MANIPAL INSTITUTE OF TECHNOLOGY
(Constituent Institute of MAHE- Deemed University)
MANIPAL-576104

II Test
V Sem B.Tech(CS&E)

Operating Systems and Linux (CSE 309)

TIME : 1 HOUR **29-10-2014** **MAX.MARKS : 20**

Scheme of Evaluation

.

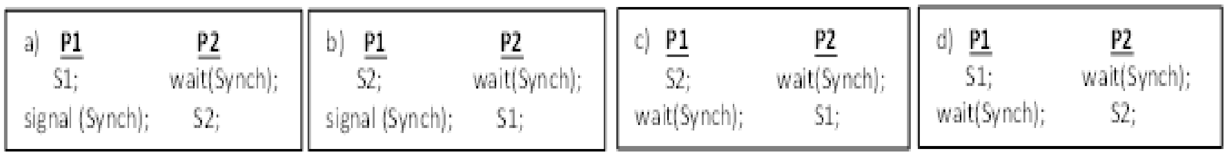
1 Multiple Choice Questions

[Note: Students are required to write down the correct answer completely along with the option selected while answering MCQs]

1A. The number of bits required to represent page offset and page number for a system with logical address of 32-bits and page size of 1024 bytes are

a)22 bits, 10 bits b) 16 bits to each c) 10 bits, 22 bits d)None of them

1B. Consider two processes P1 and P2 running concurrently. It is expected that statement S1 to be executed only after S2 has completed. Use a common semaphore synch which is initialized to 0. Which of the following will meet this?



```
a) P1              P2
   S1;             wait(Synch);
   signal (Synch); S2;

b) P1              P2
   S2;             wait(Synch);
   signal (Synch); S1;

c) P1              P2
   S2;             wait(Synch);
   wait(Synch);    S1;

d) P1              P2
   S1;             wait(Synch);
   wait(Synch);    S2;
```

1C. When a page is selected for replacement, and its modify bit is set means:

a) the page is clean
b) the page in the occupying frame is clean
c) the page has been modified and same copy is in the disk
d) the page has been modified since it was read in from the disk

1D. The disk bandwidth of the disk is :

a) the total number of bytes transferred
b) total time between the first request for service and the completion on the last transfer
c) the total number of bytes transferred divided by the total time between the first request for service and the completion on the last transfer
d) the total time between the first request for service and the completion on the last transfer divided by the total number of bytes transferred.

1E. pthread_join() function in POSIX standard

a) suspend execution of the calling thread until the user intervention

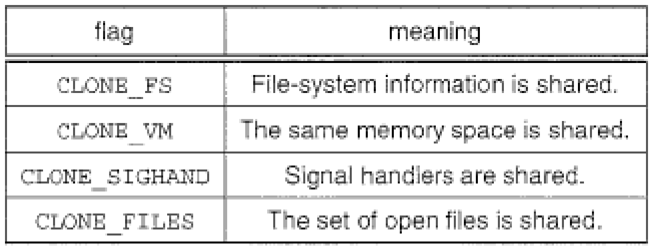
b) suspend execution of the calling thread until the target thread terminates
c) suspend execution of the calling thread until calling thread terminates
d) Never suspends the execution of calling thread. (1X5)

Answers:

1A. c 1B. b 1C. d 1D. c 1E. b

2A. How threads are implemented in Linux? Explain.

Ans: Linux provides the fork() system call with the traditional functionality of duplicating a process. Linux also provides the ability to create threads using the clone() system call. However, Linux does not distinguish between processes and threads. In fact, Linux generally uses the term task-rather than *process* or *thread-when* referring to a flow of control within a program. When clone() is invoked, it is passed a set of flags, which determine how much sharing is to take place between the parent and child tasks. Some of these flags are listed below:



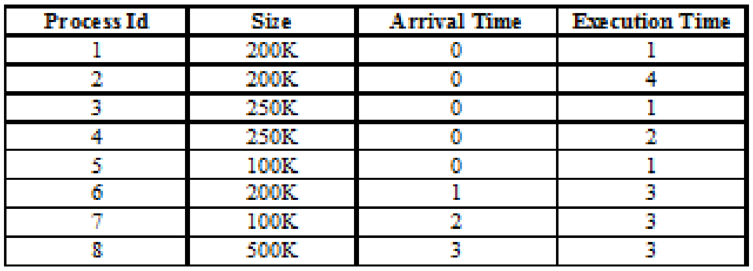
flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

(1)

For example, if clone() is passed the flags CLONE_FS, CLONE_VM, CLONE_SIGHAND, and CLONE_FILES, the parent and child tasks will share the same file-system information (such as the current working directory), the same memory space, the same signal handlers, and the same set of open files. Using clone() in this fashion is equivalent to creating a thread, since the parent task shares most of its resources with its child task. However, if none of these flags is set when clone() is invoked, no sharing takes place, resulting in functionality similar to that provided by the fork() system call.

The varying level of sharing is possible because of the way a task is represented in the Linux kernel. A unique kernel data structure (specifically, struct task_struct) exists for each task in the system. This data structure, instead of storing data for the task, contains pointers to other data structures where these data are stored -for example, data structures that represent the list of open files, signal-handling information, and virtual memory. When fork()is invoked, a new task is created, along with a *copy* of all the associated data
structures of the parent process. A new task is also created when the clone() system call is made. Howevet~ rather than copying all data structures, the new task *points* to the data structures of the parent task, depending on the set offlags passed to clone(). (1)

2B. Consider a user memory of size 1000K bytes which is initially empty. List of processes(in order) waiting for memory allocation is given along with their arrival time and execution time. Under variable- partition scheme, how would best-fit and worst-fit algorithms place the processes? Show the content of user memory across the time units t0, t1, .. tn, until 500K process is allocated. (2)+(1.5+1.5))



Process Id	Size	Arrival Time	Execution Time
1	200K	0	1
2	200K	0	4
3	250K	0	1
4	250K	0	2
5	100K	0	1
6	200K	1	3
7	100K	2	3
8	500K	3	3

Ans:

Best-Fit

T0			**T1**			**T2**			**T3**		
Block	**Size**	**Pid**	**Block**	**Size**	**Pid**	**Block**	**Size**	**Pid**	**Block**	**Size**	**Pid**
1	200	1	1	200	6	1	200	6	1	200	6
2	200	2	2	200	2	2	200	2	2	200	2
3	250	3	3	250		3	500		3	500	8
4	250	4	4	250	4	4	100	7	4	100	7
5	100	5	5	100							

Worst –Fit (1.5 + 1.5): T0 -- .5; T1,…Tn. – 1

T0			T1			T2			T3			T4		
Block	Size	Pid	Block	Size	Pid	Block	Size	Pid	Block	Size	Pid	Block	Size	Pid
1	200	1	1	200		1	200		1	200		1	500	8
2	200	2	2	200	2	2	200	2	2	200	2	2	150	
3	250	3	3	200	6	3	200	6	3	200	6	3	100	7
4	250	4	4	50		4	50		4	50		4	250	
5	100	5	5	250	4	5	100	7	5	100	7			
			6	100		6	250		6	250				

3A**.** Assume the amount of memory on a system is inversely proportional to the page fault rate. Each time memory doubles, the page fault rate is cut in half. Currently the system has 32MB of memory. When a page fault occurs the average access time is 1ms, and 1μs otherwise. Over all, the effective access time is 300μs.

i. What is the page fault rate for the above system? What is the page fault rate if the effective access time is 80 μs?
ii. How much memory would be needed to make the effective access time to 80μs? Assume the total memory in the system to be power of 2.

Ans

i) The current page fault rate can be computed as

Effective access time = (1 – p) normal access + p * page fault access time

300 = (1-p) * 1 + p * 1000 //computation step = 0.25 mark

p = 29.93% or 0.0293 // Result 0.25 mark

ii) The page fault rate necessary for a 100µs access time can be computed as

80 = (1-p)*1 + p*1000 //computation step = 0.25 mark

p = 7.90 % or 0.0790 //calculate new p = 0.25 mark

iii) The actual page fault rate

29.93/ 2 = 14.96; 14.96/2 = 7.48

must be halved twice to 7.48% to get it under 7.90%. // explanation 0.5

Memory must be double twice to be increased to 128MB // calculate memory 0.5 mark

3B. Explain the following concepts with the help of a diagram

i. Steps to handle page fault in Demand Paging

ii. Second-Chance algorithm ((1+1)+(2+1))

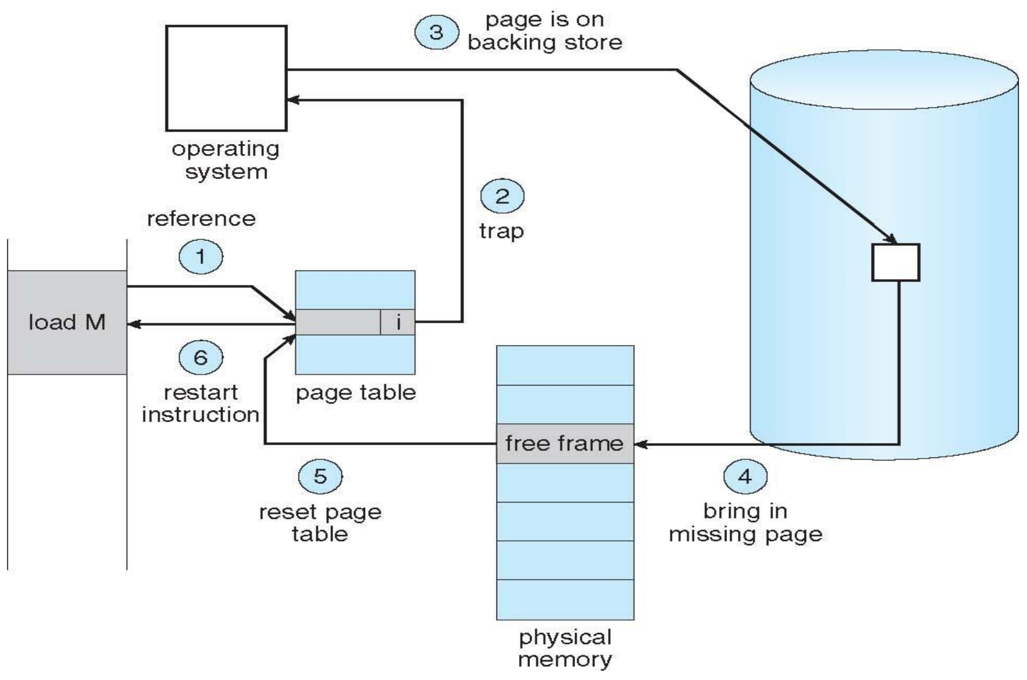
Ans:

i) i. The procedure for handling page fault is straight forward

1) We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or invalid memory access
2) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
3) we find a free frame(by taking one from the free frame list)
4) We schedule a disk operation to read the desired page into the newly allocated frame
5) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory
6) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory

// 0.25 mark for each point in the same order = 0.25 * 6 = 1.5 marks

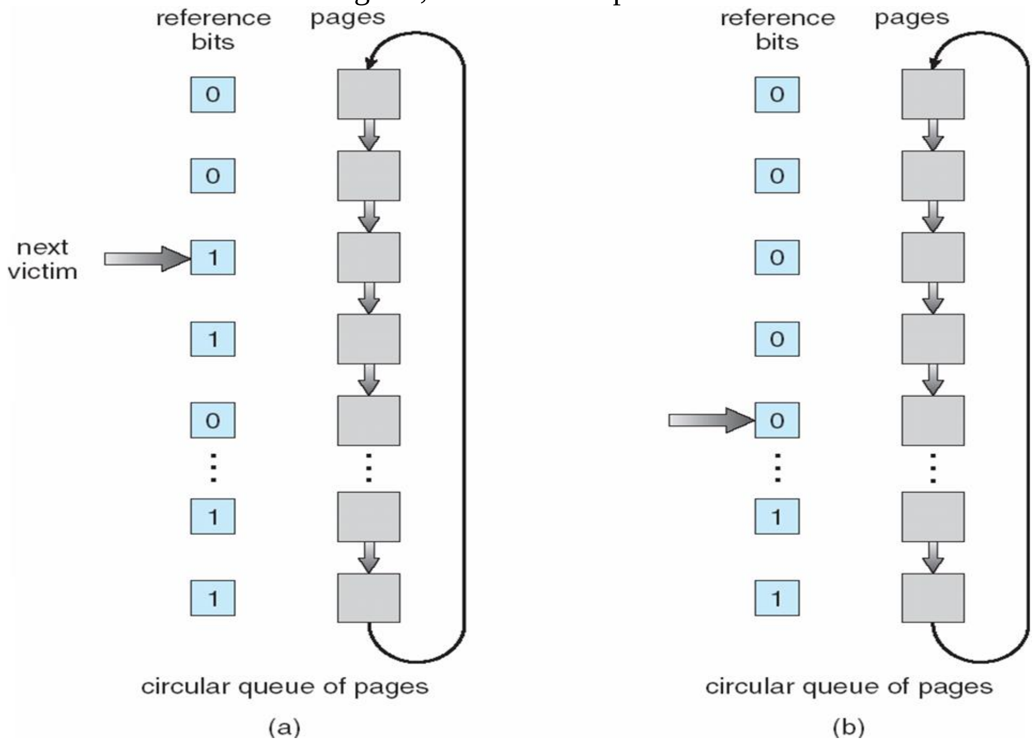
//0.5 mark - diagram



ii. Second chance algorithm is a version of FIFO replacement algorithm. When a page has been selected, we inspect its reference bit. If the value is 0, we replace the page. But if the value is 1, we give the page a second chance and move on to select the next FIFO page.

When a page gets a second chance, its reference bit is cleared and its arrival time is reset to current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chance)

//0.5 mark – diagram; 0.5 mark – explanation



4A. Suppose that a disk drive has 5000 cylinders(0 – 4999). The drive is currently serving a request at cylinder 143, and the previous request was 125. The queue of pending requests, in FIFO order, is: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance in cylinders that the disk arm moves to satisfy all the pending request for SCAN disk scheduling algorithm. Represent the head movements diagrammatically.

Ans:

Ans: Diagramatically the head moves from

143 → 913 → 948 → 1022 → 1470 → 1509 → 1750 → 1774 → 4999 →130 → 86 1M

Total Head Movements: 770 + 35 + 74 + 448 + 39 + 241 + 24 + 3225 + 4869 + 44 =9769 Cylinders. 1M

4B. Specify the three requirements which need to be satisfied by a critical section problem. Provide a algorithm using TestAndSet() hardware instruction which satisfies the above requirements. (2+(1+2))

The three requirements which need to be satisfied by a critical section problem are:

Mutual Exclusion, Progress and Bounded Wait. 1M

The algorithm using TestAndSet() instruction is as follows:

```
do {
        waiting[i] = TRUE;
        key = TRUE;
        while (waiting[i] && key)
            key = TestAndSet(&lock);
        waiting[i] = FALSE;
            // critical section
        j = (i + 1) % n;
        while ((j != i) && !waiting[j])
            j = (j + 1) % n;
        if (j == i)
            lock = FALSE;
        else
            waiting[j] = FALSE;
            // remainder section
    } while (TRUE);                                   2M
```

****