Solver: Zhou Jingyuan

Email Address: zhou0235@e.ntu.edu.sg

1. (a)
   (i) False. Privileged instruction can only be run in kernel mode.
   (ii) True. Almost all OS use preemptive scheduling.
   (iii) True. DMA usually used for transmitting large amount of data, whereas mouse IO are lightweight.
   (iv) True. Multiprocessing refer to the hardware capability to run multiple process, whereas the multi-programming refers to the software capability to run multiple program.
   (v) False. CPU could be non-preemptive; the stage change is only due to an IO of the process.

   (b)
   (i) When the IO burst occurred, the system will first save the state of P2 into PCB_2, and then reload the state from PCB_1, where the system resumes the execution of P1.

   (ii) Shortest job first: shortest job first executes jobs with lowest <u>CPU burst</u>.

Time	1	2	3	4	5	6	7	8	9
Running Process	P2	P2	P1	P1	P2	P2	P1	P1	P1

Note at time 5, P2 finishes its IO and returned to the ready queue, since it only left with 2 CPU burst (P1 has 3 left), it takes over the CPU and runs.
Average turn around time= (8+9)/2 = 8.5
Round-Robin:

Time	1	2	3	4	5	6	7	8	9	10
Running Process	P1	P1	P2	P2	P1	P1	P2	P2	P1	P2ends

Note that in this case, P2 ends at time= 10. Although P2 does not use any CPU burst at time 10. It will not terminate during time 10. Average turn around time = (10+9)/2 = 9.5

(c)
(i) There are multiply lines have mode switch, namely:
line 7 & line 11: Context switch is necessary as the printf function tries to access IO, as it has to be done in kernel mode.
Line 6: forking a process need to allocate pointers and spaces, which also need to be done in the kernel mode.
(ii) The question asked about <u>the process</u>, assume it ask the mother process.
Text: does not change, as the code it self does not change.

Data: empty. As there is no global variable. The pid variable is only available in function scope and stored in the stack together with the function.
Heap: empty, as you don't see any new keyword, forked process are not stored in the heap, they are completely new process.
Stack: pushed in with f1(5) f1(4) … f1(0) function calls, and then popped out as the function completes

Additional Challenging Questions:
How many additional process will be created?
How many *done* will this code segment print?
(Hard) How many lines will this code segment print?
*Note: fork() create and run a new process with identical state of the caller.
Answer available in the very end of this solution.

2. (a)
(i) True, as it is the definition of safe state.
(ii) True, busy waiting need to constantly check the state of the variable
(iii) True, simple batch system occurred in 1960's where user submit "tapes" of there operation to the computer. Deadlock only occurred when multi-thread/process need to access same resources. Simple batch system does not have the notion of wait to acquire the resources.
(iv) True, one of the four condition for deadlock is no preemption
(v) False, as in the previous statement, the system could use preemption to avoid deadlock.
(b) minimum value is 2. If x = 2, the process can be finished in the sequence of P2, P1, P0. Whereas if x = 1, the only process can finish is P2, and after that, we have 1 of ABCD, and non of process 1 nor 0 can proceed.
(c)
A: semaphore hydrogenSemaphore;
B: signal(hydrogenSemaphore)
C: wait(hydrogenSemaphore);wait(hydrogenSemaphore);
(d) This is a common question in exam. Consider the following approach:
ID those command:

ID	P1	ID	P2
1	Y=1	5	X = 1
2	Y = Y+Z	6	X = X+1
3	Z = Y + 1	7	X = X - Y
4	Y = Z – Y	8	Z = X + Z

For the execution sequence, the following condition must meet:

1. 1<2<3<4
2. 5<6<7<8
3. 1<7(semaphore S)
4. 7<4(semaphore Q)

now we try to find all possible combination of sequence, and before going into many possible permutation, we could save time by doing the following:
1) since P1 does not use X, 5 & 6 can combine into X=2(ID=5)

2) If a set of statement are independent, i.e. the variable being changed is not used by the set of statement, they can be run in any order.
3) 1 & 5 are independent, i.e. order 1574 and 5174 are the same stuff
4) 2&5 are independent. 3&5， 4&6， 3&7 are independent

(you could do this by fixing 1,5,7,3, 4, and fill in 2, 8):

All possible sequences & results:

Seq	X	Y	Z
1 5 2 7 8 3 4	1	1	2
1 5 2 7 3 8 4	1	2	3
1 5 2 7 3 4 8	1	3	2
1 5 7 8 2 3 4	1	1	3
1 5 7 2 3 8 4	1	3	4
1 5 7 2 3 4 8	1	1	2

3. (a) First-fit: 20KB; Best-Fit: 14KB; Worst-Fit: 40KB

(b)

(i)

Id	Frame	Valid bit
0		0
1	2	1
2		0
3	1	1
4		0
5		0
6	0	1
7	3	1

(ii) 001 101101011 → 10 101101011

010 110110101 → no corresponding physical address

(iii) 2^9 = 512 bits

(c) Thrashing: If a process does not have “enough” pages, the page-fault rate is very high. This leads to:

– Low CPU utilization

– OS thinks that it needs to increase the degree of multiprogramming

– Another process is added to the system – CPU utilization drops further

i.e. a process is busy bringing pages in and out (no work is being done)

Solution Two approaches are used:

1) Working-Set Model - During the lifetime of the process, references are confined to a subset of pages

2) Page-Fault Frequency - Establish “acceptable” page-fault rate

– If actual rate of a process is too low, remove a frame from that process

– If actual rate too high, give that process a frame

(d)(i)

12 page faults

(ii) same as FIFO

0	1	2	3	4	5	0	1	2	3	4	5
F	F	F	F	F	F	F	F	F	F	F	F

(iii) in this special case, we could use a modified clock algorithm where instead of iterating from 0->n then jump to 0, this modified algorithm loop through 0 to n then loop back to 0.

0	1	2	3	4	5	0	1	2	3	4	5
F	F	F	F	F	F			F	F	F	F

In this case there will be 10 faults.
Notice that you should not answer unrealistic page replacement algorithm even if it give less page fault, such as the optimal algorithm: replace the farthest page (you cannot predict the future page demand) nor replace the newest page (unrealistic as the memory content almost cannot change after the memory is full)

4. (a)
   (i) True
   (ii) True, owner: rwx; group: rw; all: r
   (iii) False, block allocation will cause fragmentation, linked allocation won't.
   (iv) False, non-blocking will keep the process in running state
   (v) False, RAID 1 consists of an exact copy (or mirror) of a set of data on two or more disks;
   (vi) True, buffering avoids constant read/write to external disks
   (b) Problems:
   – Waste of space (similar to dynamic storage allocation of main memory)
   – Finding hole big enough using First-fit (faster) or Best-fit may result in external fragmentation
   – File space constricted by size of hole, so may later have to move to a bigger hole
   – If instead needed file space is overestimated -> internal fragmentation
   How index file allocation method overcomes these problems:
   Each file has an index block which contains all pointers to the allocated blocks. Directory entry contains the block number of the index block.
   – Support random access.
   – Dynamic storage allocation without external fragmentation (similar to the allocation of main memory using paging scheme).
   (c)
   FCFS
   [(120-60)+(120-30)+(130-30)+(130-40)+(180-40)+(190-180)]*2+10*6 = 1040ms
   SSTF
   Serving sequence: 40 30 120 130 180 190
   Time = [(60-30)+(190-30)] * 2+ 10 *6 = 440 ms
   SCAN
   Serving sequence: 120 130 180 190 40 30
   Time = [(199-60)+(199-30)] *2 +10*6 = 676 ms

   Challenge Question (answer):
   31 additional process will be created: f(5) create 1 new process, and called *two* f(4). each f(n) create 1 new process and call 2 f(n-1) … total new process = 1+2+4+8+16.
   32 done will be printed 31+1.

Total 94 lines will be printed. 32 done, 31 new pid and 31 of 0(child process will print 0 as pid =0)

reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺