

COS214 Tutorial 3

Roberto Togneri, 2000

SOLUTIONS

1. What two advantages do threads have over multiple processes? What major disadvantage do they have? Suggest one application that would benefit from the use of threads, and one that would not.

Answer
 Threads are very inexpensive to create and destroy, and they use very little resources while they exist. They do use CPU time for instance, but they don't have totally separate memory spaces. Unfortunately, threads must "trust" each other to not damage shared data (concurrency and protection problem). For instance, one thread could destroy data that all the other threads rely on, while the same could not happen between processes unless they used a system feature to allow them to share data. Any program that may do more than one task at once could benefit from multitasking. For instance, a program that reads input, processes it, and out-puts it could have three threads, one for each task. "Single-minded" processes would not benefit from multiple threads; for instance, a program that displays the time of day.

2. (a) What resources are used when a thread is created? How do they differ from those used when a process is created?
 (b) Describe the actions taken by a kernel to switch context:
 (i) Among threads.
 (ii) Among processes.

Answer
 (a) A context must be created, including a register set storage location for storage during context switching, and a local stack to record the procedure call arguments, return values, and return addresses, and thread-local storage. A process creation additionally results in memory being allocated for program instructions (text) and data.
 (b)
 (i) The thread context must be saved (registers and accounting if appropriate), and another thread's context must be loaded.
 (ii) The same as (i), plus the memory context (e.g page tables, MMU registers, etc.) must be stored and that of the next process must be loaded.

3. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Calculate the per process and average turnaround times for each of the following scheduling algorithms:

- (a) Shortest-Job-First;
 (b) Nonpreemptive priority (smaller number means higher priority);
 (c) Round Robin with $q=1$.

Assume P1 is at the head of the ready queue and P5 is at the tail.

Answer
 (a) SJF:
 P2 1
 P4 1+1 = 2
 P3 2+2 = 4
 P5 4+5 = 9
 P1 9+10 = 19
 Finish Time = (Start-time + Burst-time)
 Turnaround Time = Finish Time (since Arrival Time = 0)
Ave: $(1+2+4+9+19)/5=7.0$

(b) Nonpreemptive priority (if equal priorities default to FCFS):
 P2 1
 P5 1+5 = 6
 P1 6+10 = 16
 P3 16+2 = 18
 P4 18+1 = 19
 Finish Time = (Start-time + Burst-time)
 Turnaround Time = Finish Time (since Arrival Time = 0)
Ave: $(1+6+16+18+19)/5 = 12.0$

(c) RR($q=1$)

RR →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Process																			
1	X					X			X		X		X		X	X	X	X	X
2		X																	
3			X				X												
4				X															
5					X			X		X		X		X					
Finish		2		4			3							5					1

Ave: $(2+4+7+14+19)/5=9.2$

4. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Arrival Time
P1	10	3
P2	1	1
P3	2	4
P4	1	3
P5	5	0

Calculate the per process and average turnaround times for each of the following scheduling algorithms:

- (a) Shortest-Job-First (SJF);
 (b) Shortest Remaining Time (SRT);
 (c) Round Robin (RR) with $q=3$.

Assume each context switch takes 1 unit of time.

Answer

(a) P5 is scheduled first since it is the only process in the ready queue at $t=0$. When P5 completes the remaining processes have all arrived and are scheduled in the order of SJF: P2, P4, P3, P1 (if equal burst-times default to FCFS):

Finish Time = (Start-time + Context Switch overhead + Burst Time)

Turnaround Time = Finish Time - Arrival Time

($t=0$) P5 0+1+5 = 6 → 6 - 0 = 6
 ($t=6$) P2 6+1+1 = 8 → 8 - 1 = 7
 ($t=8$) P4 8+1+1 = 10 → 10 - 3 = 7
 ($t=10$) P3 10+1+2 = 13 → 13 - 4 = 9
 ($t=13$) P1 13+1+10 = 24 → 24 - 3 = 21

Ave: $(6+7+7+9+21)/5=10.0$

(b) P5 is scheduled first since it is the only process in the ready queue at $t=0$. When P2 arrives it has a shorter remaining time and so it pre-empts P5 at $t=2$ (P5 will then have a burst-time of 4). When P2 finishes at $t=4$ (an extra unit of time is needed for each context switch!), P3 and P4 have arrived, so P4 is scheduled next, then P3, then P5 and finally P1:

Finish Time = (Start-time + Context Switch overhead + Burst Time)

Turnaround Time = Finish Time - Arrival Time

($t=0$) P5 1+1 = 2 (P5 has not yet finished!)
 ($t=2$) P2 2+1+1 = 4 → 4 - 1 = 3
 ($t=4$) P4 4+1+1 = 6 → 6 - 3 = 3
 ($t=6$) P3 6+1+2 = 9 → 9 - 4 = 5
 ($t=9$) P5 9+1+4 = 14 → 14 - 0 = 14
 ($t=14$) P1 14+1+10 = 25 → 25 - 3 = 22

Ave: $(3+3+5+14+22)/5=9.4$

(c)

RR →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...	26	
Process																						
1																						
2					C	X																
3																						
4																						
5	C	X	X	X																		
Arrival	2		1,4	3																		
Finish						2						4			3			5				1

Ave: $([6 - 1] + [12 - 3] + [15 - 4] + [18 - 0] + [26 - 3])/5=13.2$

5. Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

Answer

It will favor the I/O-bound programs because of the relatively short CPU burst requested by them (least processor time); however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively more often to do their I/O.

6. Consider a variant of the Round Robin (RR) scheduling algorithm where the entries in the ready queue are pointers to the Process Control Block's (PCB).
 (a) What would be the effect of putting two pointers to the same process in the ready queue?
 (b) What would be the major advantages and disadvantages of this scheme?
 (c) How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer

- (a) Process appear twice in the ready queue and is scheduled twice as often as other processes.
 (b) *Advantage:* implement priorities. *Disadvantage:* overheads in maintaining pointers; same number of context switches.
 (c) Adaptive quantum for each process. A higher priority process can use up to 2/3/4/etc. quantum of time over the single quantum for normal processes.

7. Consider a system that collects and processes data from two sensors, A and B. The deadline for collecting data from sensor A must be met every 20 ms, and that for B every 50 ms. It takes 10 ms, including operating system overhead, to process each sample of data from A and 25 ms to process each sample of data from B.
 (a) List the execution profile of task A, listing the arrival time, execution time, and completion deadline. Repeat for the execution profile of task B.
 (b) Show how an earliest deadline scheduling policy with preemptive scheduling satisfies the real-time requirements for both tasks.

Answer

(a)

For Task A:

Process	Arrival Time	Execution Time	Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
...	...	10	...

For Task B:

Process	Arrival Time	Execution Time	Deadline
B(1)	0	25	50
B(2)	50	25	100
B(3)	100	25	150
...	...	25	...

(b)

With earliest-deadline scheduling and preemption the scheduling sequence would be:

Start	End	Process	Completion	Deadline	Desc
0	10	A(1)	10	20	A(1) completes
10	20	B(1)	--	--	B(1) pre-empted by A(2) since A(2) has an earlier deadline
20	30	A(2)	30	40	A(2) completes
30	45	B(1)	45	50	B(1) resumes and completes, A(3) arrives at 40
45	55	A(3)	55	60	A(3) completes, B(2) arrives at 50
55	60	B(2)	--	--	B(2) pre-empted by A(3)
...