Indian Institute of Technology, Kharagpur
End-Spring Semester 2017-18

Date of Examination: 24-04-2018 **Session:** AN (2-5 pm) **Duration:** 3 hrs
Subject No.: CS31702
Subject: COMPUTER ARCHITECTURE AND OPERATING SYSTEMS
Department/Center/School: Computer Science and Engineering
Specific charts, graph paper, log book etc., required: NO **Total Marks :** 100
Special instructions (if any): ANSWER ALL QUESTIONS
Note: All parts of the question (a,b,c,....) should be answered at a stretch.

1. (a) **What is meant by dual mode of operation in the context of execution of a process?**
User mode: Instructions in user programs will be executed.
Kernal mode: Execution of system calls/interrupts/kernal operations/...

(b) **What are the services offered by operating system from system point of view?**
Resource allocation, accounting and protection-cum-security.

(c) **What is direct memory access (DMA)?**
Used for high-speech I/O devices to transfer blocks of data between memory and devices. CPU will not intervene for each byte/word, but CPU will be informed at the end of each block transfer through interrupt.

(d) **With appropriate example discuss how user program, API, system call interface, system call and operating system are related?**
printf() calls standard C-library (API) (user program through API calls C-library function printf())
C-Library calls system call write() (C-lib function will call a system call write() through system call interface)
Operating system will execute write() system call and the result will be return to API (C-lib) through system call interface. API inturn retuns the result to user program.

(e) **What is core-dump and crash-dump? Where these are used?**
Core-dump: Failure of an application can generate core dump file capturing memory of the process
Crash-dump: Operating system failure can generate crash dump file containing kernel memory
These files are used to analyse the failures of user program code and operating system codes.

(f) **With a neat sketch (process stete diagram), explain various states the process may undergo from its creation to its termination. Mark all possible transitions and states and clearly explain the details of states and state-transitions.**
States:
new: The process is being created
running: Instructions are being executed
waiting: The process is waiting for some event to occur
ready: The process is waiting to be assigned to a processor
terminated: The process has finished execution

Events for state transitions:
Admitted: New - ready
Interrupt: Running - ready
Scheduler dispatch: Ready - running
I/O or event wait: Running - waiting
I/O or event completion: Waiting - Ready
Exit: Running - terminated

(g) **Briefly discuss the pros and cons of shared memory and message passing schemes in the context of inter-process communication.**

Shared Memory: Less delay (faster access), less load on operating system, It is convenient for processes sharing the information within a system, Symultaneous modifications on the shared data to be avoided.

Message passing: Efficient in case of passing the messages betwen the processes present in different systems. Generally used for exchanging small pieces of information. It is slow compared to shared memory. The entire communication takes palce through system calls and hence OS is heavily loaded.

(h) **Discuss the salient features of ordinary pipes in the context of inter-process communication.**

Unidirectional communication
Parent-child relationship is required for communication

(1+1+1+2+2+4+2+2 = 15M)

2. (a) **How process and thread creations differ? Why threads are considered as light-weight-processes?**

New process is created in separate address space. Whereas threads are created within the address space of a process. Only CPU-registers and stack are specific to each thread and the rest are common to all threads.

As threads are just an executional entities and deals with limited resources such as stack, CPU state and PC, it can be viewed as light-weight-process. Where as process has two roles: execution as well as maintaining all its resources (Address space,,Global variables,Open files, Child processes, Pending alarms Signals and signal handlers, Accounting infor mation).

(b) **Mention various multi-threading models, and discuss their features.**

Many-to-One, One-to-One, Many-to-Many and Two-level

(c) **Briefly explain the following in the context of threads: (i) Signal Handling and (ii) Thread pools.**

Signal handling: (i) Deliver the signal to the thread to which the signal applies (ii)Deliver the signal to every thread in the process (iii) Deliver the signal to certain threads in the process and (iv) Assign a specific thread to receive all signals for the process

Thread pools: Create a number of threads in a pool where they await work

Advantages: (i) Usually slightly faster to service a request with an existing thread than create a new thread (ii) Allows the number of threads in the application(s) to be bound to the size of the pool

(d) **What is preemptive and non-preemptive scheduling mechanisms? Provide an example to each.**

Non-preemptive scheduling: When the process is scheduled to CPU for its execution, it will continue to execute till its termination or some I/O requirement. (FCFS, SJF)
Preemptive scheduling: The scheduled process will be supended in case of any interrupt/signal/event (ex: round-robin scheduling, priority-based preemptive scheduling)

(e) **Mention various parameters (criteria) considered for CPU scheduling.**
CPU utilization, throughput, turn-around time, waiting time and response time

(f) **Briefly discuss the following in the context of multi-processor scheduling: (i) processor affinity, (ii) soft affinity, (iii) hard affinity, (iv) push migration and (v) pull migration.**
Processor affinity: process has affinity for processor on which it is currently running. Otherwise, if we migrate a process from processor-i to processor-j, then lot of over head is involved in clearing the cache of processor-i and cache of processor-j has to be re-populated.
soft affinity: OS want to keep the process on the same processor which it has affinity, but it will not gaurantee to do that always.
hard affinity: Some systems using some system calls ensure that the process will be atached to the desired processor always.
push migration: Periodically checks the load on each processor and if it finds an imbalance, it evenly distributes the load by moving (or pushing) processes from overloaded to idle or less-busy processors.
pull migration: An idle processor pulls the processes from the ready queue of a heavily loaded (busy) processor.

(g) **Consider the following set of processes, with the length of the CPU burst given in milliseconds:**

Process	Burst Time (ms)	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

i. **Draw four Gnatt charts that illustrate the execution of these processes using the folloing scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a highest priority), and RR (quantum = 2).**

ii. **What is the turnaround time of each process for each of the scheduling algorithms in part-i?**
FCFS: T1 = 2, T2 = 3, T3 = 11, T4 = 15 and T5 = 20
SJF: T1 = 3, T2 = 1, T3 = 20, T4 = 7 and T5 = 12
NPP: T1 = 15 (19), T2 = 20, T3 = 8, T4 = 19 (17) and T5 = 13
RR: T1 = 2, T2 = 3, T3 = 20, T4 = 13 and T5 = 18

iii. **What is the waiting time of each process for each of these scheduling algorithms?**
FCFS: W1 = 0, W2 = 2, W3 = 3, W4 = 11 and W5 = 15; Wav = 6.2
SJF: W1 = 1, W2 = 0, W3 = 12, W4 = 3 and W5 = 7; Wav = 4.6

NPP: W1 = 13 (17), W2 = 19, W3 = 0, W4 = 15 (13) and W5 = 8; Wav = 11 (11.4)
RR: W1 = 0, W2 = 2, W3 = 12 W4 = 9 and W5 = 13; Wav = 7.2

iv. **Which of the algoritms results in the minimum average waiting time (over all processes)?**
SJF

(2+2+2+2+2+5+8 = 23M)

3. (a) **Define a critical section problem. Specify the requirements to be satisfied by the solution to the critical section problem.**
If multiple processes have a shared section of code, in which the shared code should be executed by only one process at any point of time, then the problem of handling the shared code executed by only one process is know as critical section problem.
Requirements to be satisfied by the solution to the critical section problem: (i) mutual exclusion, (ii) progress and (iii) bounded waiting.

(b) **Provide the codes correspond to entry and exit sections of a process Pi in view of Peterson solution to critical section problem.**
Entry section:flag[i] = TRUE; turn = j; while (flag[j] && turn == j);
Exit section: flag[i] = FALSE;

(c) **Write the pseudo code with TestAndSet(&lock) to solve the critical section problem addressing the issues of bounded-waiting and mutual-exclusion.**

```
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);
```

(d) **What is a semaphore? What are the standard operations that modify semaphore value? How semaphore handles the critical section problem using entry and exit sections? Provide the detailed code segments for the standard semaphore operations in case of buzy-waiting and without buzy-waiting?**
Semaphore is a synchronization tool. It is an integer variable, except initialization it is used to access by two atomic operations wait() and signal().
Handling critical section by semaphore:

```
do {
```

```
wait (S) ;
critical section
signal(S);
remainder section
} while (TRUE);
Semaphore operations (with buzy-waiting):
wait(S) {
while (S <= 0) ; no-op
S−−;
}
signal(S) {
S++;
}
Semaphore operations (without buzy-waiting):
wait(semaphore *S) {
S− >value−−;
if (S− >value < 0) {
add this process to S− >list;
block();
}
}

signal(semaphore *S) {
S− >value++;
if (S− >value <= 0) {
remove a process P from S− >list;
wakeup(P);
}
}
```

(e) Illustrate the problem of deadlock using pair of processes want to access a pair of critical sections.

Suppose a pair of processes P1 and P2 want to access a pair of critical sections C1 and C2 which are protected by S1 and S2 semaphores. If the sequence of operations by the processes are as follows:

```
S1 = S2 = 1;
P1: wait (S1);
P2: wait (S2);
P1: wait (S2);
P2: wait (S1);
.....
.....
P1: signal (S1);
P2: signal (S2);
P1: signal (S2);
P2: signal (S1);
```

(f) mention the necessary conditions to be satisfied for deadlock.

Mutual exclusion, hold-and-wait, no preemption and circular wait

(g) **Consider the following snapshot of a system. P0, P1, P2, P3, P4 are the processes and A, B, C, D are the resourse types. The values in the table indicates the number of instances of a specific resource (for example: 3 3 2 1 under the last column indicates that there are 3 A-type, 3 B-type, 2 C-type and 1 D-type resources are available after allocating the resources to all five processes). The numbers under allocation-column indicate that those number of respources are allocated to various processes mentioned in the first column. The numbers under Max-column indicate the maximum number of resources required by the processes. For example: in 1st row under allocation-column 2 0 0 1 indicate there are 2 A-type, 0 B-type, 0 C-type and 1 D-type resources are allocated to process P0. Whereas 4 2 1 2 under Max-column indicate that process P0's maximum requirement is 4 A-type, 2 B-type, 1 C-type and 2 D-type resources. Answer the fol-**

Process	**Allocation**	**Max**	**Available**
	A B C D	A B C D	A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 1 2	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

lowing questions using banker's algorithm by providing all intermediate steps:

i. **How many instances of resources are present in the system under each type of a resource?**
A = 12; B = 12; C = 8; D = 10;

ii. **Compute the Need matrix for the given snapshot of a system.**
P0 = 2 2 1 1
P1 = 2 1 3 1
P2 = 0 2 1 3
P3 = 0 1 1 2
P4 = 2 2 3 3

iii. **Verify whether the snapshot of the present system is in a safe state by demonstrating an order in which the processes may complete.**
The given snapshot of the system is in safe state. With the available resources the processes will complete their execution with the following sequences: P0-P3-(any combination of P1, P2 and P4)

iv. **If a request from process P1 arrives for (1,1,0,0), can the request be granted immediately?**
Yes, the request (1 1 0 0) may be granted to P1, and the system will remain in safe state. With the available resources the processes will complete their execution with the following sequences: P0-P3-(any combination of P1, P2 and P4)

v. **If a request from process P4 arrives for (0,0,2,0), can the request be granted immediately?**

No, the above said request (0 0 2 0) cannot be granted to P4. In case, if you grant the resources the system will go to un-safe state and all the processes will be in deadlock.

(h) **Briefly discuss the policies to recover from the deadlock.**
Process temination and Resource preemption

(2+2+4+7+2+2+6+2 = 27M)

4. (a) **What are logical and physical addresses of a process? How logical address is converted to physical address? With appropriate figure, discuss how memory management unit (hardware) protects the memory address space of other processes and operating system.**
Logical addresses of a process are generated by CPU, they start from zero for all processes. Physical addresses are related to the address space in the physical memory (Main memory) where the process is actually present (loaded) in the main memory. For each process there is an associated base and limit registers, which caontains the starting physical memory address of the process and total size of the process. The physical address will be generated from the logical address by adding adding the contents of base register to logical address.
The generated physical address of a process should be less than the sum of the contents of base and limit registers. Otherwise it will be treated as illegal memory access, and trap will be generated to OS.

(b) **Discuss 3 different stages of address binding.**
(i) Compile-time, (ii) Load-time and (iii) Execution-time

(c) **Briefly discuss about external and internal fragmentations in the context of contiguous memory space allocation.**
External Fragmentation: In case of contiguous allocation, even though the total available space is more than the required space of a process, but still process will be unable to load into memory, because of non-availability of continuous space required by the process.
Internal Fragmentation: When you allocate a block of memory to a process, the space left-over in the block, after process has loaded into the memory.

(d) **Show the process of paging (conversion of logical address to physical address) with TLBs using neat diagram.**
First the logical address will be checked whether it is in valid logical address space. If it is not the valid logical address, then the process will be terminated. If it is valid logical address then it will be searched first in TLB and then in Page table. From the logical address, page number is identified and check with the entries of TLB. If it is present in TLB, then physical address can be computed by adding offset to frame starting address. If the page is not found in TLB, then from the page table the frame containing the page is identified and desired physical address will be generated by adding the offset to frame start address.

(e) **Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512-MB of physical memory.**

i. **How many entries will be there in a conventional single-level page table and inverted page table?**

Entries in conventional single-level page table $= \frac{2^{32}}{2^{12}} = 2^{20} = 10^6$
Entries in inverted page table $= \frac{2^{29}}{2^{12}} = 2^{17} = 128 \times 10^3$

ii. **What will be the memory requirement for storing these tables?**
Memory for storing single-level page table = number of pages $\times$ *number of bits for representing frame* $= 10^6 \times 17bits = 10^6 \times 3bytes = 3MB$
Memory for storing inverted page table = number of frames $\times$ *number of bits for representing page* $= 128 \times 10^3 \times 20bits = 128 \times 10^3 \times 3bytes = 384KB$

iii. **If a memory reference takes 50 nanoseconds, how long does a paged memory reference take in the context of conventional page table?**
Access time to a paged memory reference take in the context of conventional page table = 2 $\times$ *memory access time = 2* $\times$ *50 = 100 ns*

iv. **If we add TLBs, and 75% of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)**
Access time in the context of TLB + Page Table (TLB hit = 75% and TLB miss = 25%) = (0.75 $\times$ *(50+2)) + (0.25*$\times$*(2+50+50)) = 64.5 ns*

(f) **What is hashed-page table? How address translation (logical to physical) is carried out using hashed-page table?**
If the logical address space is very large (32/64 bit), then paging using conventional page tables will be very complex due to huge memory space requirement for storing the page tables in main memory. To overcome this one of the solutions is use of hashed page tables. Hashed page table size is fixed, and all logical pages are mapped to fixed number of outputs. From the logical address, page number is obtained and then the page is hashed and it produces one of the output. Here in hashing large number of inputs are mapped to small number of outputs, and hence the output entry is related to multiple inputs. Therefore the output is represented using a list of inputs that are mapped to it. Based on this, the logical page is first hashed to one of the outputs and the list attached to the output is searched for the given page-frame mappping. Once the desired frame is found from hashed table, the physical address is determined by adding the offset to the frame start address.

(g) **What is page fault? With appropriate diagram clearly discuss the steps involved in handling the page fault by an operating system.**
If the page associated to a logical address is valid and it is not present in the physical memory (present on the disk), then it is know to be page fault.
Handling the Page Fault:
1. Trap to the operating system
2. Save the user registers and process state
3. Determine the location of the page on the disk
4. Issue a read from the disk to a free frame:
i. Wait in a queue for this device until the read request is serviced
ii. Wait for the device seek and/or latency time
iii. Begin the transfer of the page to a free frame
5. While waiting, allocate the CPU to some other user
6. Receive an interrupt from the disk I/O subsystem (I/O completed)
7. Save the registers and process state for the other user
8. Determine that the interrupt was from the disk
9. Correct the page table and other tables to show page is now in memory

10. Wait for the CPU to be allocated to this process again

11. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

(h) **With appropriate diagram explain the concept of copy-on-write in the context of process creation.**
When a new process is created logical address spaces of both parent and child are mapped to single phisical address space. There after, if any of the process (either parent or child) what to modify data, then page/pages corresponds to that modification will be first copied into free frame/frames and then the required modifications will be performed on the copied page and these copied pages are updated in the page table of the corresponding process.

(i) **Consider the following page reference string: 7,2,3,1,2,5,3,4,6,7,7,1,0,5, 4,6,2,3,0,1. Assume demand paging with 3 frames, how many page faults would occur for the following replacement algorithms?**

i. **LRU replacement**
18 page faults

ii. **FIFO replacement**
17 page faults

iii. **Optimal replacement**
13 page faults

(j) **What is thrashing? What is the working-set model (WSM) of a process? How do you track the working-set? Comment on the relation between WSM and size of the physical memory in view of thrashing.** *If the processes present in the system (main memory), doesn't have enough pages, then each process generage page fault. These page faults replace the pages with new ones, soon after that again page faults occur requesting for the just recently removed pages. This continues and CPU will be mostly idle waiting for the new pages and most of the time will be spent for disk I/O. With this OS will load new processes from disk to increase the multiprogramming for enhancing the CPU utilization, and it further worsen the overall performance. This phenomenon is known to be thrashing.*
Working-set model: The set of pages referred in the recent time interval is know as working set of a process at that time. Ex: 0,2,0,0,0,4,4,4,2,2 for this history, the working-set model is {0,2,4}.
Tracking the working set: Tracking the working set means as per the present requirment of page references, if page fault occurs, then to decide which page from the present WS has to be replaced, is based on the page-usage in the previous WSs
Relation between WSM and size of the physical memory: If the size of sum of optimal WSMs of all processes present in memory is greater than the size of physical memory, then thrashing will result. So to avoid thrashing we need to ensure that size of sum of optimal WSMs of all processes present in memory should be less than the size of physical memory. Suppose, if there is a severe thrashing, then to maintain the above relation to reduce thrashing, we need to swapout some processes to disk.

(3+3+2+2+7+2+4+2+6+4 = 35M)

Q 2.(g)

Process	Burst time (ms)	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

<u>FCFS</u>

2	1	8	4	5
P_1	P_2	P_3	P_4	P_5

$w_1 = 0$ $\quad T_1 = 2$

$w_2 = 2$ $\quad T_2 = 3$

$w_3 = 3$ $\quad T_3 = 11$

$w_4 = 11$ $\quad T_4 = 15$

$w_5 = 15$ $\quad T_5 = 20$

$$w_{av} = \frac{2+3+11+15}{5} = \frac{31}{5}$$

$$= 6.2 \text{ ms}$$

<u>SJF</u>

1	2	4	5	8
P_2	P_1	P_4	P_5	P_3

$w_1 = 1$ $\quad T_1 = 3$

$w_2 = 0$ $\quad T_2 = 1$

$w_3 = 12$ $\quad T_3 = 20$

$w_4 = 3$ $\quad T_4 = 7$

$w_5 = 7$ $\quad T_5 = 12$

$$w_{av} = \frac{1+12+3+7}{5}$$

$$= \frac{23}{5} = 4.6 \text{ ms}$$

Non-preemptive priority (NPP)

8	5	4	2	1
P3	P5	P4	P1	P2

8	5	2	4	1
P3	P5	P1	P4	P2

$w_1 = 13 (17)$ $T_1 = 15 (19)$

$w_2 = 19$ $T_2 = 20$ $w_{av} = \frac{13+19+15+8}{5}$

$w_3 = 0$ $T_3 = 8$

$w_4 = 15 (13)$ $T_4 = 19 (17)$ $= \frac{55}{5} = 11$

$w_5 = 8$ $T_5 = 13$

$w'_{av} = \frac{17+19+13+8}{5} = \frac{57}{5} = ~~[illegible]~~$
$= 11.4$

RR

2	1	2	2	2	2	2	2	2	1	2
P1	P2	P3	P4	P5	P3	P4	P5	P3	P5	P3

$w_1 = 0$ $T_1 = ~~3~~ 2$

$w_2 = 2$ $T_2 = 3$

$w_3 = 3+4+4+1 = 12$ $T_3 = 20$

$w_4 = 5+4 = 9$ $T_4 = 13$

$w_5 = 7+4+2 = 13$ $T_5 = 18$

$w_{av} = \frac{2+12+9+13}{5} = \frac{36}{5} = 7.2$

3(g)

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 1 2	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

(i) A — 12 ; B — 12 ; C — 8 ; D — 10

(ii) Need Matrix Max − Allocation

	Max	Allocation	Need	Available
P0	4 2 1 2	2 0 0 1	2 2 1 1	3 3 2 1
P1	5 2 5 2	3 1 2 1	2 1 3 1	
P2	2 3 1 6	2 1 0 3	0 2 1 3	
P3	1 4 2 4	1 3 1 2	0 1 1 2	
P4	3 6 6 5	1 4 3 2	2 2 3 3	

(iii) P0 can be completed with the available res[illegible]

Completion	Available
P0	5 3 2 2
P3	6 6 3 4
P1	9 7 5 5
P2	11 8 5 8
P4	12 12 8 10

Safe state with the order P0 − P3 − P1 − P2 − P4

P0 − P3 < P1 − P4 − P2 ; P2 − P1 − P4 ; P2 − P4 − P1

P0 − P3 − P4 − P1 − P2

P0 − P3 − P4 − P2 − P1

3(8) (IV) Request from P_1 process (1 1 0 0)

Available = $\underline{2\ 2\ 2\ 1}$

Process	Allocation A B C D	Max A B C D	Need A B C D
P_0	2 0 0 1	4 2 1 2	2 2 1 1
P_1	4 2 2 1	5 2 5 2	1 0 3 1
P_2	2 1 0 3	2 3 1 6	0 2 1 3
P_3	1 3 1 2	1 4 2 4	0 1 1 2
P_4	1 4 3 2	3 6 6 5	2 2 3 3

Allocate to P_0 & complete = 4 2 2 2

complete P_3 = 1 3 1 2 + 4 2 2 2 = 5 5 3 4

complete P_1 = 5 5 3 4 + 4 2 2 1 = 9 7 5 5

complete P_2 = 9 7 5 5 + 2 1 0 3 = 11 8 5 8

complete P_4 = 11 8 5 8 + 1 4 3 2 = 12 12 8 10

Yes, the request of 1100 can be granted to P_1.

After that system is still in safe state $P_0 - P_3 - P_1 - P_2 - P_4$

$P_0 - P_3 -$ (all combinations of P_1, P_2, P_4 are in safe state)

(V) Request from P_4 (0 0 2 0)

Process	Allocation	Max	Need	Available
P_0	2 0 0 1	4 2 1 2	2 2 1 1	3 3 $\boxed{0}$ 1
P_1	3 1 2 1	5 2 5 2	2 1 3 1	
P_2	2 1 0 3	2 3 1 6	0 2 1 3	
P_3	1 3 1 2	1 4 2 4	0 1 1 2	
P_4	1 4 5 2	3 6 6 5	2 2 1 3	

Request to P_4 cannot be granted. If we grant ~~all~~ all processes will be in deadlock.

(4)(e)

(i) # entries in conventional single-level page table $= \frac{2^{32}}{2^{12}} = 2^{20} \approx \underline{10^6}$

entries in inverted page table

$= \frac{2^9 \times 2^{20}}{2^{12}} = 2^{17} = \underline{\underline{128 \times 10^3}}$

(ii) Memory for storing single-level page table

$= 10^6 \times$ # bits for no. of frame #

$= 10^6 \times 17$ bits $= 10^6 \times 3$ bytes $= \underline{\underline{3\text{ MB}}}$

Memory for storing inverted page table

$= 128 \times 10^3 \times$ # bits for no. of page #

$= 128 \times 10^3 \times 20$ bits $= 128 \times 10^3 \times 3$ bytes

$= \underline{\underline{384\text{ KB}}}$

(iii) Access time to access page memory ref in the context of conventional single level page table $= 50 \times 2 = \underline{\underline{100\text{ ns}}}$

(iv) Access time in the context of TLB + PT (75% + 25%)

$= 0.75 \times (50 + 2) + 0.25(50 + 2 + 50)$

$= \underline{\underline{64.5\text{ ns}}}$

4 (J) 7 2 3 1 2 5 3 4 6 7 7 1 0 5 4 6 2 3 0 1

LRU

7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
	7	2	3	1	2	5	3	4	6	6	7	1	0	5	4	6	2	3	0
		7	2	3	1	2	5	3	4	4	6	7	1	0	5	4	6	2	3

18 - Page Faults

~~FCFS~~

7	7	7	2	3	1	2	5	3	4	4	6	7	1	0	5	4	6	2	3
	2	2	3	1	2	5	3	4	6	6	7	1	0	5	4	6	2	3	0
		3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1

~~18 - Page Faults~~

optimal

7	7	7	2	3	1	1	1	1	1	1	1	1
	2	2	3	1	5	5	5	5	0	0	0	0
		3	1	5	4	6	7	0	4	6	2	3

13 - page faults

FCFS

7	7	7	2	2	3	3	1	5	4	4	6	7	1	0	5	4	6	2	3
	2	2	3	3	1	1	5	4	6	6	7	1	0	5	4	6	2	3	0
		3	1	1	5	5	4	6	7	7	1	0	5	4	6	2	3	0	1

17 - page faults.