COMP 346 A3 - Theory

1)

i.

a) Relocatable programs are programs that can be accessed dynamically. That is it can be accessed/loaded into memory in parts, and not the entire process all at once.
b) A program is relocatable if it can be stored and accessed by jumping locations in memory via relocation registers, ranging from base to limit register.
c) In terms of memory management it is helpful to have programs be relocatable, so that they don't need to be entirely loaded into memory at once, only the parts of the process that need to be used will be loaded into memory, in order to save on memory.

ii. Small page sizes might result in more frequent external fragmentation and slower access time and more context switches. Large page sizes could result in less external fragmentation and quicker access times but larger internal fragmentation.

iii. Paging offers speed and saves on memory. While segmentation does not necessarily save on memory or speed. Paging is not visible to a programmer, but, segmentation requires a programmer to be aware of the segments in his code.

iv. Segmentation offers security of individual segments of a program.

2)

mem access = 1 ms
TLB access = 0.2 ms
page req =
    20% page fault
    98% page hits
page fault = 20 ms
page hits =
    80% TLB hits
    20% TLB misses

Effective Memory Access Time
EAT = 2 + ε - α
α = 0.80
ε = 0.2 ms
mem access = 1 ms
EAS = 2 + 0.2 - 0.8 = 1.4 ms

3)
R = {0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 6, 7, 6, 7, 0, 1, 2, 3, 4}

a) Least Recently Used (LRU)

P	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F0	0*	0	0	0	0	0	0	0	0	3*	3	3	3	3	0*	0	0	3*	3
F1		1*	1	1	1	1	1	1	1	1	6*	6	6	6	6	1*	1	1	4*
F2			2*	2	2	2	2	2	2	2	2	7*	7	7	7	7	2*	2	2

Page faults = 11

b) Belady Optimal Algorithm

P	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F0	0*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1*	1	1	1
F1		1*	1	1	1	1	1	1	1	1	1	7*	7	7	7	7	2*	2	2
F2			2*	2	2	2	2	2	2	3*	6*	6	6	6	6	6	6	3*	4*

Page faults = 10

c) Working Set Model (Δ=3)

P	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F0	0*	0	0	0	0	0	0	0	0	3*	3	3	3		0*	0	0	3*	3
F1		1*	1	1	1	1	1	1	1	1	6*	6	6	6	6	1*	1	1	4*
F2			2*	2	2	2	2	2	2	2	2	7*	7	7	7	7	2*	2	2
ws	0	1	2	{0,1,2}															
ws	{1,2,3}																		
ws	{2,3,6}																		
ws	{3,6,7}																		
ws	{6,7}																		
ws	{6,7,0}																		

ws	{7,0,1}			
ws	{0,1,2}			
ws	{1,2,3}			
ws	{2,3,4}			

Page faults = 11

4)

a) An advantage of a page table on the CPU rather than in MM is that CPU memory is significantly faster to access.
b) A disadvantage is that even though it take up little memory, it is a significant amount of memory usage in terms of CPU memory.

5) In global allocation a process resulting in page fault can select a replacement frame from the SET OF ALL frames. In local replacement a process resulting in page fault can ONLY select from its OWN frames for replacement. An advantage of global over local replacement has it that it allows high priority processes to increase frame allocation at the expense of low priority process frames. Because the behavior of all external process frames can affect the way it behaves this results in varying execution times.

6)

i) If TPF > TFS than page faults occur less often than it takes it to service a page fault which can result in the best multiprogramming.

ii) If TPF < TFS than page faults take longer to service than they occur which can result in terrible multiprogramming.

iiI) If TPF = TFS then it will perform slightly better than if it were less but page faults will still occur fairly often.

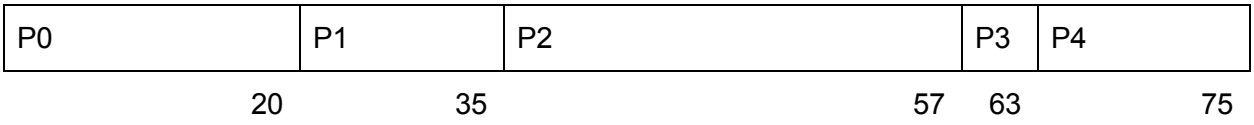
7)

Process ($P_{i)}$	Service Time [$T(P_i)$]	Priority
0	20	3
1	15	1
2	22	3
3	6	4

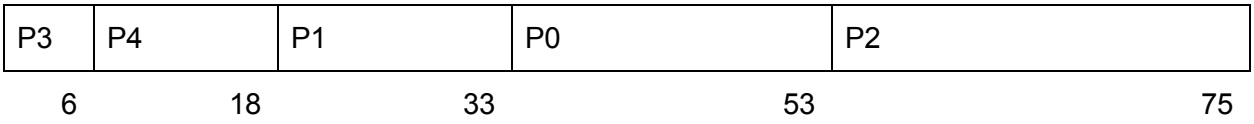
4	12	2

a) Gantt Charts

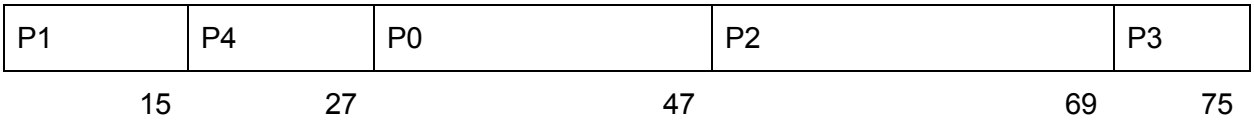
First Come First Serve (FCFS)



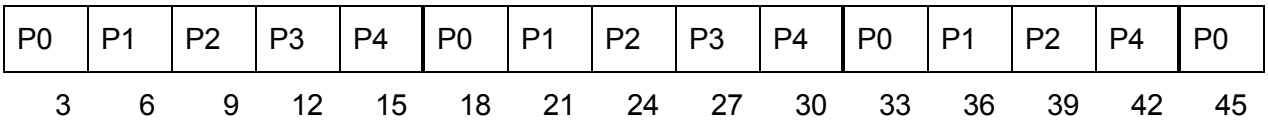
Shortest Job First (SJF)

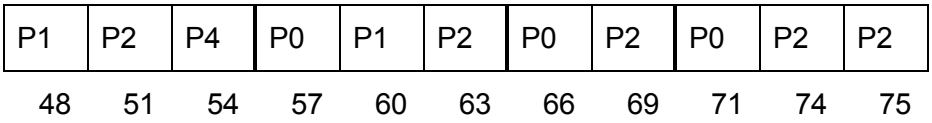


Priority



Round Robin (RR)(quantum=3)





b)

Waiting Times

	$W(P_0)$	$W(P_1)$	$W(P_2)$	$W(P_3)$	$W(P_4)$
FCFCS	0	20	35	57	63
SJF	33	18	53	0	6
Priority	27	0	47	69	15
RR	(69-6*3) = 51	(57-4*3) = 45	(P0+3) = 54	(24-1*3) = 21	(51-3*3) = 42

c)

Response Times

	$W(P_0)$	$W(P_1)$	$W(P_2)$	$W(P_3)$	$W(P_4)$
FCFCS	0	20	35	57	63
SJF	33	18	53	0	6
Priority	27	0	47	69	15
RR	0	3	6	8	12

d)

Turn Around Times

	$W(P_0)$	$W(P_1)$	$W(P_2)$	$W(P_3)$	$W(P_4)$
FCFCS	20	35	57	63	75
SJF	53	33	75	6	18
Priority	47	15	69	75	27
RR	71	60	75	27	54

8)

An advantage is that the system does not need to wait for I/O once the system is done it is kicked out. A disadvantage however is that the file could cause starvation due to poorly written code. And even worse it could result in deadlock.

9) Preemptive scheduling allows a process to yield the CPU to another process.
Non-preemptive does not allow this, a process will remain in the CPU until it is finished.

a) Strict non-preemptive is unlikely in a batch and timesharing system, because non-preemptive scheduling does not allow for priority and context switching before a process terminates. In an environment that has batch and timesharing where process need to be switched out of the CPU often this type of scheduling would be very slow.
b) A quantum size that is too small in pure RR will be very slow because it will context-switch very often. A large quantum size will result in FIFO scheduling, thus increased response time, and lowering the degree of multiprogramming.

10) The advantage of different scheduling values at different levels in feedback queueing systems is that it can give higher priority processes more time to run and lower priority processes less time to run. Fairness amongst processes on the same level is good, but not amongst level, as soon as there is priority there is no longer fairness - “Hanna”. Because the processes that take too long are moved down on priority and the processes that have not been

allowed any time move up this prevents starvation in the system. In terms of efficiency it is good considering that it allows for priority.