or restart) the simulation
- CYCLE – execute one time-cycle

The area below the "Simulator A …" banner

- 1st column – Processes in the "READY" queue, each in need of CPU cycles, the first P3 needs 1 Cycle, the second P5 needs 2 more cycles , …
- Columns 2-5 – show processes waiting for IO service, P6 is serviced by IO1 needs 2 more cycles to finish. Process P4 waits in the queue (needs 2 cycles)..
- Columns 6-16 – show the scenarios of 1-9 processes
    - CPU <n> - needs <n> cycles of CPU
    - IO<a> <n> - needs IO<a> service for <n> cycles

Line "CPU" ➔ number of CPU cycles allocated to each process

Line "IO" ➔ number of IO cycles allocated to each process

Line "IDLE" ➔ number of "idle" cycles of each process

Line "FINISH" ➔ clock-time when process finished its scenario

Line "WORK" ➔

- 1st column "CPU" – which process is serviced (Round Robin)

- 2-5 columns – "1" busy servicing 1st process in queue, "0" – idle
- 6-15 columns – number of item in the scenario (item color is red)

How to use Simulator A

- Setup the processes scenarios – you may leave some of the scenarios empty.
- Click on the "RESET" button
- Click the "CYCLE" button to "execute" one cycle
- Repeat the previous step ("CYCLE" click) until all processes reach the FINISH line.

How to view the screen of Simulator A

- In the beginning all processes need the CPU ➔ all in the "CPU" queue. Each process gets one CPU-Cycle (Round Robin)
- When a process finishes using the CPU it needs IO service, the scheduler moves it to the appropriate IO queue.
- In the IO queue ➔ only the 1st process is serviced, the other wait in line.
- The log (below) is produced automatically by the simulator
    - 1st column – clock
    - 2nd column- time needed for the task
    - 3rd column - Description

17	7	P1 = >>> start IO3
17	4	P2 = $$$ T=4 K=5 put in Q=> CPU
17	0	P2 = <<< finish IO3
17	2	P4 = $$$ T=2 K=2 put in Q IO1
17	0	P4 = <<< finish CPU
16	7	P1 = $$$ T=7 K=2 put in Q IO3
16	0	P1 = <<< finish CPU
15	8	P8 = >>> start IO2
15	8	P8 = $$$ T=8 K=1 put in Q IO2
15	0	P8 = <<< finish CPU
14	8	P6 = >>> start IO1
14	8	P6 = $$$ T=8 K=1 put in Q IO1
14	0	P6 = <<< finish CPU
11	6	P2 = >>> start IO3
11	6	P2 = $$$ T=6 K=1 put in Q IO3
11	0	P2 = <<< finish CPU
0	3	P9 = $$$ T=3 K=9 put in Q=> CPU
0	2	P8 = $$$ T=2 K=8 put in Q=> CPU
0	6	P7 = $$$ T=6 K=7 put in Q=> CPU
0	2	P6 = $$$ T=2 K=6 put in Q=> CPU
0	4	P5 = $$$ T=4 K=5 put in Q=> CPU
0	3	P4 = $$$ T=3 K=4 put in Q=> CPU
0	2	P3 = $$$ T=2 K=3 put in Q=> CPU
0	2	P2 = $$$ T=2 K=2 put in Q=> CPU
0	3	P1 = $$$ T=3 K=1 put in Q=> CPU

Simulator A – demonstrate ?

- How the processes are waiting in the different queues.
- How one CPU is multiplexing its time among the processes using the Round-Robin algorithm.
- How each IO controller serves the 1st process in its queue.
- How processes are moved between the queues by the scheduler.
- NOTE: Operating System tasks (time) are not counted/simulated in Simulator A
- Simulator B is doing the elaborate job of counting and showing the OS tasks.

Simulator B

The simulator screen is divided into the following sections:

- PCBs – [top-left] shows the status, service and other parameters of the nine Processes Control Blocks.
- CPU – [top-right] CPU status and the scenario used (see "Data 7")
    - The scenario itself is shown below
- RCBs – [bottom-right] four (4) Resource Control Block, each RCB shows the status of an IO controller.
- Script – [bottom-left] shows the scenarios used in the simulation.

Simulator B main Screen

CYCLE RESET

PCBs									
ID	1	2	3	4	5	6	7	8	9
Status	Run	S-Block	Block	S-Block	S-Block	S-Ready	S-Block	Block	S-Ready
Service	None	Wait IO	Start IO	Wait IO	Wait IO	None	Wait IO	Start IO	None
Swap Type	None	None	None	None	None	None	None	None	None
Priority	2	2	2	2	2	2	2	2	2
Memory Size	200	300	400	200	300	400	200	300	400
In Memory	Yes	No	Yes	No	No	No	No	Yes	No
Q Number	395	175	359	154	204	94	233	281	124
Top Script	7	6	6	6	6	5	6	6	5
Request									
Request	CPU	IO	IO	IO	IO	CPU	IO	IO	CPU
Request IO	0	2	3	4	1	0	3	4	0
Request Parameter	70	4	4	4	4	30	4	4	30
Request Left	69	1	4	2	4	30	2	4	30
Time									
Start Time	10	30	50	10	30	50	10	30	50
End Time	0	0	0	0	0	0	0	0	0
CPU Use	11	15	30	10	15	0	10	15	0
IO Use	200	197	0	213	30	0	142	0	0
KB Wait	0	0	0	0	0	0	0	0	0
Cur Start	395	378	359	342	365	94	393	281	124
Cur Time	30	0	2	0	0	0	0	2	0
KB Time	0	0	0	0	0	0	0	0	0
Live Time									
CPU %									
Script									
Previous	O1 4	C 15	C 30	C 10	C 15	S 50	C 10	C 15	S 50
Current	C 70	O2 4	O3 4	O4 4	O1 4	C 30	O3 4	O4 4	C 30
Next		C 20	C 90	C 70	C 20	O2 4	C 70	C 20	K 200

CPU		Data	7
Clock	395	100	
Status	RUN		
PCB	1		
Memory	1000		
Free	100		
Run	106	27%	
Kernel	279	71%	
IDLE	10	3%	

RCB	1	RCB	2
Status	WORK	Status	WORK
Per Block	50	Per Block	60
PCB	5	PCB	2
Total T	230	Total T	197
Wait	23	Wait	21
Time	20	Time	43

RCB	3	RCB	4
Status	WORK	Status	WORK
Per Block	70	Per Block	80
PCB	7	PCB	4
Total T	142	Total T	213
Wait	18	Wait	26
Time	68	Time	27

Line "ID" ➔ PCB numbers

Line "Status" ➔ RUN, READY, BLOCKED S-BLOCK (Suspend=Block) …

Line "Service" ➔ Process is waiting for OS service: Wait-IO, Start-IO …

Line "Swap Type" ➔ used when the processes is swap out to a suspend state

Line "Memory Size" ➔ How much memory is needed

Line (4 lines) "Request" ➔ Parameters for the current service request

Line "Time" ➔ Cycles used in various tasks

Line "Cur Start" ➔ Clock when current task started

Line "Cur Time" (in red) ➔ cycles needed to complete current task

<u>RCBs (four gray squares on the right)</u>

- RCB <number>
- Status: Work, IDLE

- Per Block: Time needed to access one block on the disk
- PCB: working for PCB
- Total T: total time working
- Wait: time waiting for OS to handle interrupts
- TIME: Time remains to access the current block

Simulator B Scenario

The scenario instructions are:

- M <size> - size of process in memory
- P <n> - Process priority
- S <cycles> - when to start the process
- C <cycles> - CPU cycles needed at this stage
- O<n> <cycles> - need <cycles> service from IO number <n>

$ P1	$ P2	$ P3	$ P4	$ P5	$ P6	$ P7	$ P8	$ P9
M 200	M 300	M 400	M 200	M 300	M 400	M 200	M 300	M 400
P 2	P 2	P 2	P 2	P 2	P 2	P 2	P 2	P 2
S 10	S 30	S 50	S 10	S 30	S 50	S 10	S 30	S 50
C 10	C 15	C 30	C 10	C 15	C 30	C 10	C 15	C 30
O1 4	O2 4	O3 4	O4 4	O1 4	O2 4	O3 4	O4 4	K 200
C 70	C 20	C 90	C 70	C 20	C 90	C 70	C 20	C 90

How to use Simulator B

- Setup the scenario in a different sheet (in this example “DATA7”)
- Enter the number “7” in the blue cell (top-right)
- Click the “RESET” button to initialize the simulator
- Click the “CYCLE” ➔ Clock is running (one or more cycles) until the next event.

How to view the screen of Simulator A

- Watch what happens with the processes in the first three PCBs lines: “Status”, “Service” and “Swap Type”, **and** the how time of current task “CUR Time” in red is changing.
- Watch the CPU counters on the right
- Watch the RCBs statuses as they work for different PCBs

Simulator B – demonstrate ?

- How the OS is using the CPU in order to service the pCBs, there are various services each need the sole attention of the CPU.
- How the RCBs (IO controller) works in parallel to the CPU

Simulator B parameters

The tables below show various numbers used to control the simulation. For example: if the OS needs to start an IO operation (line 7 “Start IO”) it takes 2 simulator cycles.

Service		Time
None	None	0
Create	Create Process	10
Close	Close Process	8
Start KB	Start KB	1
Wait KB	Suspend if KB >	70
INT KB	Handle KB	1
Start IO	Start IO Operation	2
Wait IO	Suspend if IO >	1
INT IO	Handle IO Interrupt	4
Priority	Priority Change	3
Memory	Memory Change	5

Swap		
None	None	0
Load IN	Load Into Memory	6
L OUT	Load OUT of memory	9
S-IN	Swap Into CPU	2
S-OUT	Swap OUT of CPU	2

Simulator B – Log File

Detailed Log File:

395	30	CPU doing P1 Status=Run T=30
395	30	P1 $ Status is = Run
393	2	CPU doing P1 Status=Ready Swap=S-IN T=2
393	2	P1 * Swap Request is = S-IN
393	70	IO3 handles a Block for P7
393	0	P7 = Service IO Next R3 Blocks=2 of 4
389	4	CPU doing P7 Status=S-Block Service=INT IO T=4
389	0	P1 $ Status is = Ready
387	2	CPU doing P1 Status=Run Swap=S-OUT T=2
387	2	P1 * Swap Request is = S-OUT
387	4	P7 = Service Request is = INT IO
387	0	IO3 Interrupt for P7 !!!
386	30	CPU doing P1 Status=Run T=30
386	30	P1 $ Status is = Run
384	2	CPU doing P1 Status=Ready Swap=S-IN T=2
384	2	P1 * Swap Request is = S-IN
384	0	P1 $ Status is = Ready
378	6	CPU doing P1 Status=S-Ready Swap=Load IN T=6
378	6	P1 * Swap Request is = Load IN
378	60	IO2 handles a Block for P2
378	0	P2 = Service IO Next R2 Blocks=1 of 4
374	4	CPU doing P2 Status=S-Block Service=INT IO T=4
373	4	P2 = Service Request is = INT IO
373	0	IO2 Interrupt for P2 !!!
365	9	CPU doing P5 Status=Block Service=Wait IO Swap=L OUT T=9
365	9	P5 * Swap Request is = L OUT

Conclusion

Undergraduate students find it difficult to comprehend the concepts of the operating systems. There are many new ideas, buzz-words, multiple tasks and “strange” algorithms. Using the proposed method (see below) shows that the students experience and reaction to the course was changed from “dull” / “too complicated” to “interesting” and “challenging”. The use of the simulators coupled with the exercise resulted in much better understanding of the scheduler (as tested in the mid-term)

1. Introduction to OS, Interrupts, Processes States
2. Show the simulator A
3. Explain System Calls, Timing and Scheduling
4. Show the simulator B
5. Exercise

References

Dietel, H. M. (1992). *Operating systems* (2nd ed.). Reading, MA: Addison-Wesley.

Finkel, R.A. (1988). An operating systems vade mecum", 2nd ed. Prentice-Hall, Englewood Cliffs, NJ,.

Goscinski, A. (1991). *Distributed operating systems: The logical design*.

Hartley, S. J. (1994). *Operating systems programming*.

Krakowiak, S. (1988). *Principles of operating systems*. MIT Press.

Lane, M. G., & Mooney, J. D. (1988). *A practical approach to operating systems*. Boyd and Fraser.

Milenkovic, M. (1992). *Operating systems: Concept and design* (2nd ed.). McGraw-Hill.

Silberschatz, A., & Galvin, P.B. (1994). *Operating system concepts* (4th ed.). Reading, MA: Addison-Wesley.

Tanenbaum, A. S. (1987). *Operating systems: Design and implementation*.

Tanenbaum, A. S. (1992). *Modern operating systems*. Englewood Cliffs, NJ: Prentice-Hall.

Tel, G. (2000). *Introduction to distributed algorithms*.

Biography

Shimon Cohen has been working in the Israeli Hi-Tech Industry for the last 30 years, including projects in areas of: Databases, Parallel computing, Artificial Intelligence, E-Learning, Geo-Graphical information, Cellular Communication. Teaching Operating Systems, Object Oriented Programming, Artificial Intelligence, and other Computer Science courses.