CS703 – Advanced Operating Systems - MidTerm Papers – Spring 2016
By: Asif Mansoor

My Todsy's Paper (05-06-2016):

- 10 MCQs
- IPC problem in client-server architecture
- Classes of thread-unsafe
- Uses of threads with examples
- Long-term schedulers handle OS Jobs

Q1. The maximum no. of pages in process address space is one million & total size (p+ d) of process address space is 32 bits with page size of 4096 bytes.
Answer:

p = 12 bits	2^p = 4096
off set =5	2^p = 2^12
==========	p =12
Let the number of bits required = x	2^d = 32
so, 2^x = 4096	2^d =2^5
so, x=12	d =5
offset = 5	
==========	

Q2. An operating system is running three concurrent processes, P1, P2, and P3 for three different users. Each process requires six units of computing time. Process P1 consists of a single thread. Process P2 consists of two threads, each requiring three of P2's six total units of computing time. Process P3 consists of three threads, each requiring two of P3's six total units of time. Assume that the threads never block and that the processes are entirely resident in primary memory.
a. Suppose that the threads are kernel-supported, and that the kernel implements preemptive, round-robin scheduling of threads, with a quantum on one time unit. Of the first six units of time, how many will be devoted to each of the three processes?
Answer:
a. Process 1 will receive one time unit, process 2 will receive 2 units, and process 3 will receive three.
b. Each process will receive two time units.
c. Scheduler (b) is a fair share scheduler. Scheduler (a) is not. Scheduler (a) could be modified to become a two-level scheduler. The top level would select a process to run, using a fair scheduling discipline like round-robin. The bottom level would select one thread from the process that was selected by the top level scheduler.

Q3. What is the way that two processes cannot enter in critical section simultaneously?
Answer:
A solution to the critical section problem must show that
• mutual exclusion is preserved

• progress requirement is satisfied
• bounded-waiting requirement is met.

Q4. Choose the best algorithm for the scenario, and scenario was given. and this q is related to FIFO, SJF, and RR.
Answer:
RR was the best answer of this question. Solve it according to the scenario give.

Q. Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

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. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.
b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.
c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quantums possible in the Round-Robin scheme.

Q. Writ down the main difference between preemitive and non-premitive scheduling schemes and write down at least one from each.

Sol:
Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called preemptive scheduling.
Eg: Round robin

In non-preemptive scheduling, a running task is executed till completion. It cannot be interrupted. Eg First In First Out.

Q. The ctime_ts function is thread safe but not reentrant. Explain your answer with valid reasons?

```
char *ctime_ts(const time_t *timep, char *privatep)
{
char *sharedp;
P(&mutex);
sharedp = ctime(timep);
strcpy(privatep, sharedp); /* Copy string from shared to private */
V(&mutex);
return privatep;
}
```

Answer:
This wraps the call to ctime in a lock that means only one thread can access a call to ctime.
The locked critical section will copy the string pointed to by the static pointer of ctime and copy it into a local non-shared pointer.
Thus, one thread's call to ctime_ts cannot be affected by another thread's call to ctime_ts.
“The ctime_ts function is not reentrant because each invocation shares the same static variable returned by the ctime function.

Q. Describe a mechanism by which one segment could belong to the address space of two different processes.

Answer:

Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base and limit values.

Q. Describe the working of explicit and implicit memory allocators?

Explicit: application allocates and frees space e.g., malloc and free in C
Implicit: application allocates, but does not free space e.g. garbage collection in Java, ML or Lisp.

Q. Describe the differences among short-term, medium-term, and long-term scheduling.

- Short-term (CPU scheduler): selects a process from those that are in memory and ready to execute, and and allocates the CPU to it.
- Medium-term (memory manager): selects processes from the ready or blocked queue and removes them from memory, then reinstates them later to continue running.
- Long-term (job scheduler): determines which jobs are brought into the system for processing.

Q. Process Burst Time Priority

*P*1	10	3
*P*2	1	1
*P*3	2	3
*P*4	1	4
*P*5	5	2

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. What is the turnaround time of each process for each of the scheduling algorithms in part a?

b. What is the waiting time of each process for each of the scheduling algorithms in part a?

c. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

a. Turnaround time

	FCFS	RR	SJF	Priority
*P*1	10	19	19	16
*P*2	11	2	1	1
*P*3	13	7	4	18
*P*4	14	4	2	19
*P*5	19	14	9	6

b. Waiting time (turnaround time minus burst time)

	FCFS	RR	SJF	Priority
*P*1	0	9	9	6

*P*2	10	1	0	0
*P*3	11	5	2	16
*P*4	13	3	1	18
*P*5	14	9	4	1

c. Shortest Job First (3.2 ms).

Q. How is it possible to prevent the deadlocks?

Answer:

There are four necessary conditions which must hold simultaneously for a deadlock to occur. So, deadlock can be prevented by implementing some methodology that guarantees that at least one of the four necessary conditions must not hold. By implementing this kind of strategy, it is possible to prevent a deadlock from occurring. Relaxing first three conditions i.e. Mutual exclusion, Hold & wait, and no pre-emption are not always useful. There may be low utilization of resources & starvation. Circular Wait is the ultimate choice. This is the most suitable condition i.e. ensuring that circular wait never holds. Simple methodology used for this purpose is order all the resources and each process always requests resources in increasing order. Hence, there may never be a process waiting for a low numbered resource holding a higher numbered resource, ensuring that occurrence of cycle is impossible. So, implementing that there will be no circular wait, deadlocks can be prevented.