CST Part IA: Operating System, SV 2
Joe Yan
2017-3-2

1 2014P2Q3

(a) To prevent a process using the CPU forever and preventing other processes making progress. The computer needs a counting-down timer to provide CPU a periodic interruption. Once the CPU receive such an interruption, it stores the state and context of the current running process and load the state and context of the ready process with the highest scheduling priority to the register. To ensure such mechanics works, the timer should prevent any modification or access from user processes.

(b) First we can use extra base and limit register to implement the memory protection. To make this work, the context of a process should have a base physical address and the length of memory it has access to. When the process tries to reference a memory address, the CPU will first add the base address to it and then check whether it is smaller or equal to the limit in the limit register. If the reference is legal the it will be pushed to the bus and sent to memory. Otherwise, a fault is generated and the reference will be rejected. Overall this strategy provides some protections by checking whether every memory reference request is in the range which might be a bottleneck for the performance of CPU.
Second we can use the memory abstraction called virtual address. The computer needs a Memory Management Unit (MMU) to map the virtual address to physical address by paging. Paging will dynamically give a map from a series of continuous virtual addresses (entry) to a chunk of physical frame (typically 4kB). Each entry have protection bit to limit the right of operation of specific process. So it will also provide protection for multiple processes sharing the same physical addresses. To speeding up the paging, a Translation Lookaside Buffer (TLB) will be needed for storing the most recent or frequently used entry in the page and dynamically update them due to some priority algorithms.

(c) The long-term user information will be stored in the file system in the disk. We can use access control or capacity to provide the protection. Capacity might not be suitable because creation or modification to a file may require other users to change their access to this file which is breaking the protection between different users. So access control will need some spaces in a file to store protection bits. (e.g. read, write and executable) To ensure the security when a user process invokes a specific file, the operating system should check the protection information in the header and then decide whether give the process access to the file. User processes should not be able to read or rewrite the protection bit with out the permission from operating system.

(d) Although we no longer need a protection between different users the operating system should still try to give all running processes minimum privilege. e.g the answer in a,b,c discussed, period interruption to CPU for preventing single process using CPU forever or use the memory protection for preventing the bad process read, write or execute the memory they do not have access or use the access control to protect important information not being modified or read. Especially for those untrusted third part software, they may cause damage or steal information if there is no protection at all.

(e) The operating system will have user mode and kernel mode. Like (a) discussed, the system will turn into kernel mode periodically by the implementation of a specific timer who provides periodic interruption. The kernel mode should have full privilege to access the memory, file system, IO devices or network service. So if there is a process invoking some access which previously it does not have the operating system will interrupt and provide a solution. (giving the process a fault exit or asking administer whether modify the access control to this process) This will provide some flexibilities to prevent over-restriction.

2 2013P2Q3

(a) An IO bounded process will only happen when the process needs to wait for IO devices. Its performance is determined by the speed of input or output by other devices instead of the speed of the CPU. So it may have many CPU bursts with small time interval.
An CPU bounded process will happen when there is no IO invoking or the speed of IO devices is much faster than the CPU. Its performance is determined by the speed of CPU and the number of processors it is actually running on. It has a single long CPU burst.

(b) Pre-emptive schedule will have a periodic interruption to CPU and such interruption will turn the CPU into kernel mode and change the state and context to the process with highest priority in the ready data structure. This process does not need the permission and cooperation from the running process. It need a counting down timer to generate the interruption when the timer hits 0.
In non-pre-emptive schedule, the running precess in the CPU will handle the context switching by itself. e.g. the time to give the CPU to other process. This strategy may lose some protection because the process has potential of using the CPU for ever and blocking other ready process.

(c) (i) Add all time together get 23 ms.

(ii) Define R means using CPU, I means doing IO, W means waiting and F means finished.

	A	B
1	R	W
2	R	W
3	R	W
4	R	W
5	I	R
6	I	I
7	R	I
8	R	W
9	R	W
10	R	W
11	I	R
12	I	I
13	R	I
14	R	W
15	R	W
16	R	W
17	F	I

(iii)

	A	B
1	R	W
2	R	W
3	W	R
4	R	I
5	R	I
6	I	R
7	I	I
8	R	I
9	R	Q
10	W	R
11	R	F
12	R	F
13	I	F
14	I	F
15	R	F
16	R	F
17	R	F
18	R	F

(iv) For both scheduling, the process A which takes longer time to run dominated the length of total time cost. But the process B which is the shorter one finished significantly faster in the pre-emptive scheduling.

3 2010P2Q3

(a) The shortest time first scheduler is non-pre-emptive. It always give CPU to the job with the shortest required computing time when the previous job is done.
The shortest remaining time first is pre-emptive. It will give CPU to the job with the shortest remaining time whenever the previous job is done or a new job is added to the scheduler.
The Round Robin is pre-emptive. It assign processes to a time interval which is called quantum. When the time hits the end of the quantum, the context switch will happen to make CPU run the process assigned with the next quantum.

(b) Let T be the waiting time and set time unit to be 5 ms.

(i) 1111134222 (Or 1111143222 also satisfies the definition.)
$T_{p_1} = 0$
$T_{p_2} = 7 - 1 = 6$
$T_{p_3} = 5 - 2 = 3$
$T_{p_4} = 6 - 3 = 3$
$\bar{T} = 3$

(ii) 1234221111
$T_{p_1} = 5$
$T_{p_2} = 2$
$T_{p_3} = 0$
$T_{p_4} = 0$
$\bar{T} = 1.4$

(iii) Consider a queue maintaining the priority information.
Assume the if the job creation and job quantum finishing enqueue happen at the same time, then job creation will enqueue first. Otherwise there might be several different possible answers.

Assume if a job is finished before the end of its quantum, it will call context switch to process the next job in the queue.

	running	queue
1	1	
2	1	2
3	2	31
4	2	314
5	3	142
6	1	42
7	1	42
8	4	21
9	2	1
10	1	

1122311421
$T_{p_1} = 5$
$T_{p_2} = 5$
$T_{p_3} = 2$
$T_{p_4} = 4$
$\bar{T} = 4$

(iv)

	running	queue
1	1	
2	2	1
3	1	32
4	3	241
5	2	41
6	4	12
7	1	2
8	2	1
9	1	
10	1	

$T_{p_1} = 5$
$T_{p_2} = 4$
$T_{p_3} = 1$
$T_{p_4} = 2$
$\bar{T} = 3$

Average waiting will be improved. However there will be twice CPU cost for context switch.
Notice I assumed if the previous job finished before the end of the quantum then call context switch to process the next job in the head of the queue. If I discard this assumption then a 5 ms quantum will be a significant improve for 15 ms less "wasted CPU time".
Because now 10 ms RB looks like:
11223x114x2x1
where every x is a 5 ms wasted CPU time.