21st CSEC – Past Year Paper Solution (2017 – 2018 Semester 2)
CE/CZ 2005 – Operating Systems

1 **(a)** **(i)** False, especially if these instructions do not directly access hardware. If all program instructions are run in monitor mode, any application crash can potentially crash the entire OS. This would not happen in user mode; applications can run in isolation and crashes would be limited to that application.

(ii) True. Even though response time is minimized which is a main goal, priority cannot be given to any process if required. Choosing an appropriate fixed time quantum may also be difficult; a quantum that is too long/short affects the system efficiency.

(iii) False. Preemptive CPU scheduling means that the CPU can be taken away from a running process at any time a process changes state (e.g. timer interrupt/event wait).

(iv) False. Multi-programming still allows for several processes in main memory and context switching between these processes within each of the cores for concurrency.

(v) True. An interrupt is generated per bit of data transferred. If there are many of such I/O operations, there will be constant switching to kernel mode, causing CPU overhead.

(b) **(i)**

Scheduling Algorithm	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
RR	P1	P1	P1	P3	P3	P4	P4	P4	P1	P1	P1	P2	P2	P2	P4	P4	P4	P1	P1	P1	P2	P2	P1
SJF	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P3	P3	P2	P2	P2	P2	P2	P4	P4	P4	P4	P4	P4
SRTF	P1	P1	P3	P3	P2	P2	P2	P2	P2	P4	P4	P4	P4	P4	P4	P1	P1	P1	P1	P1	P1	P1	P1

Turnaround Time = Completion Time – Arrival Time

<u>Round-Robin</u>
Average Turnaround Time = [(23 – 0) + (22 – 3) + (5 – 2) + (17 – 2)] ÷ 4 = 15

<u>Non-preemptive Shortest Job First</u>
Average Turnaround Time = [(10 – 0) + (17 – 3) + (12 – 2) + (23 – 2)] ÷ 4 = 13.75

<u>Shortest Remaining Time First</u>
Average Turnaround Time = [(23 – 0) + (9 – 3) + (4 – 2) + (15 – 2)] ÷ 4 = 11

(ii) Priority-based Scheduling, with P1 having the highest priority, followed by P3, P2 and lastly P4. The Gantt Chart will be identical to SJF's Gantt Chart and have the same average turnaround time.

(c) Improving CPU utilization will require the CPU to be kept as busy as possible and this can be done by maximizing the percentage of time during which there is some process in the "running" state.

The medium-term and long-term scheduler should ensure that sufficient processes are in main memory so that when the current process exits the "running" state (e.g. due to an I/O or event wait), there are processes for the short-term scheduler to select for context switching. If round-robin scheduling algorithm is used, increasing the time quantum can also decrease the number of context switches done and maximize time spent on the processes.

(d) **(i)** Reading a number from the keyboard is an I/O operation, requires a switch to kernel mode.
Computing an arithmetic expression can be done after a switch back to user mode.
Storing the result into a shared memory location requires a switch to kernel mode as memory protection is done through loading base and limit registers which are privileged instructions.

(ii) Process P1 starts by reading a number from the keyboard and goes into "waiting" state since it issues an I/O request. Context switch to Process P2 can be done for P2 to compute the arithmetic expression. There could be a context switch back to Process P1 when I/O request is done and P2 has finished executing, for P1 to store the result into the shared memory location.

2 **(a)** **(i)** True. A context switch between S.value— and the checking of the S.value can lead to an inconsistent S.value and potentially cause a mutual exclusion violation.

(ii) False. If there is no process in the critical section and there exists a process that can cut the queue into the critical section all the time, it means that selection of the next process is not postponed indefinitely and hence progress is not violated.

(iii) False. Windows ignores the deadlock problem and pretends they never occur in the system, since deadlock prevention/detection incurs constant overhead and reduces performance.

(iv) True. Busy waiting may be better in performance if the critical sections of the processes are short since there is no context switch overhead and wasted CPU cycles are minimal.

(v) False. If there exists at least one safe completion sequence, the system is in safe state even if there are other unsafe completion sequences.

(b) **(i)** Yes, as there exists a safe completion sequence of P2 → P3 → P1. Initially Available > P2's Need, after P2's execution, Available = (3,4,3). Updated Available > P3's Need, after P3's execution, Available = (4,4,5). Updated Available > P1's Need, P1 executes.

(ii) If request is granted, Available = (0,1,0) and P3's Need = (1,0,0). Since the updated Available cannot fulfill all 3 processes' Need, there is no safe completion sequence and system will be in unsafe state. Hence the request should not be granted.

(c) (i) **Problem 1**

The producer process acquires mutex first before trying to acquire *empty*. If the buffer is already full and a producer process tries to acquire *empty*, it will wait for an item in buffer to be consumed. However, the consumer process cannot do so because it is unable to obtain *mutex* which is already held by the consumer process, leading to a deadlock.

Problem 2

The consumer process acquires mutex first before trying to acquire *full*. If the buffer has no items and a consumer process tries to acquire *full*, it will wait for an item to be added to buffer. However, the producer process cannot do so because it is unable to obtain *mutex* which is already held by the producer process, leading to a deadlock.

(ii) Maximum value for *mutex* is 1 since only one process can gain exclusive access to the buffer at any one time.

Minimum value for *mutex* is -9, the first process decrements *mutex* to 0 and gains exclusive access to the buffer; the remaining 9 processes get blocked after decrementing the *mutex*.

(iii) Additional Variable

noOfItems,
initialized to 10

Producer Process

```
item p;
while (1) {
   wait(empty);
   wait(mutex);
   produce an item p;
   add p to buffer;
   noOfItems++;
   signal(mutex);
   signal(full);
}
```

Consumer Process

```
item c;
while (1) {
   wait(full);
   wait(mutex);
   while (noOfItems != 0) {
      c = an item from buffer;
      consume item c;
      noOfItems--;
   }
   signal(mutex);
   signal(full);
}
```

3 (a) (i) False. Logical address needs to be converted to physical address through a relocation register inside the memory management unit if execution time binding is used.

(ii) False. Paging divides physical/logical memory into fixed-sized blocks and will thus suffer from internal fragmentation (on average half a page is unused for a process).

(iii) False. Effective memory access time also depends on the hit ratio, if the hit ratio is very low, the overall effective memory access time could be increased due to frequent TLB misses.

(iv) False. With segmentation, segments of a process can be allocated in a non-contiguous manner.

(v) False. Reference bits are used for LRU approximation algorithms such as the second-chance (clock) algorithm to determine if a page has been referenced. If the reference bit is set, there is a possibility that the page has been modified and hence needs to be written back.

[Note the difference between reference bit VS modify/dirty bit!]

(vi) True. If there is a low level of CPU utilization even though there is a high level of multiprogramming, it means processes are busy bringing pages in and out, which is thrashing.

(b) (i) Size of physical memory = 1024 × 8 = 8192 bytes
Number of bits needed for physical address = (8192) = 13

(ii) Number of bits for offset = (1024) = 10

Logical address: 1052 (dec) = $\underline{1}$00 0001 1100 (Page 1 → Frame 7)
Physical address: $\underline{1\ 11}$00 0001 1100 = 7196 (dec)

Logical address: 2080 (dec) = $\underline{10}$00 0010 0000 (Page 2 → No Valid Frame)
Physical address: No corresponding physical address.

(iii) Fixed allocation policy used → Local replacement must be used → 3 pages allocated
LRU Algorithm replaces page not used for longest period → Use "Time Last Accessed"

First page reference with page number 2, causing eviction of page 1.
Second page reference with page number 1, causing eviction of page 3.
Third page reference with page number 3, causing eviction of page 0.

(iv) No, as pages evicted using FIFO is different from pages evicted using LRU. FIFO evicts pages loaded in for longest period, with page 0 loaded in earliest, followed by 3 and 1.

First page reference with page number 2 still causes page fault but evicts page 0 instead.
Second page reference (page number 1) will not cause page fault, maps page 1 to frame 7.
Third page reference (page number 3) will not cause page fault, maps page 3 to frame 4.

(c) (i) Locality refers to the tendency for processes to refer to pages in a localized manner and consist of both temporal and spatial locality.

Temporal locality occurs when locations referred to previously are likely to be referred to again (e.g. iterating through loops). Spatial locality occurs when code and data are clustered physically (e.g. reference instructions in sequence).

(ii) The fragment on the left will exhibit better locality. If the array elements are stored according to rows, each row of elements will be stored within a page.

Using i as the outer loop means that the corresponding j elements in the current loop iteration will be located within fewer number of pages and hence stronger spatial locality. If j is used as the outer loop, each of the corresponding i elements will be in separate pages and thus require more pages to be loaded into memory, leading to higher number of page faults.

4 (a) (i) Acyclic graph directory structure allows for file sharing through symbolic/hard links, with directories able to "share" subdirectories/files. This is useful for users to be able to collaborate on certain documents.

(ii) Read and execute rights only. (Group Permission Bits: r – x)

(iii) Buffering is more suitable for downloading a file from the Internet to a hard disk due to the mismatch in both device speed and transfer size (packets of data vs blocks of data).

Caching is more suitable for reading a file block from hard-disk repeatedly, to reduce the time required as compared to repeatedly retrieving the same file block from hard-disk which is slow.

(iv) If the block size is too small, a file will need to consist of more blocks and thus leads to a slower data rate. Time utilization has been compromised to allow for better space utilization (lesser internal fragmentation).

(b) (i) Starting byte resides in $37000 \div 512 = 72.265 \rightarrow$ 73th block
Ending byte resides in $38000 \div 512 = 74.219 \rightarrow$ 75th block
Logical blocks that need to be addressed are 73th to 75th blocks.

(ii) 1 single indirect pointer block can contain $512 \div 8 = 64$ pointers
10 direct pointers and 1 single indirect pointer can point to a total of $10 + 64 = 74$ blocks
$\therefore$ 73th, 74th block pointed to by single indirect, 75th block pointed to by double indirect

Since only inode is in memory,
 1 disk read for single indirect pointer
 2 disk reads for 73th and 74th block
 2 disk reads for double indirect pointer + block pointed to by double indirect pointer
 1 disk read for 75th block
Total of $1 + 2 + 2 + 1 = 6$ disk reads needed.

(iii) 10 direct pointers can accommodate $512 \times 10 = 5120$ bytes
1 indirect pointer can accommodate $64 \times 512 = 32768$ bytes
1 double indirect pointer can accommodate $64 * 64 \times 512 = 2097152$ bytes
Maximum file size $= 5120 + 32768 + 2097152 = 2135040$ bytes $= 2.04$MB

(c) **LOOK Scheduling Algorithm**

Cylinder Serviced	Time AFTER Servicing (starting from 0 units)	Cylinders in Queue at Time AFTER Servicing
160	30u	40, 170
170	30u + 30u = 60u	40, 100
100	90u + 60u = 150u	40, 80
80	40u + 150u = 190u	40, 60
60	40u + 190u = 230u	40
40	40u + 230u = 270u	50
50	30u + 270u = 300u	-

Request servicing sequence: 160 → 170 → 100 → 80 → 60 → 40 → 50

C-LOOK Scheduling Algorithm

Cylinder Serviced	Time AFTER Servicing (starting from 0 units)	Cylinders in Queue at Time AFTER Servicing
160	30u	40, 170
170	30u + 30u = 60u	40, 100
40	150u + 60u = 210u	100, 80, 60
60	40u + 210u = 250u	100, 80
80	40u + 250u = 290u	100, 50
100	40u + 290u = 330u	50
50	70u + 330u = 400u	-

Request servicing sequence: 160 → 170 → 40 → 60 → 80 → 100 → 50

Solver: Chong Shen Rui (schong031@e.ntu.edu.sg)